Natural Language Processing

Info 159/259

Midterm review (Mar 9, 2021)

David Bamman, UC Berkeley

In-class questions:
http://bit.ly/nlpqs

Midterm

- Thursday 3/11, 2:10-3:30pm PST on bCourses. This midterm must be carried out entirely independently!

- You are free to use any of your notes, readings, or lecture material; the midterm will cover all material through 3/4 lecture on parsing.

- Questions will resemble what you have already experienced with the quizzes (e.g., multiple choice) along with some numerical answer/fill-in-the-blank questions as well. Have your calculator ready!

- You can expect more attention to the topics we cover in lectures and in the homeworks, but everything in the lectures and readings is fair game.

Big ideas

- Classification
    - Naive Bayes, Logistic regression, feedforward neural networks, CNN, BERT
- Where does NLP data come from?
    - Annotation process
    - Interannotator agreement
- Language modeling
    - Markov assumption, featurized, neural
- Probability/statistics in NLP
    - Chain rule of probability, independence, Bayes' rule

Big ideas

- Lexical semantics and word representations
    - Distributional hypothesis
    - Distributed representations
    - Subword embedding models
    - Contextualized word representations (ELMO/BERT)
- Evaluation metrics (accuracy, precision, recall, F score, perplexity, parseval)
- Sequence labeling
    - POS, NER
    - Methods: HMM, MEMM, CRF, RNN, BiRNN, BERT
- Trees
    - Phrase-structure parsing, CFG, PCFG
    - CKY for recognition, parsing

Big ideas

- What defines the models we've seen so far? What formally distinguishes an HMM from an MEMM? How do we train those models?

- For all of the problems we've seen (sentiment analysis, POS tagging, phrase structure parsing), how do we evaluate the performance of different models?

- If faced with a new NLP problem, how would you decide between the alternatives you know about? How would you adapt an MEMM, for example, to a new problem?

Can we get a high-level overview of what distinguishes each neural network (ex: CNN, RNN, bidirectional, etc.) and an example/case of when we would use each?

- Feedforward NN (e.g., multi-layer perceptron). Inputs: fixed-dimensional feature vector (e.g., bag of words representation; **not** original word sequence). Output: single prediction for that sequence.

- CNN. Inputs: original word sequence. Performs the same action (convolution) over each window in the original sequence. Learns to identify important ngrams in the original sequence. Can be used for document classification (by adding a softmax layer on top), or to generate a representation for the entire sequence.

Can we get a high-level overview of what distinguishes each neural network (ex: CNN, RNN, bidirectional, etc.) and an example/case of when we would use each?

- RNN. Inputs: original word sequence. Generates a representation for each token in a sequence that is aware of words on the left context (forward RNN), right context (backward RNN) or both contexts (bidirectional RNN). Information from word *i* needs to pass through the hidden states for words [*i*+1, …, *j*-1] to influence word *j*. Can be used for document classification (by adding a softmax layer on top of the final word representation), sequence labeling (by adding a softmax layer on top of each token representation), or generating representations for each token in the sequence.

Can we get a high-level overview of what distinguishes each neural network (ex: CNN, RNN, bidirectional, etc.) and an example/case of when we would use each?

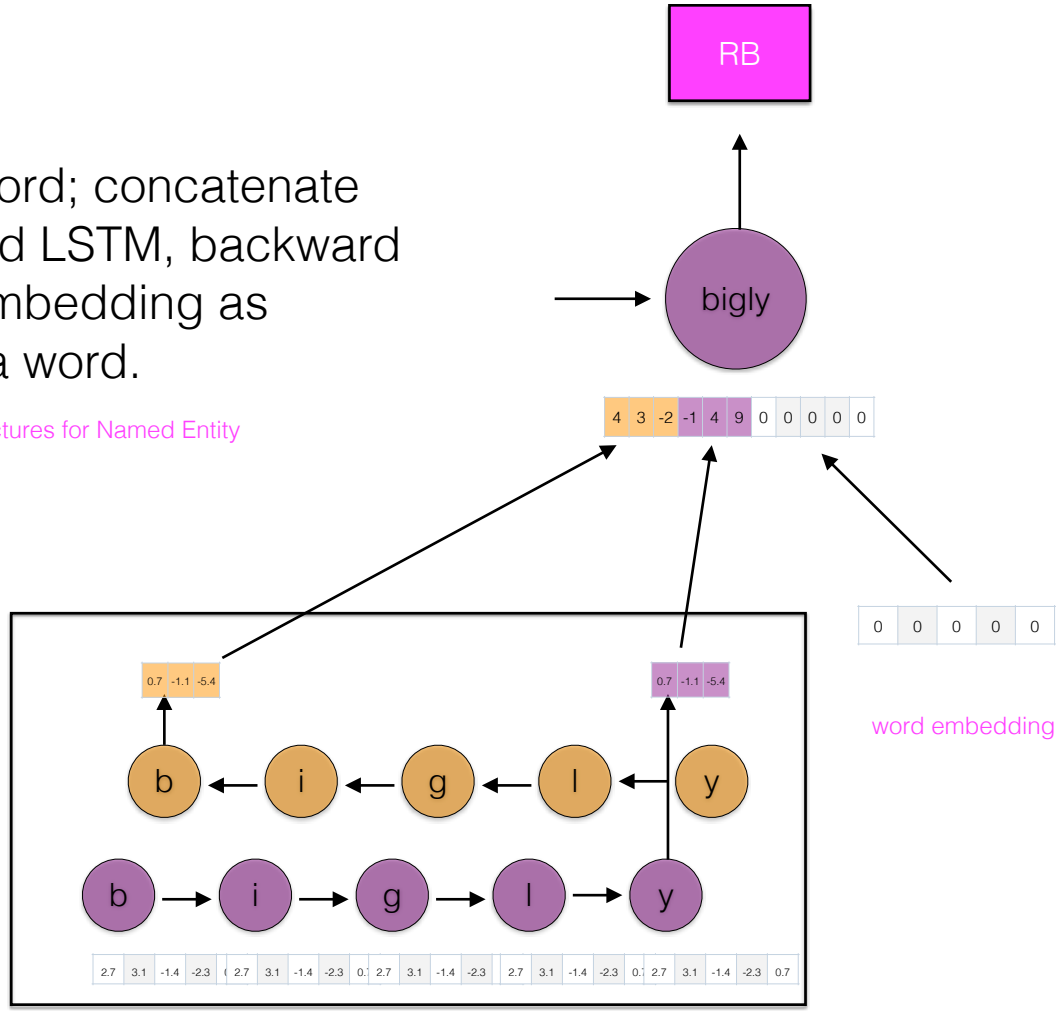
- Transformer. Inputs: original word sequence. Generates a representation for each token in a sequence, for each layer in the transformer, by using attention over all words in the *previous* layer. The representation for word i and layer k has *simultaneous* access to all word representations at layer k-1 (representations not diluted as they are for an RNN). Can be used for document classification (by adding a softmax layer on top of the final word representation), sequence labeling (by adding a softmax layer on top of each token representation), or generating representations for each token in the sequence.

Can we get a high-level overview of what distinguishes each neural network (ex: CNN, RNN, bidirectional, etc.) and an example/case of when we would use each?

	Feedforward NN (e.g. MLP)	CNN	RNN	Transformers
Input	Fixed-dimensional feature vector	Original sequence	Original sequence	Original sequence
Generates representation for each token?	No	No	Yes	Yes
Generates representation for entire input?	Yes (e.g., hidden layer in MLP)	Yes (e.g., concatenation of all filters after maxpooling)	Yes (e.g., hidden state for last token in sequence)	Yes (e.g., [CLS] token in BERT)
Common tasks	Document classification	Document classification, representation learning	Sequence labeling, representation learning	Document classification, sequence labeling, representation learning

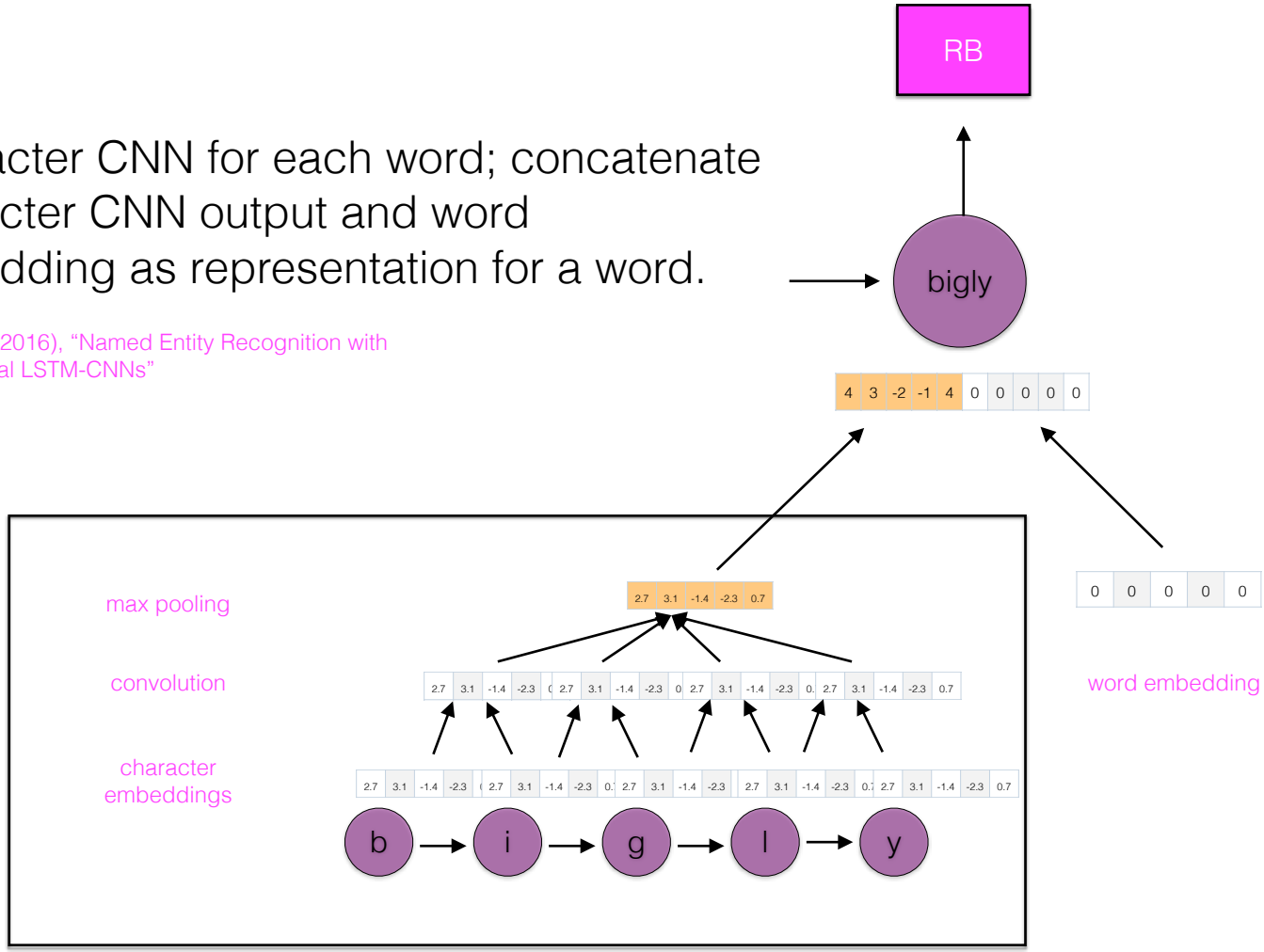
BiLSTM for each word; concatenate final state of forward LSTM, backward LSTM, and word embedding as representation for a word.

