CS378: Natural Language Processing

Lecture 17: Transformer / Contextual Word Embeddings

Eunsol Choi

Slides from Greg Durrett, Yoav Artzi, Yejin Choi, Princeton NLP

Logistics

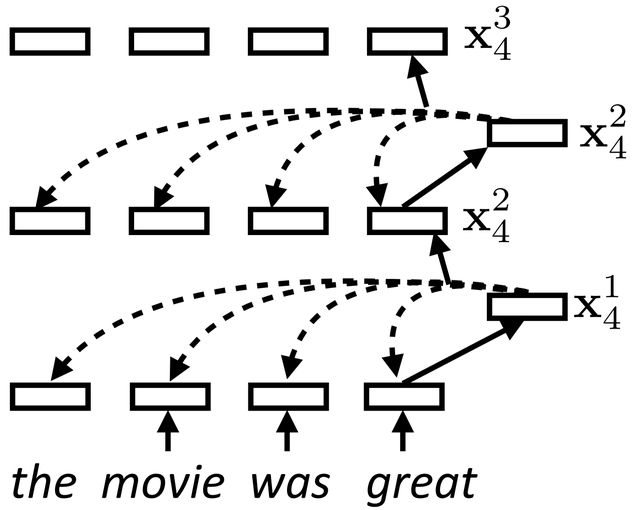
- Extra Credit on Canvas about Guest Lecture— expires the end of this week
- HW3 grades released
- Please fill out Mid-semester Survey on Canvas
  - Under the “Quiz” tab

This Week

- Wrap up transformer
- Contextualized word embeddings
  - ELMo — Embeddings from Word Embeddings
  - BERT — Bidirectional Transformers for Language Understanding
- Next Lecture:
  - Encoder-Decoder models with pre-training

Recap: Self Attention

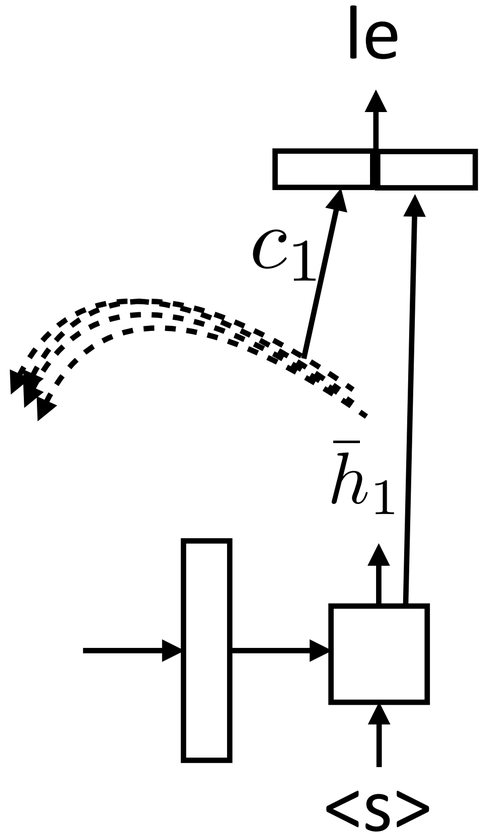
- Using attention for the ***encoder***.
- Each input token is a query to form attention over all tokens.
- Then, attention weights dynamically mix how much is taken from all tokens.



- Context-dependent representation of each token: a weighted sum of all tokens
- This will happen iteratively! Each step computing self-attention on the output of the previous level

[Vaswani et al. 2017]

Recap: Attention Score Function



$$e_{ij} = f(\bar{h}_i, h_j)$$

$$h_i, h_j \in R^{d_k}$$

- Bahdanau+ (2014): additive

$$f(\bar{h}_i, h_j) = \tanh(W[\bar{h}_i, h_j])$$

- Luong+ (2015): dot product

$$f(\bar{h}_i, h_j) = \bar{h}_i \cdot h_j$$

$$f(\hat{h}_i, h_j) = \frac{\bar{h}_i \cdot h_j}{\sqrt{d_k}}$$

- Luong+ (2015): bilinear

$$f(\bar{h}_i, h_j) = \bar{h}_i^\top W h_j$$

Luong et al. (2015)