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”



Can you summarize when to use each model (classification, language modeling, etc.), and which are discriminative vs. generative?

- Classification: predict a label for each document.
- Language modeling: applications that need information about fluency (autocorrect, translation, speech recognition, OCR); and a general framework for learning representations of words.
- Word representations: learn representations of words (symbols generally) that are sensitive to their context of use, both aggregate context (e.g. word2vec) and local sentence context (e.g., BERT).
- Sequence labeling: predict a label for each token.
- Parsing: predict syntactic structure (generally, any tree structure).

Generative vs. Discriminative models

- Generative models specify a joint distribution over the labels and the data. With this you could generate new data

$$P(X, Y) = P(Y)\, P(X \mid Y)$$

- Discriminative models specify the conditional distribution of the label y given the data x. These models focus on how to discriminate between the classes

$$P(Y \mid X)$$

Naive Bayes

$$P(Y = y|X = x) = \frac{P(Y = y)P(X = x|Y = y)}{\sum_y P(Y = y)P(X = x|Y = y)}$$

Binary logistic regression

$$P(y = 1 \mid x, \beta) = \frac{1}{1 + \exp\left(-\sum_{i=1}^{F} x_i \beta_i\right)}$$

HMM

$$P(x_1, \ldots, x_n, y_1, \ldots, y_n) \approx \prod_{i=1}^{n+1} P(y_i \mid y_{i-1}) \prod_{i=1}^{n} P(x_i \mid y_i)$$

MEMM

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

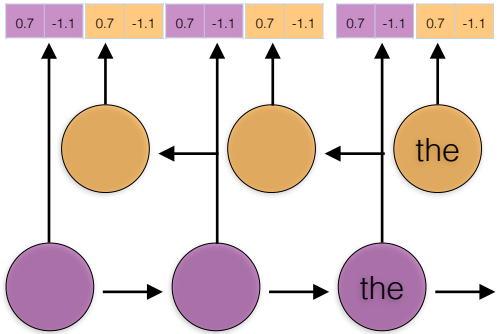
CRF

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

Can you go over BERT vs ELMO, and how attention works?

ELMo

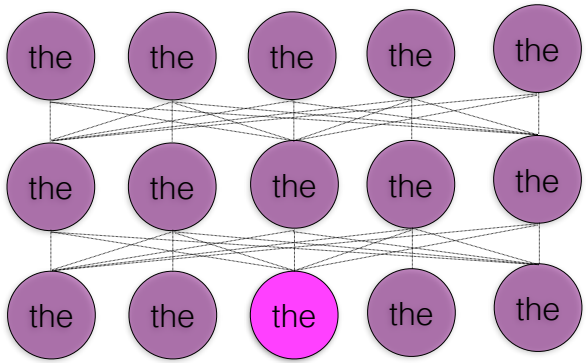
Stacked BiRNN trained to predict next word in language modeling task



Peters et al. 2018

BERT

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



Devlin et al. 2019

ELMo

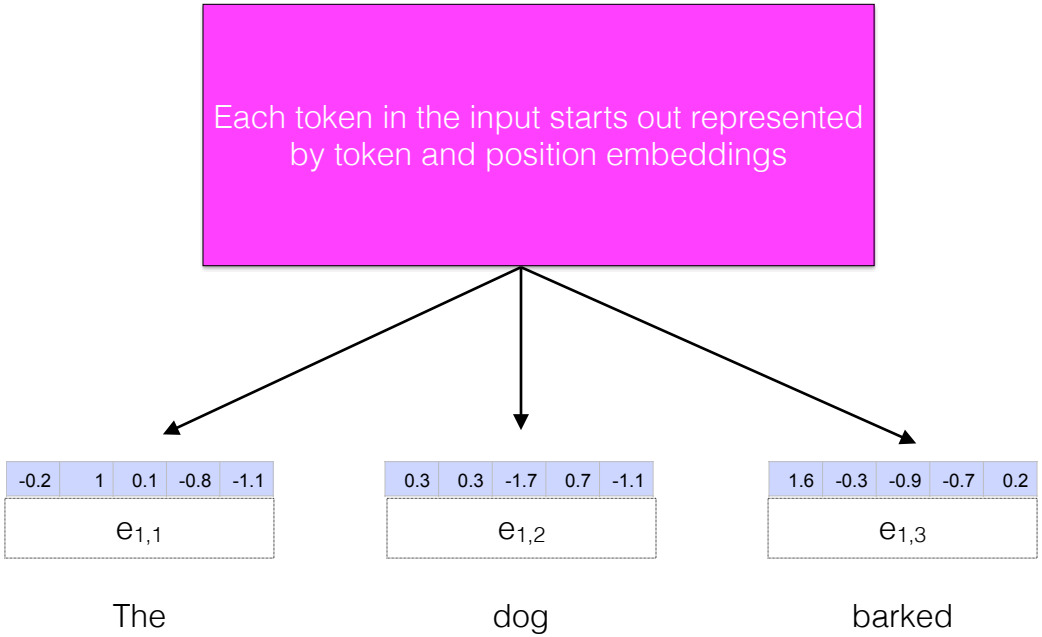
- Peters et al. (2018), “Deep Contextualized Word Representations” (NAACL)
- Big idea: transform the representation of a word (e.g., from a static word embedding) to be sensitive to its local context in a sentence and optimized for a specific NLP task.
- Output = word representations that can be plugged into just about any architecture a word embedding can be used.

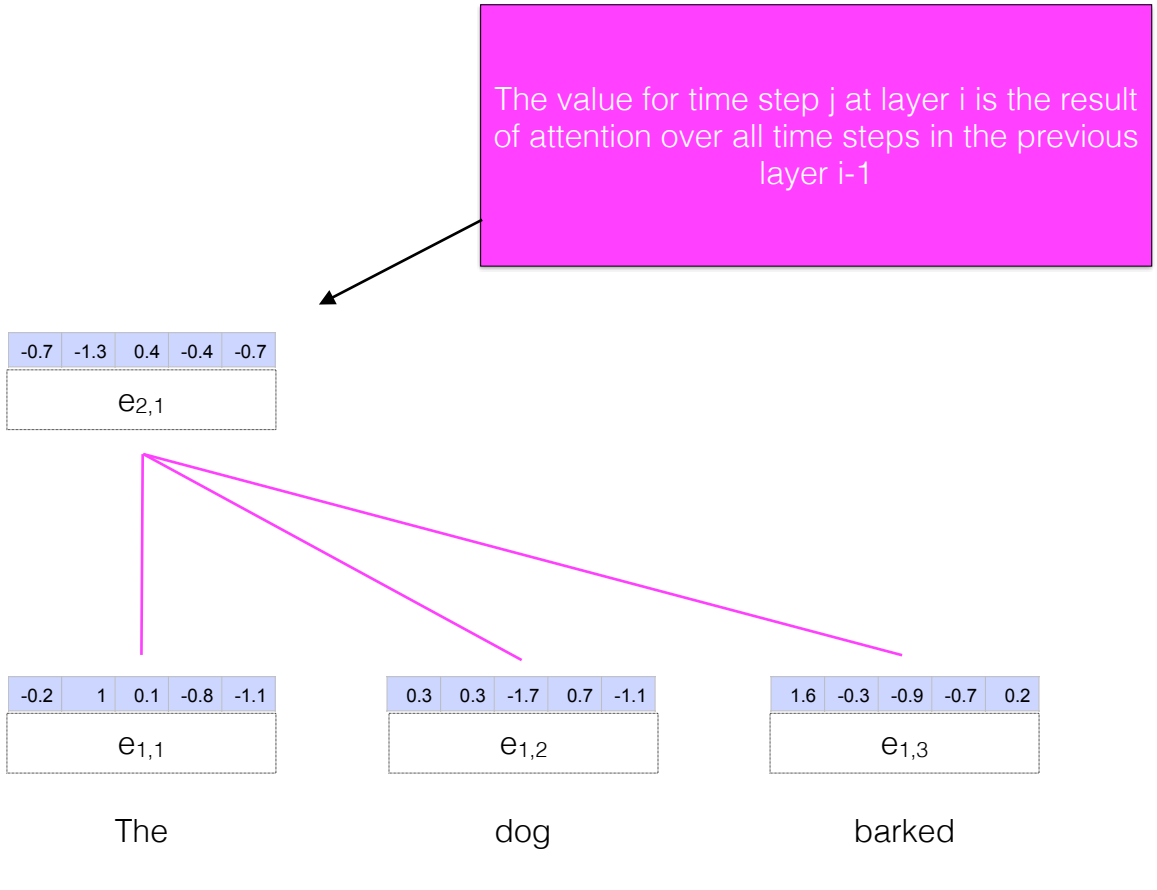
ELMo

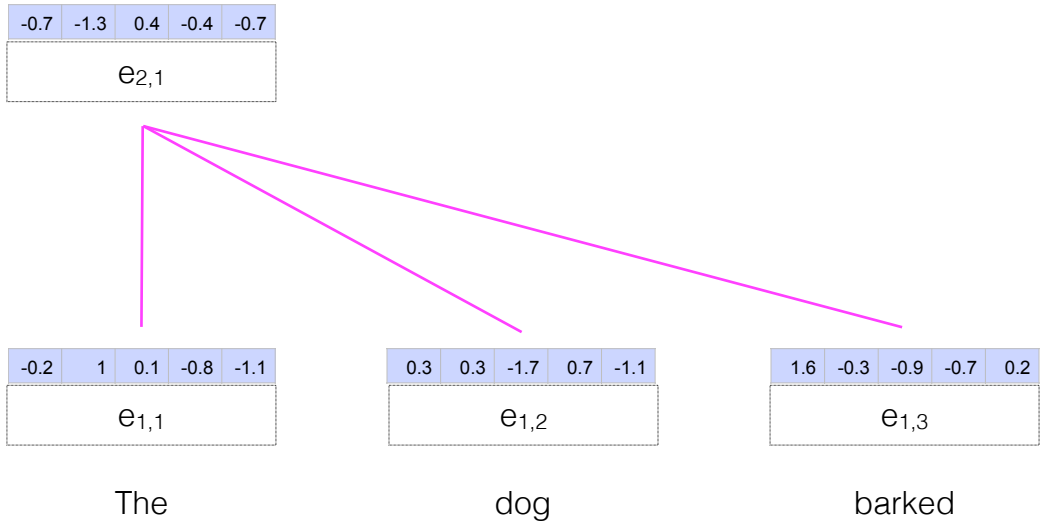
- Train a bidirectional RNN language model with L layers on a bunch of text.
- Learn parameters to combine the RNN output across all layers for each word in a sentence for a specific task (NER, semantic role labeling, question answering etc.). Large improvements over SOTA for lots of NLP problems.

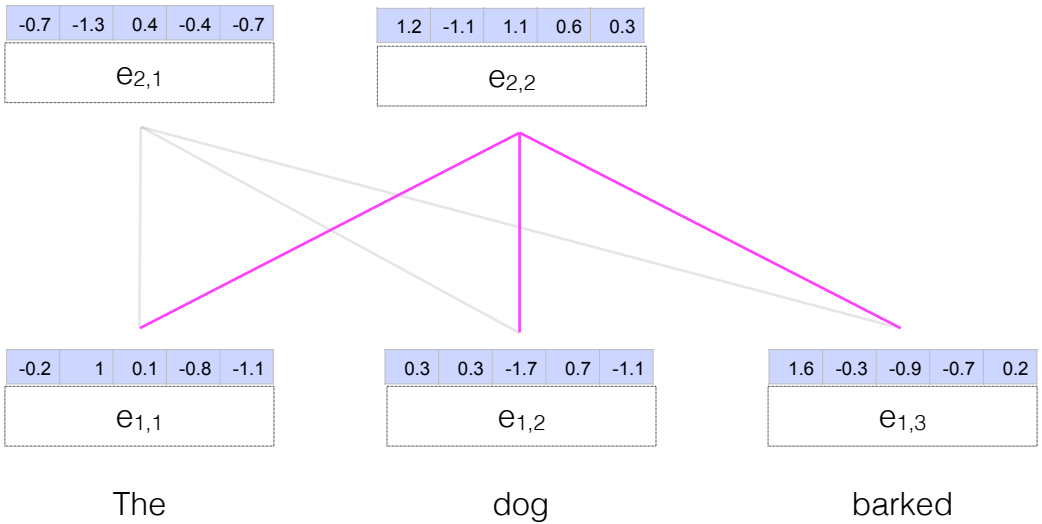
BERT

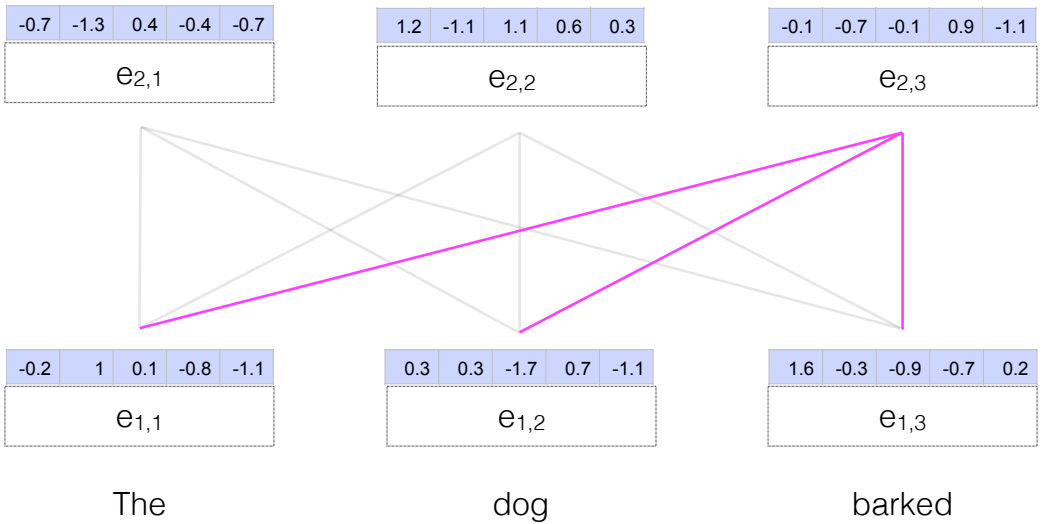
- Transformer-based model (Vaswani et al. 2017) to predict masked word using bidirectional context + next sentence prediction.
- Generates multiple layers of representations for each token sensitive to its context of use.

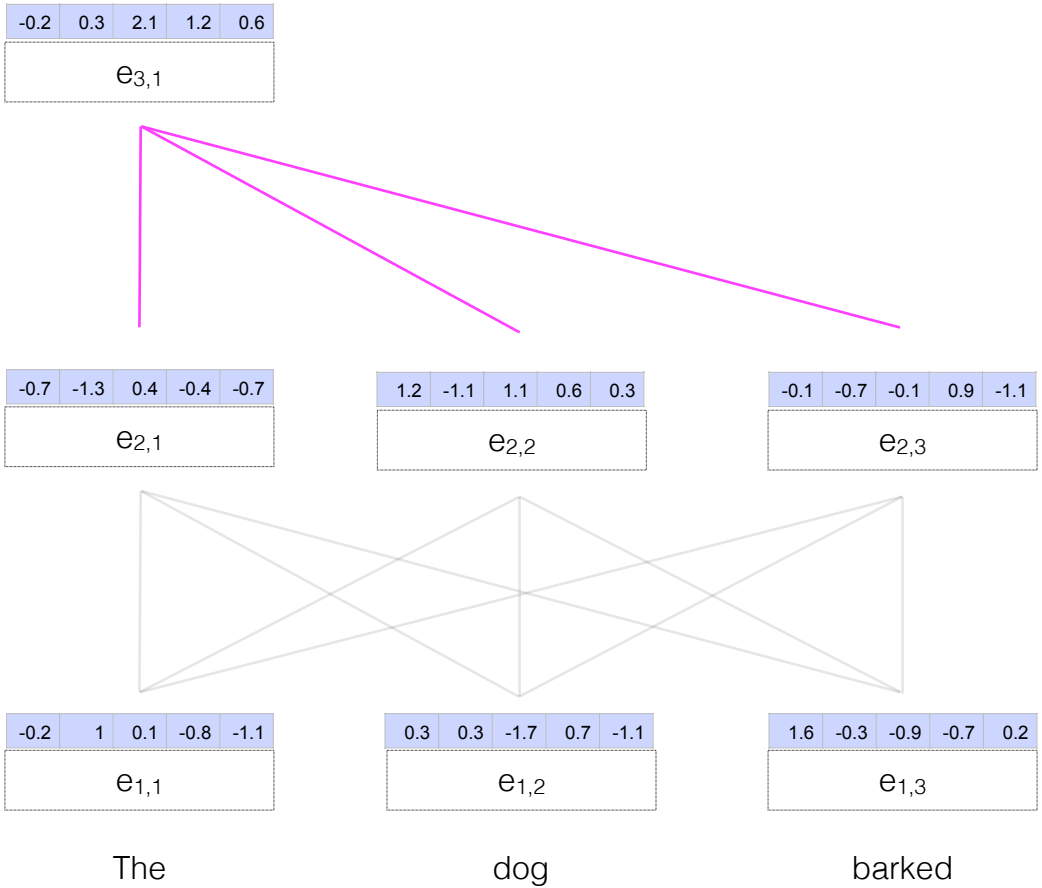

Each token in the input starts out represented by token and position embeddings
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked

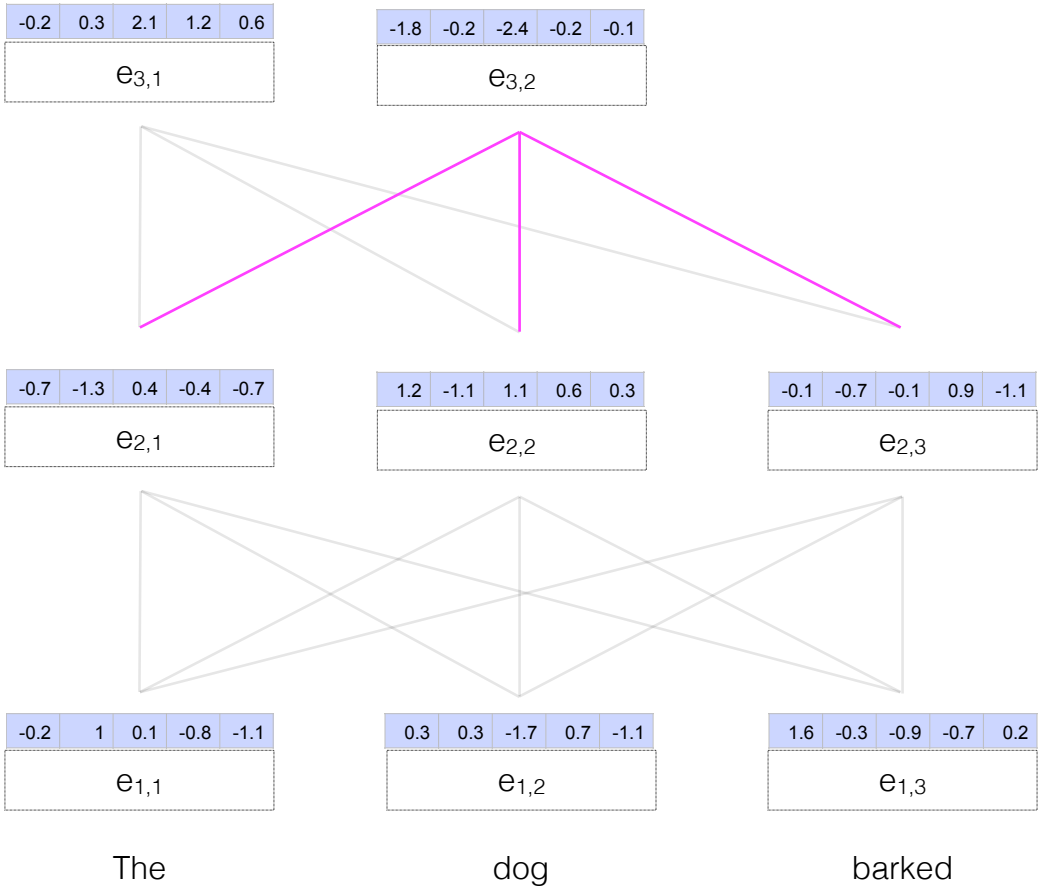

The value for time step j at layer i is the result of attention over all time steps in the previous layer i-1
-0.7 -1.3 0.4 -0.4 -0.7
e2,1
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked

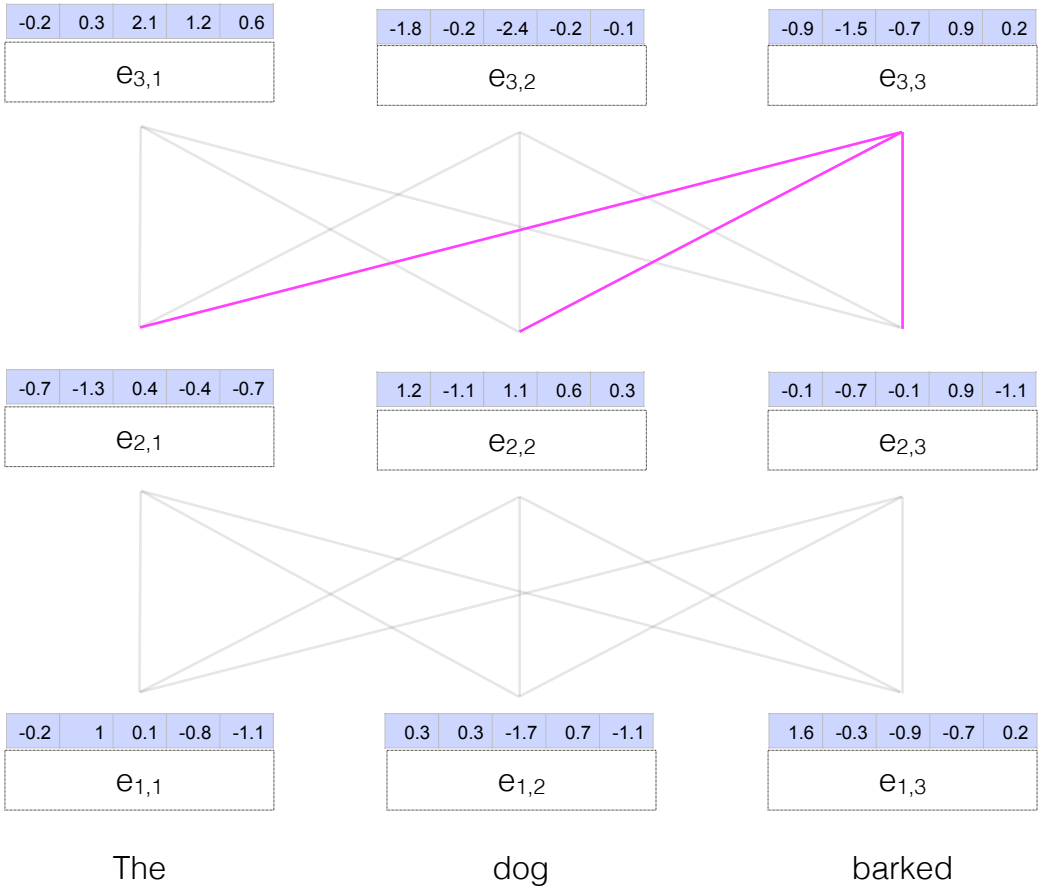

-0.7 -1.3 0.4 -0.4 -0.7
e2,1
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-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

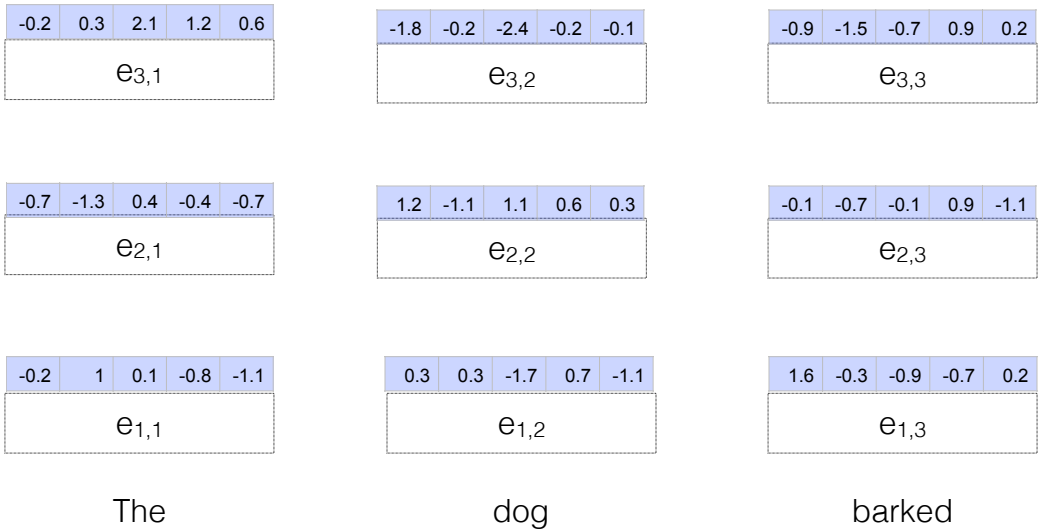

-0.7 -1.3 0.4 -0.4 -0.7
e2,1
1.2 -1.1 1.1 0.6 0.3
e2,2
-0.1 -0.7 -0.1 0.9 -1.1
e2,3
-0.2 1 0.1 -0.8 -1.1
e1,1
0.3 0.3 -1.7 0.7 -1.1
e1,2
1.6 -0.3 -0.9 -0.7 0.2
e1,3
The
dog
barked


-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
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