Multiple Attention Heads

- Why multiple heads? Softmax operations often end up peaky, making it hard to put weight on multiple items
- You can think of each head capturing different dependencies: one for finding subject, one for finding object, etc…

Multiple Attention Heads

- Single-headed attention

$$k : \text{level number}$$
$$X : \text{input vectors}$$
$$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$$
$$\mathbf{x}_i^1 = \mathbf{x}_i$$
$$\bar{\alpha}_{i,j}^k = \mathbf{x}_i^{k-1} \cdot \mathbf{x}_j^{k-1}$$
$$\alpha_{i,j}^k = \frac{\exp(\bar{\alpha}_{i,j}^k)}{\sum_j \exp(\bar{\alpha}_{i,j}^k)}$$
$$x_i^k = \sum_j \alpha_{i,j}^k x_j^{k-1}$$

- Multi-headed attention

$$k : \text{level number}$$
$$L : \text{number of heads}$$
$$X : \text{input vectors}$$
$$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$$
$$\mathbf{x}_i^1 = \mathbf{x}_i$$
$$\bar{\alpha}_{i,j}^{k,l} = \mathbf{x}_i^{k-1} \mathbf{W}^{k,l} \mathbf{x}_j^{k-1}$$
$$\alpha_{i,j}^{k,l} = \frac{\exp(\bar{\alpha}_{i,j}^{k,l})}{\sum_j \exp(\bar{\alpha}_{i,j}^{k,l})}$$
$$x_i^{k,l} = \sum_j \alpha_{i,j}^{k,l} x_j^{k-1}$$
$$\mathbf{x}_i^k = \mathbf{V}^k [\mathbf{x}_i^{k,1}; \ldots; \mathbf{x}_i^{k,L}]$$