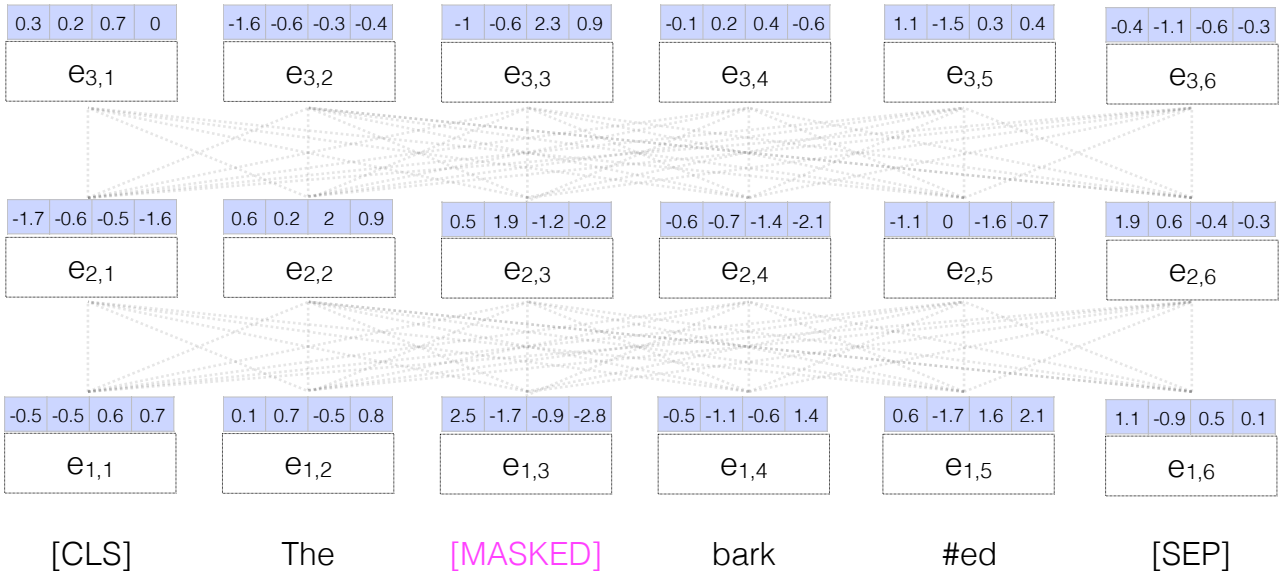


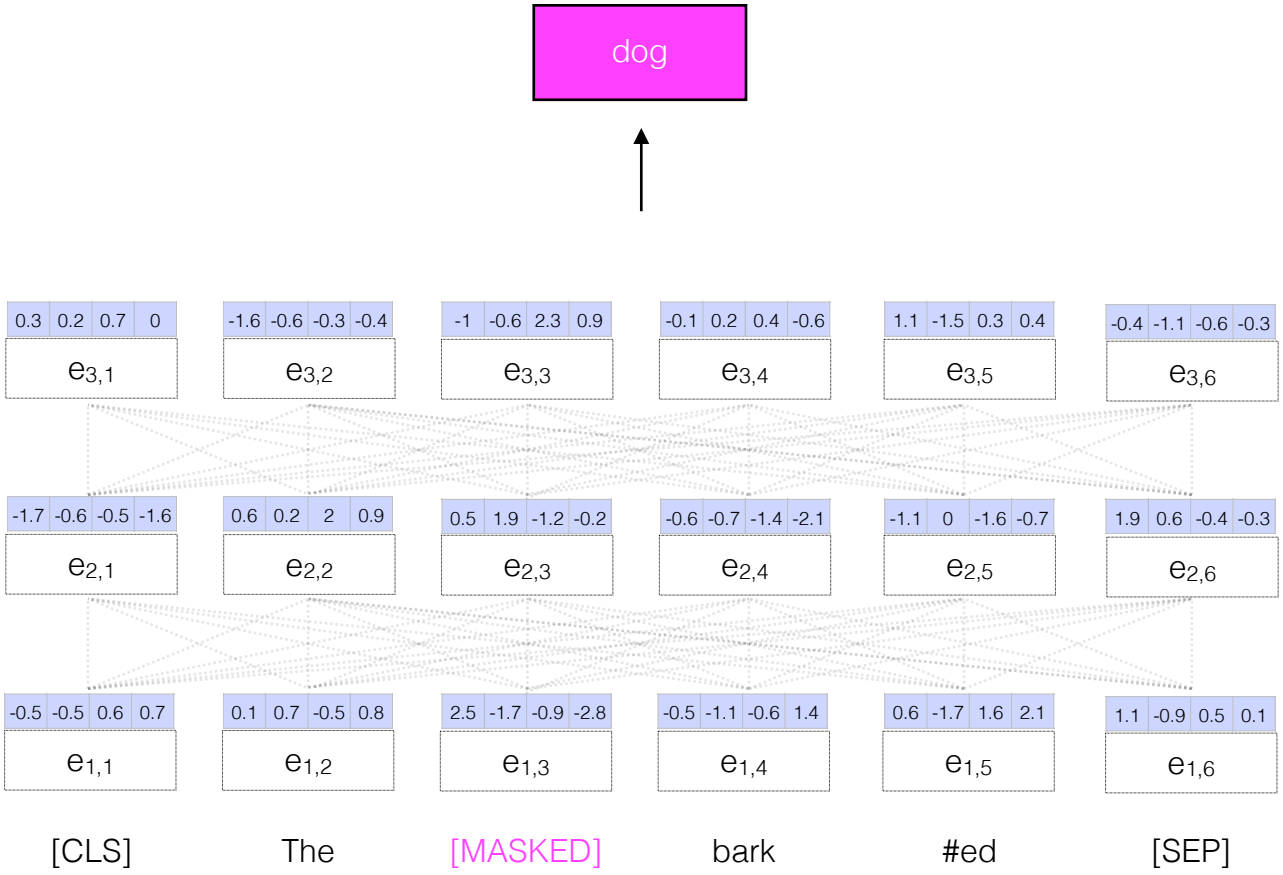
BERT

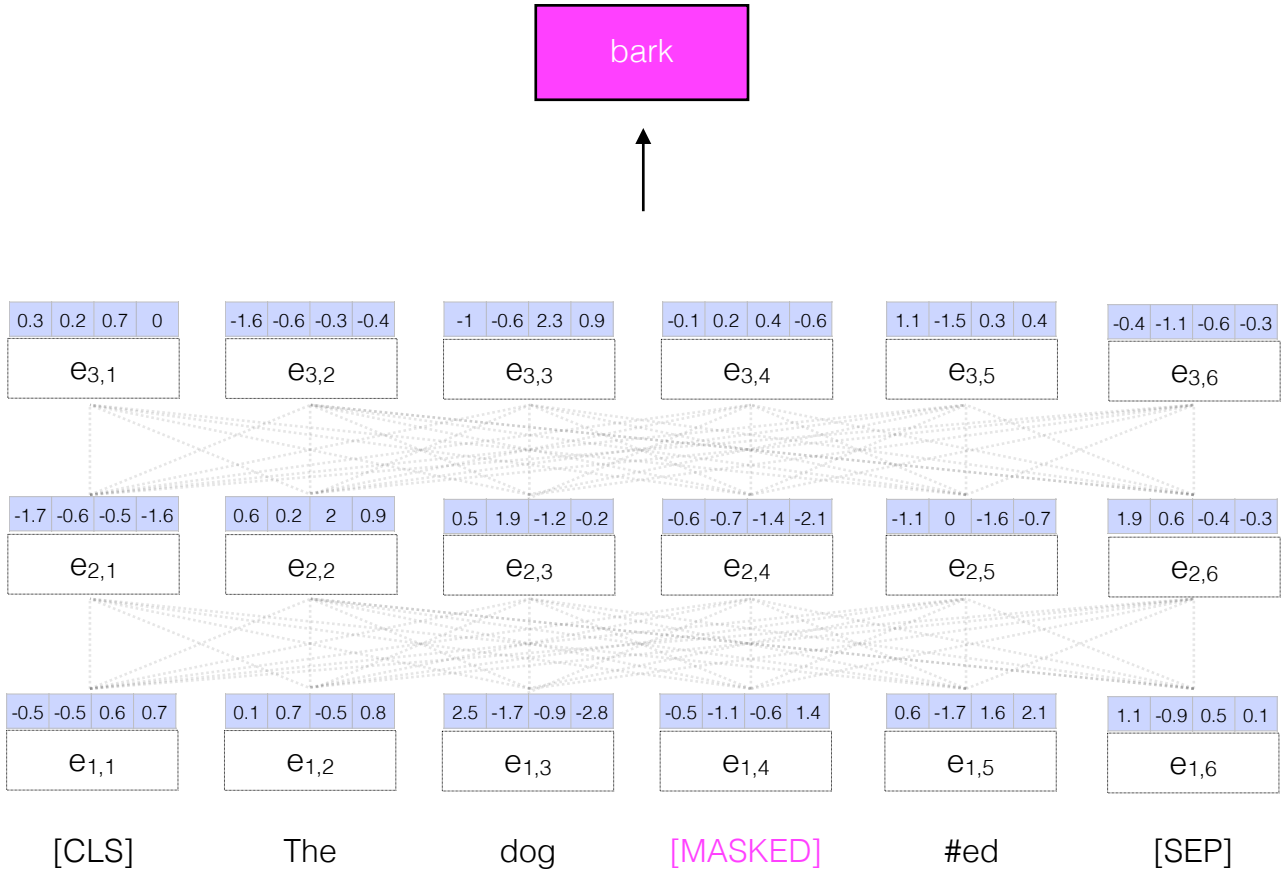
- Learn the parameters of this model with two objectives:
  - Masked language modeling
  - Next sentence prediction

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
-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
e3,1
e3,2
e3,3
e3,4
e3,5
e3,6
-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
e2,1
e2,2
e2,3
e2,4
e2,5
e2,6
-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
e1,1
e1,2
e1,3
e1,4
e1,5
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]

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)

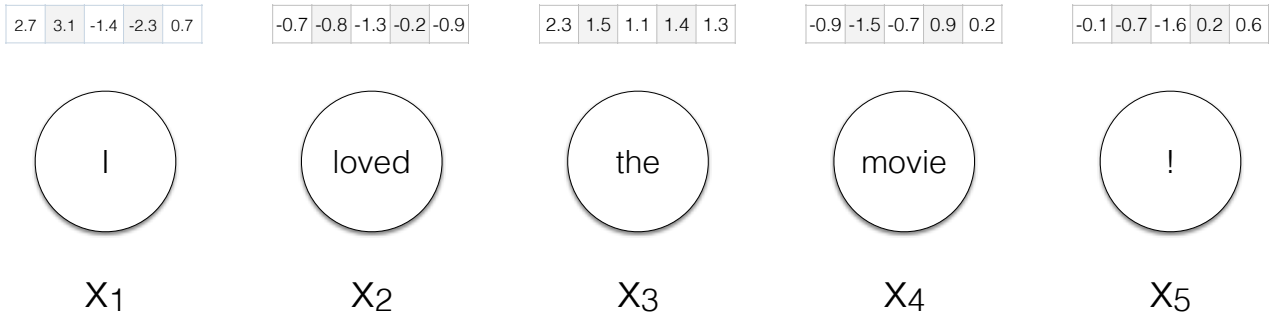
Attention

- 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



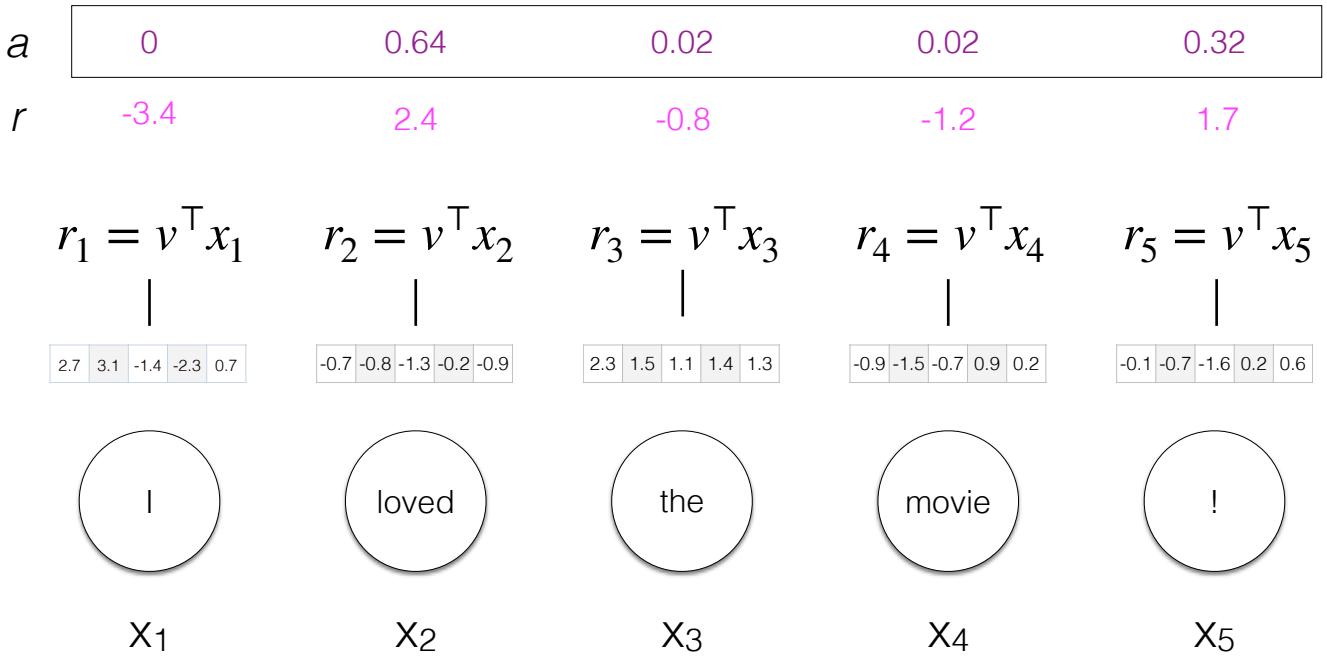
$$v \in \mathcal{R}^H$$

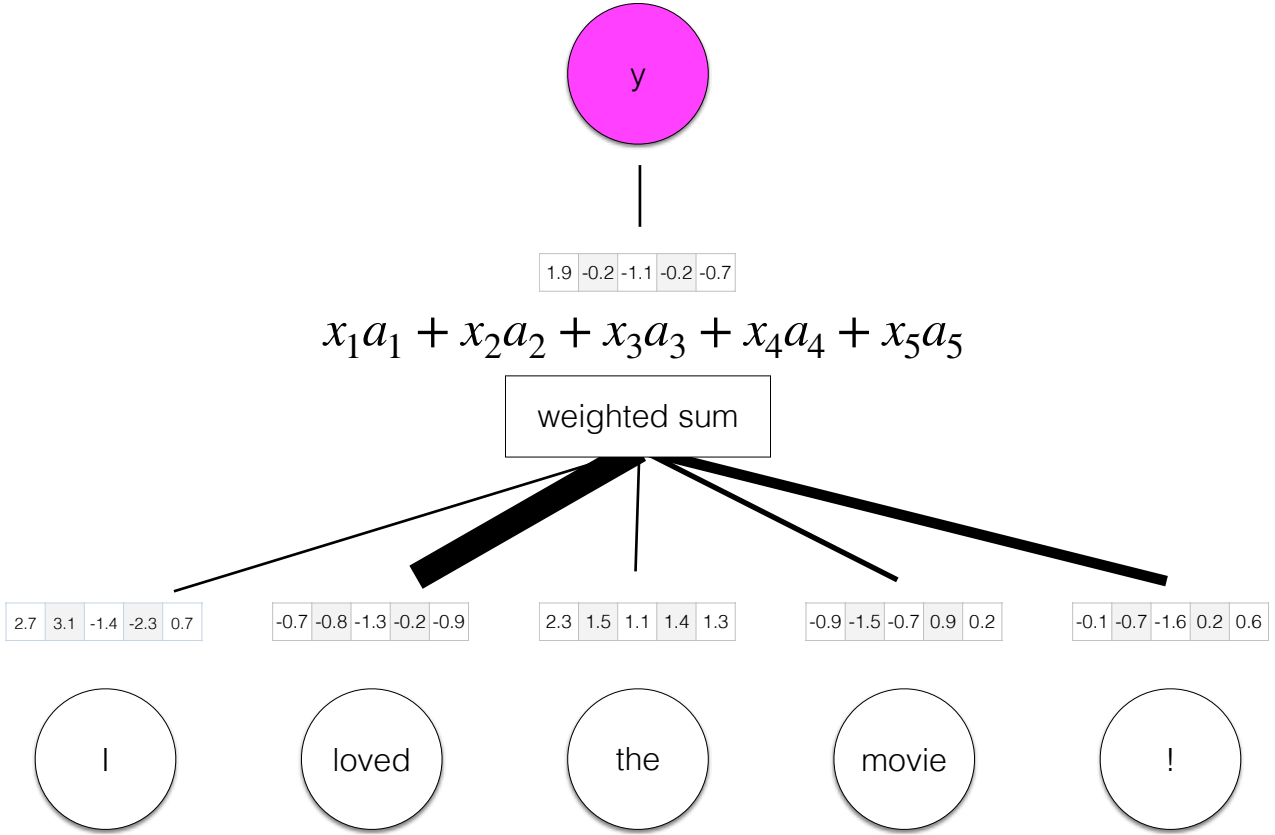
2.7	3.1	-1.4	-2.3	0.7

-3.4	2.4	-0.8	-1.2	1.7
$r_1 = v^\top x_1$	$r_2 = v^\top x_2$	$r_3 = v^\top x_3$	$r_4 = v^\top x_4$	$r_5 = v^\top x_5$
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	!
x_1	x_2	x_3	x_4	x_5

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

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




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

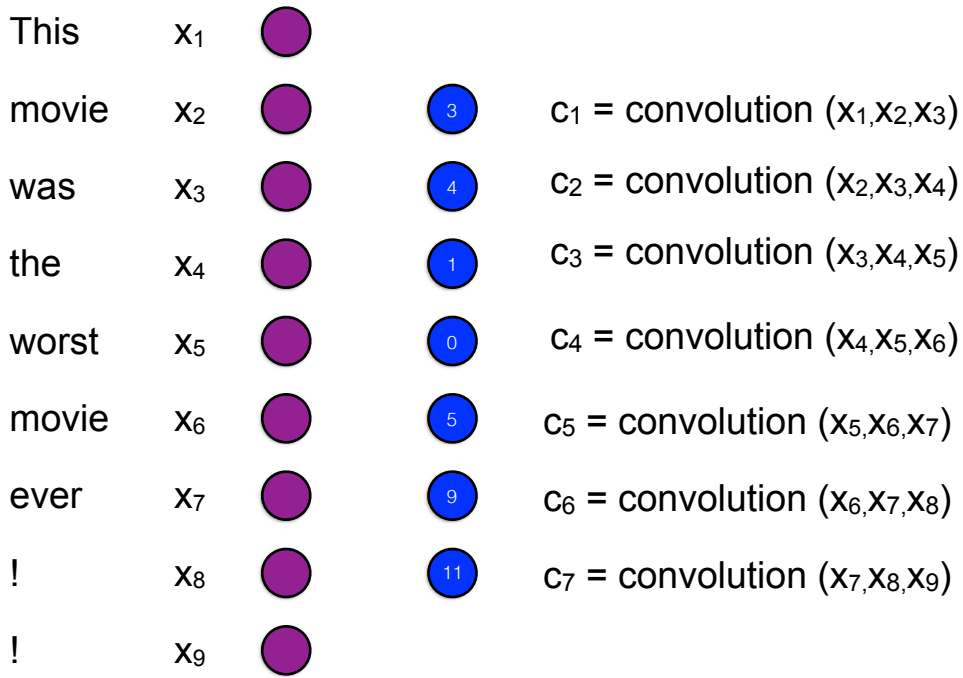
- Lots of variations on attention:
  - Linear transformation of x into before dotting with v
  - Non-linearities after each operation.
  - “Multi-head attention”: multiple v vectors to capture different phenomena that can be attended to in the input.
  - Hierarchical attention (sentence representation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

For CNNs, what is the purpose of max pooling and global pooling?

- Maxpooling down-samples a layer by selecting a single point from some set
- Max-pooling over time (global max pooling) selects the largest value over an entire sequence. This provides a mechanism to have fixed-dimensional representation for variable-sized inputs.
- Global max pooling is very common for NLP problems.

For CNNs, what is the purpose of max pooling and global pooling?

Here's the view for one filter, with a window size of 3. Let's assume we make a prediction (e.g., for sentiment) from this one filter alone

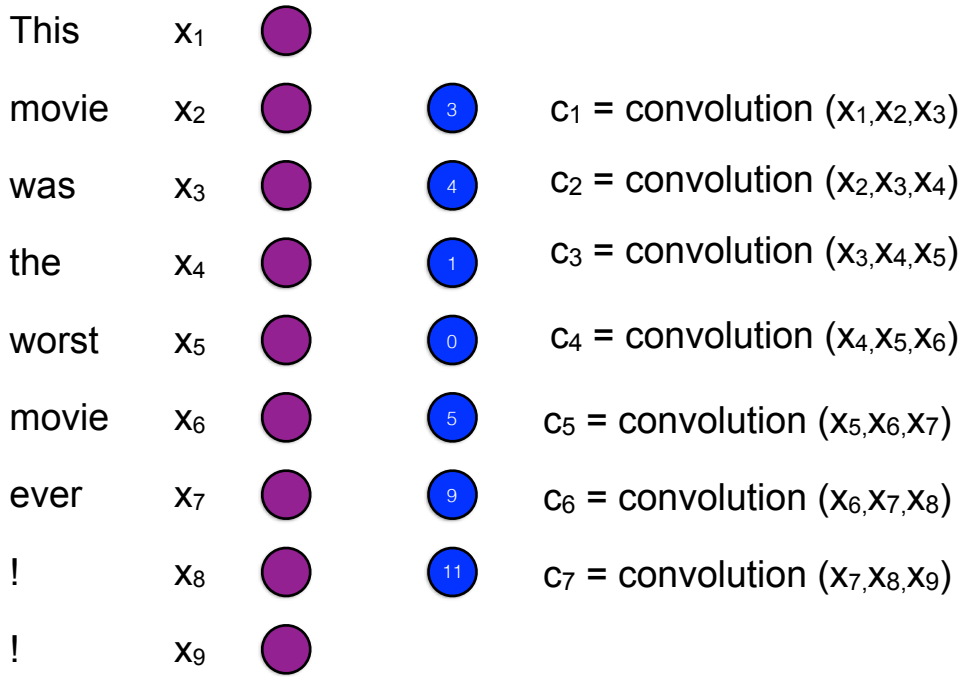


$$P(y \mid x) = \text{sigmoid}(g^\top W)$$

$$W \in \mathbb{R}^{1\times 1}$$

For CNNs, what is the purpose of max pooling and global pooling?