Multiple Attention Heads

- Single-headed attention

k : level number

X : input vectors

$X = \mathbf{x}_1, \ldots, \mathbf{x}_n$

$\mathbf{x}_i^1 = \mathbf{x}_i$

$\bar{\alpha}_{i,j}^k$

$\alpha_{i,j}^k$

x_i^k

j

- Multi-headed attention

k : level number

L : number of heads

X : input vectors

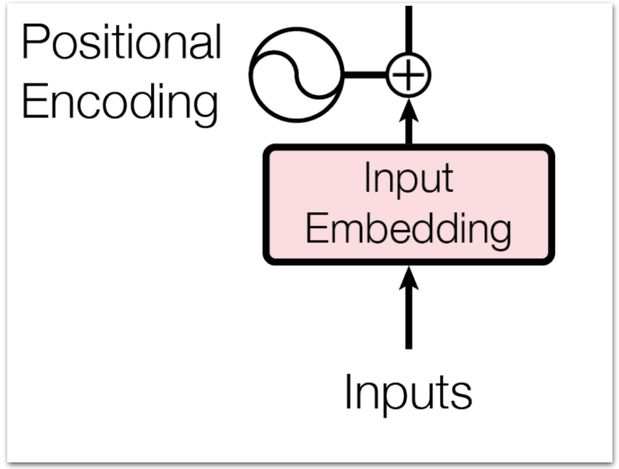
$X = \mathbf{x}_1 \quad \mathbf{x}$

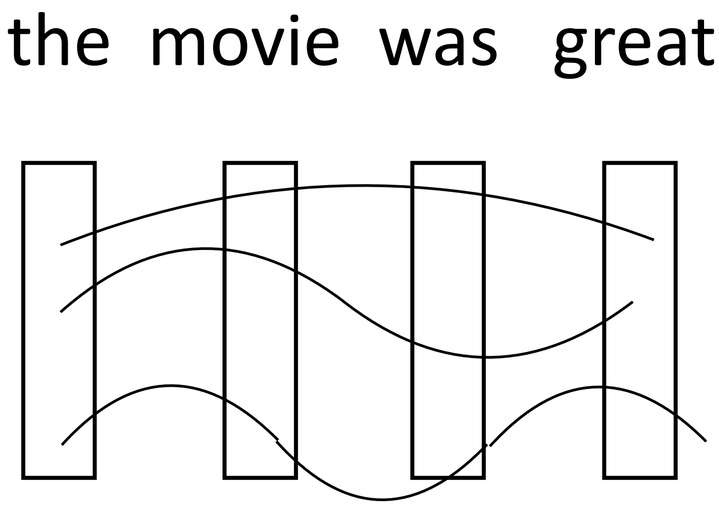
$\alpha_{i,j}^{k,l} x_j^{k-1}$

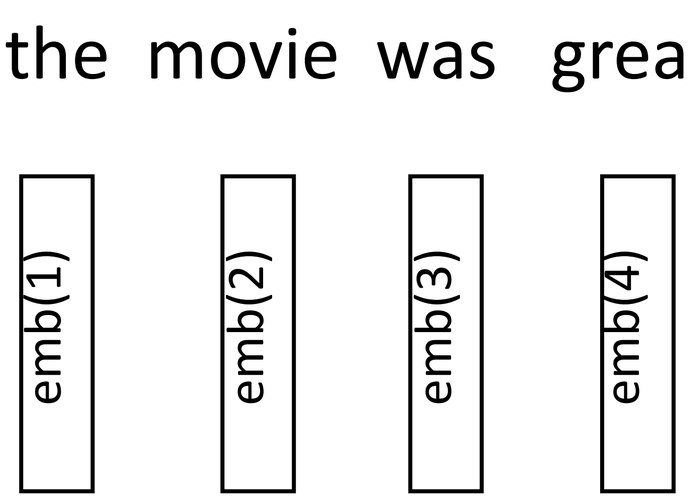
$; \ldots ; \mathbf{x}_i^{k,L}]$

Would this algorithm know the position of the words in the input sequence?

Positional Embeddings



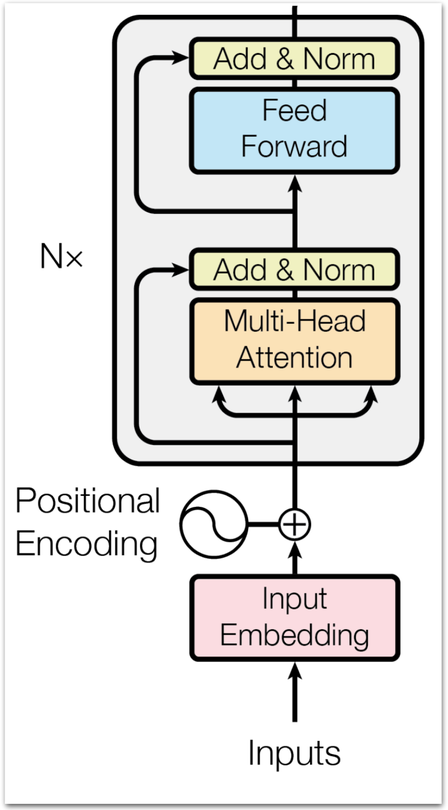




- Augment word embedding with position embeddings, each dim is a sine/cosine wave of a different frequency. Closer points = higher dot products
- Works essentially as well as just encoding position as a one-hot vector