Let's say we **didn't** use global max pooling

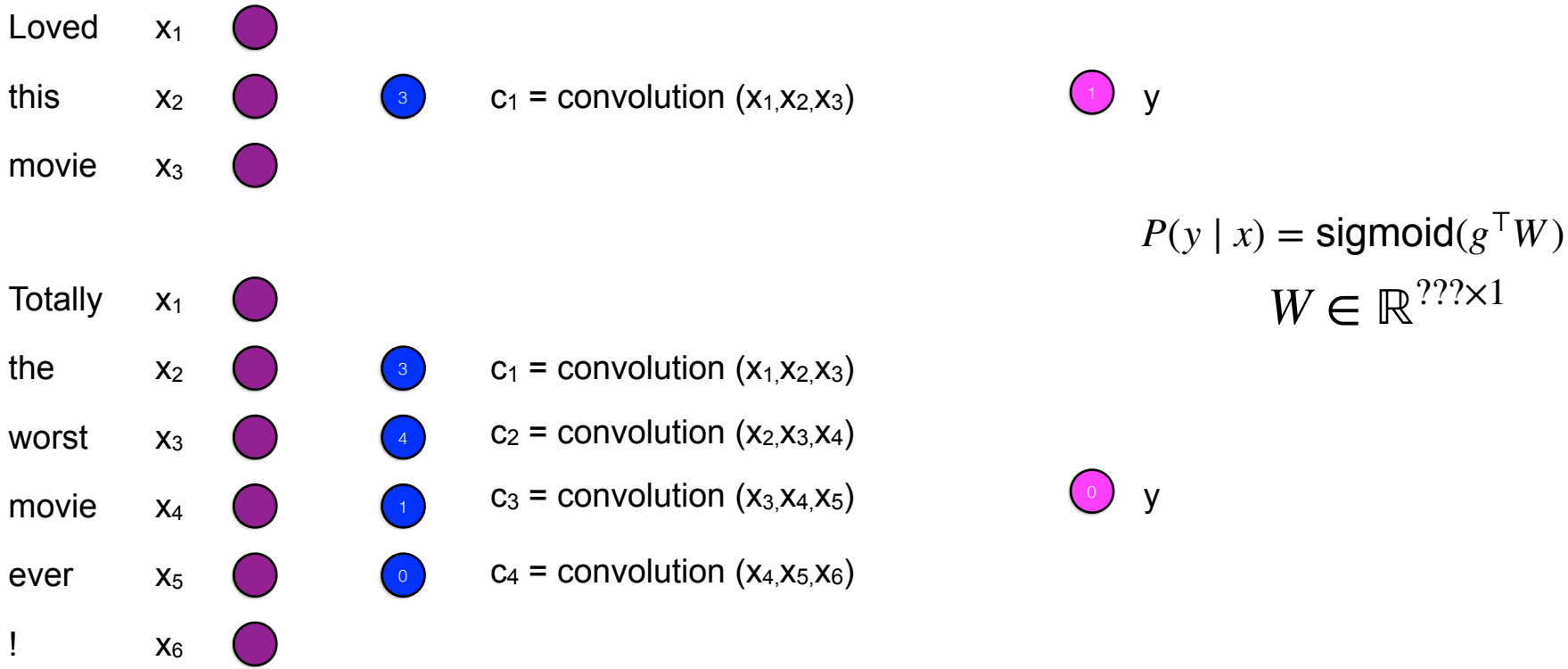


$$P(y \mid x) = \text{sigmoid}(g^\top W)$$

$$W \in \mathbb{R}^{7\times 1}$$

For CNNs, what is the purpose of max pooling and global pooling?

Let's say we **didn't** use global max pooling



Could you go over these concepts in terms of their similarities and differences (skip gram, Masked LM, Gapping n-gram)? I occasionally get those confused.

- Skip-gram: word2vec model for learning representations of word types.
- Masked language model: “masking” out a word in an input sequence and predicting it in order to learn word representations (BERT).
- Gappy ngrams (aka “skip ngrams”). A feature based on set of words that may have gaps between them.

It was a dark and stormy night

- It was
- was a
- a dark
- dark and
- and stormy
- stormy night

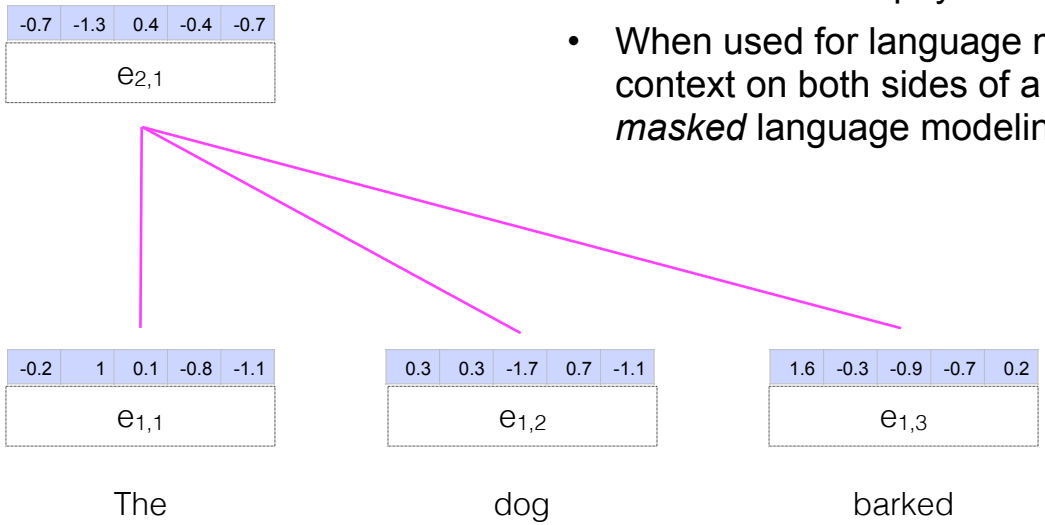
← Bigram features

Skip bigram features →

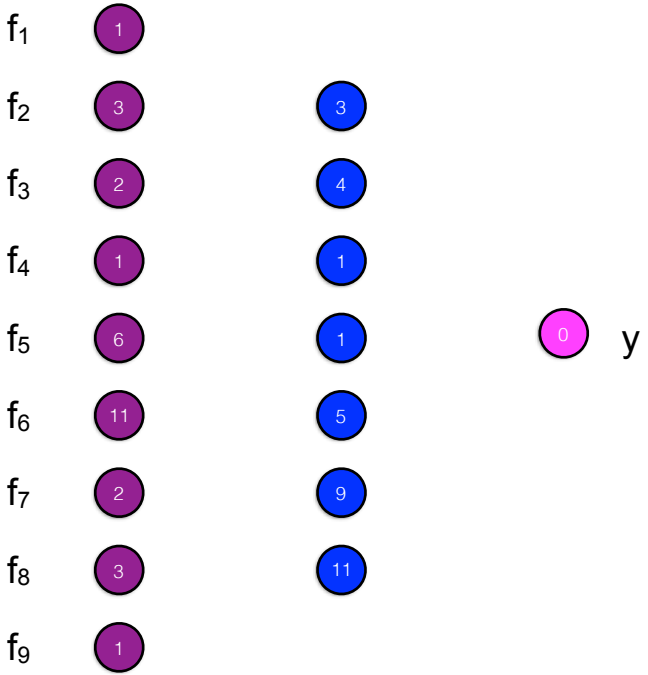
- It was
- was a
- a dark
- dark and
- and stormy
- stormy night
- It a
- It dark
- It and
- It stormy
- It night
- was dark
- was and
- was stormy
- was night
- dark stormy
- dark night
- and night

Could you go over, generally, what transformers are and what makes them so powerful?

- Transformers provide simultaneous access to all other tokens in a sequence when generating a representation (unlike RNNs)
- Attention as a mechanism allows for learning what is important for each word to pay attention to its context
- When used for language modeling, they provide access to context on both sides of a word being predicted (allowing for *masked* language modeling as an objective).

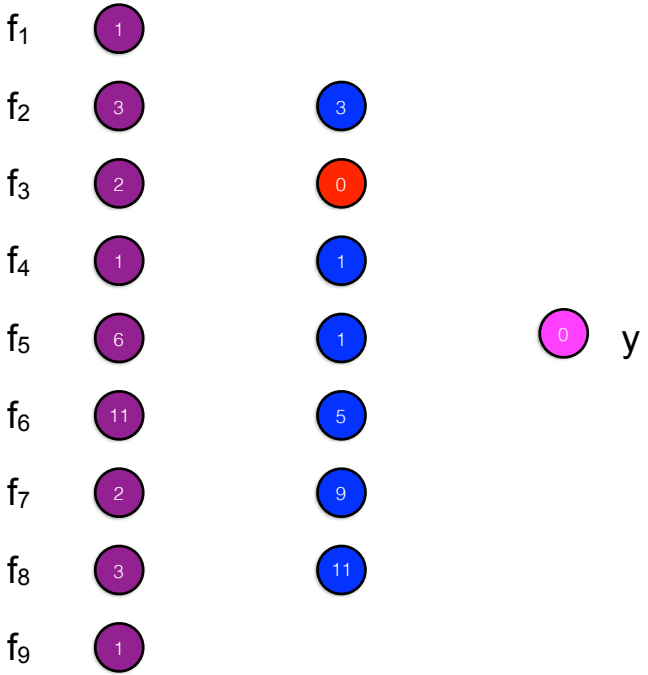


Could you go over dropout in regularization?



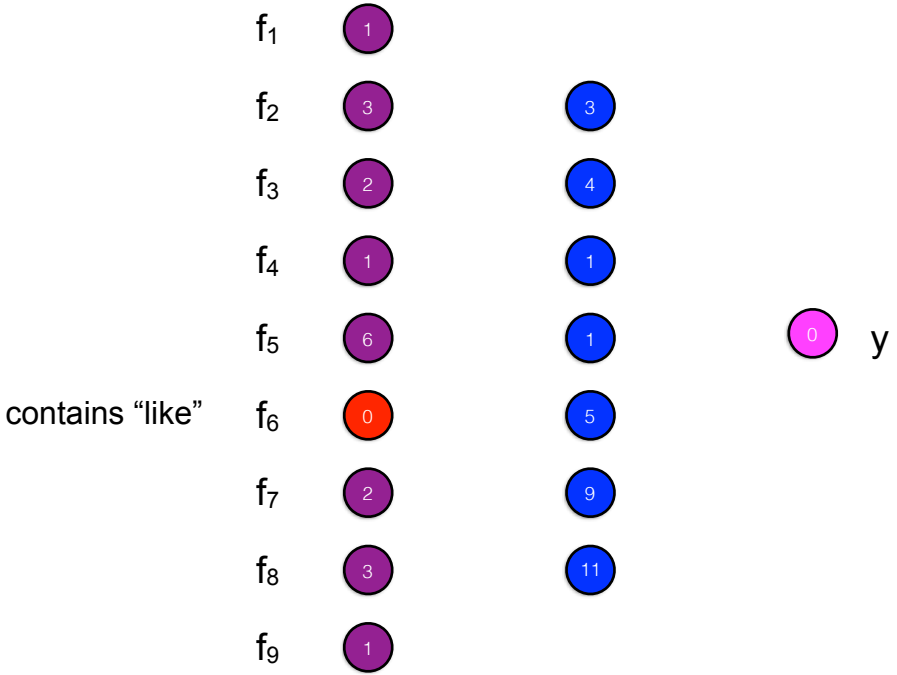
- Let's assume a FFNN (like a MLP) with a single hidden layer.
- Dropout removes nodes during training to encourage the model to not rely on them.
- Only active during train time, not at test time.

Could you go over dropout in regularization?



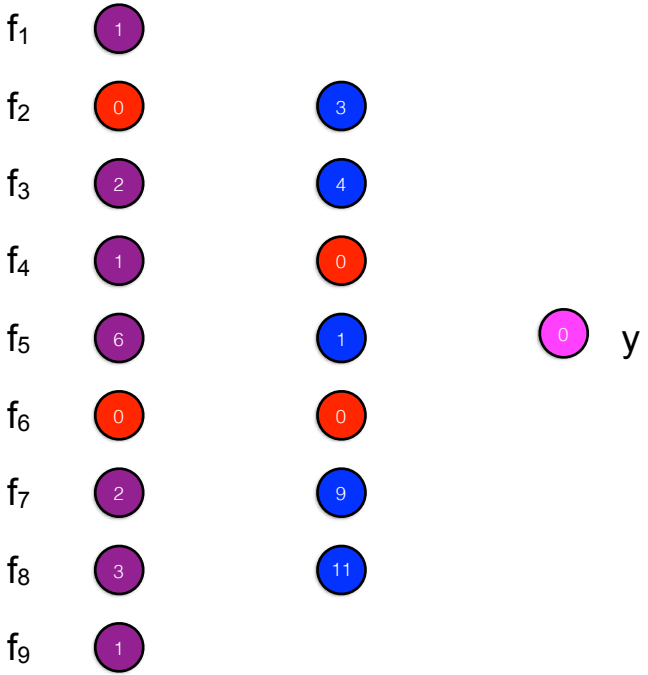
- Let's assume a FFNN (like a MLP) with a single hidden layer.
- Dropout removes nodes during training to encourage the model to not rely on them.
- Only active during train time, not at test time.

Could you go over dropout in regularization?



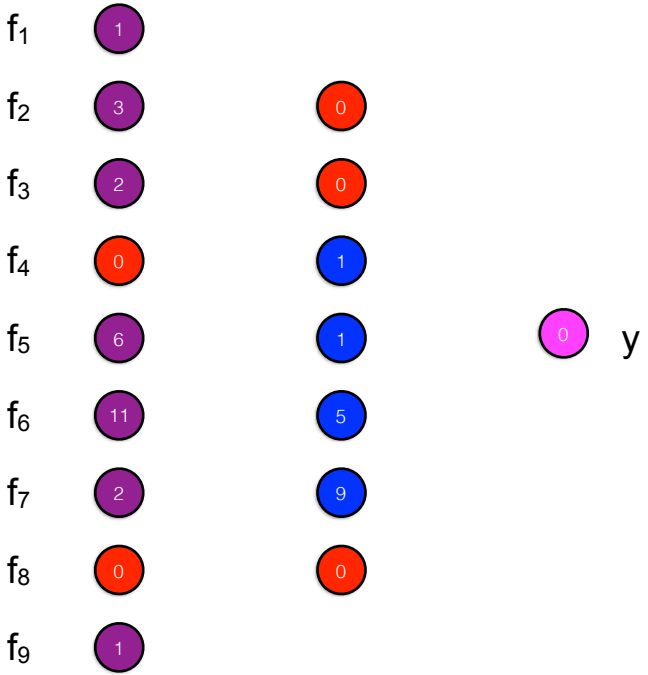
- Let’s assume a FFNN (like a MLP) with a single hidden layer.
- Dropout removes nodes during training to encourage the model to not rely on them.
- Only active during train time, not at test time.

Could you go over dropout in regularization?



- Let's assume a FFNN (like a MLP) with a single hidden layer.
- Dropout removes nodes during training to encourage the model to not rely on them.
- Only active during train time, not at test time.

Could you go over dropout in regularization?



- Let's assume a FFNN (like a MLP) with a single hidden layer.
- Dropout removes nodes during training to encourage the model to not rely on them.
- Only active during train time, not at test time.

What's the difference between HMM, MEMM, and CRF? And when do we know to use which one?

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

Can you go over the differences in the intuitions and implementations of Laplace and Kneser-Ney smoothing?

Laplace smoothing:
α = 1

$$P(w_i) = \frac{c(w_i) + \alpha}{N + V\alpha}$$

$$P(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) + \alpha}{c(w_{i-1}) + V\alpha}$$

Kneser-Ney smoothing

- Intuition: When backing off to a lower-order ngram, maybe the overall ngram frequency is not our best guess.

I can't see without my reading _____

P("Francisco") > P("glasses")

- *Francisco* is more frequent, but shows up in fewer unique bigrams ("San Francisco") — so we shouldn't expect it in new contexts; *glasses*, however, does show up in many different bigrams

Kneser-Ney smoothing

discounted mass

$$\frac{\max\{c(w_{i-1}, w_i) - d, 0\}}{c(w_{i-1})} + \lambda(w_{i-1})\, P_{CONTINUATION}(w_i)$$

discounted bigram probability

continuition probability

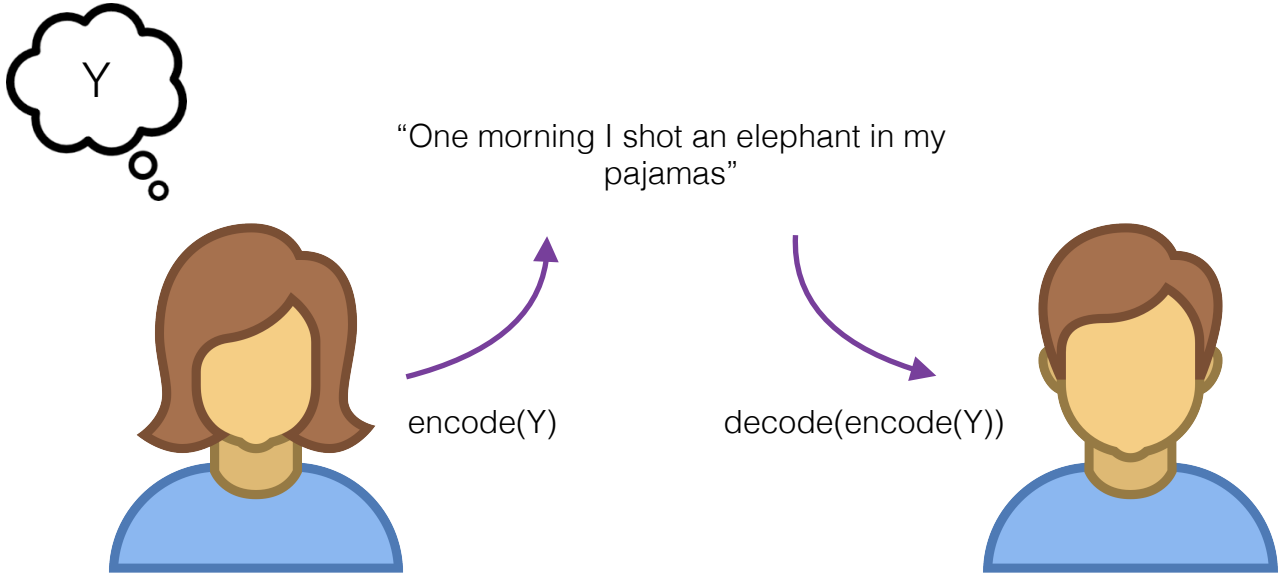
Kneser-Ney smoothing

w_{i-1}	w_i	$C(w_{i-1}, w_i)$	$C(w_{i-1}, w_i) - d(1)$
red	hook	3	2
red	car	2	1
red	watch	10	9
sum		15	12

$$\lambda(\text{red}) = 1 \times \frac{3}{15}$$

12/15 of the probability mass stays with the original counts; 3/15 is reallocated

Can you go over the information theoretic view and how it related to language modeling?



Shannon 1948

Noisy Channel

	X	Y
ASR	speech signal	transcription
MT	target text	source text
OCR	pixel densities	transcription

$$P(Y \mid X) \propto \underbrace{P(X \mid Y)}_{\text{channel model}} \quad \underbrace{P(Y)}_{\text{source model}}$$