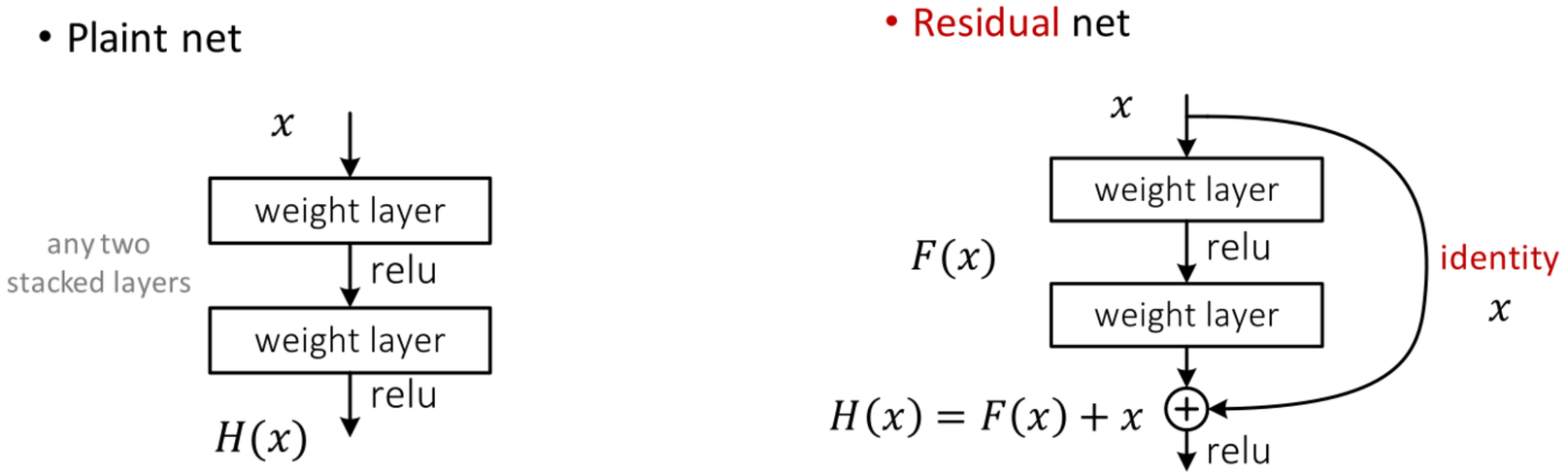
Vaswani et al. (2017)

Transformer Block



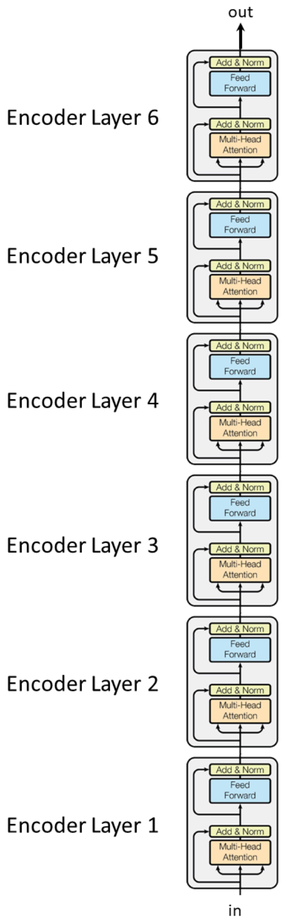
- Input is the sum of token embedding and positional encoding
- Multi-headed attention followed by feed forward layers
- Details, Details, Details!
  - Linear layer for key, value, query before self-attention
  - Residual layer
  - Layer normalization
  - Regularization (dropout)

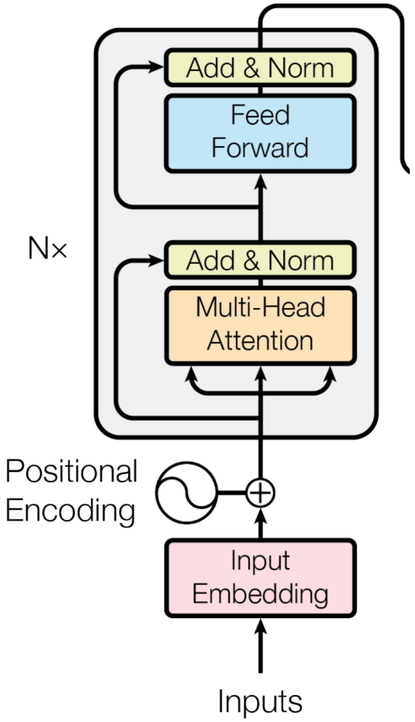
Residual Network



- ResNet (He et al. 2015): first very deep (152 layers) network successfully trained for object recognition

Transformers

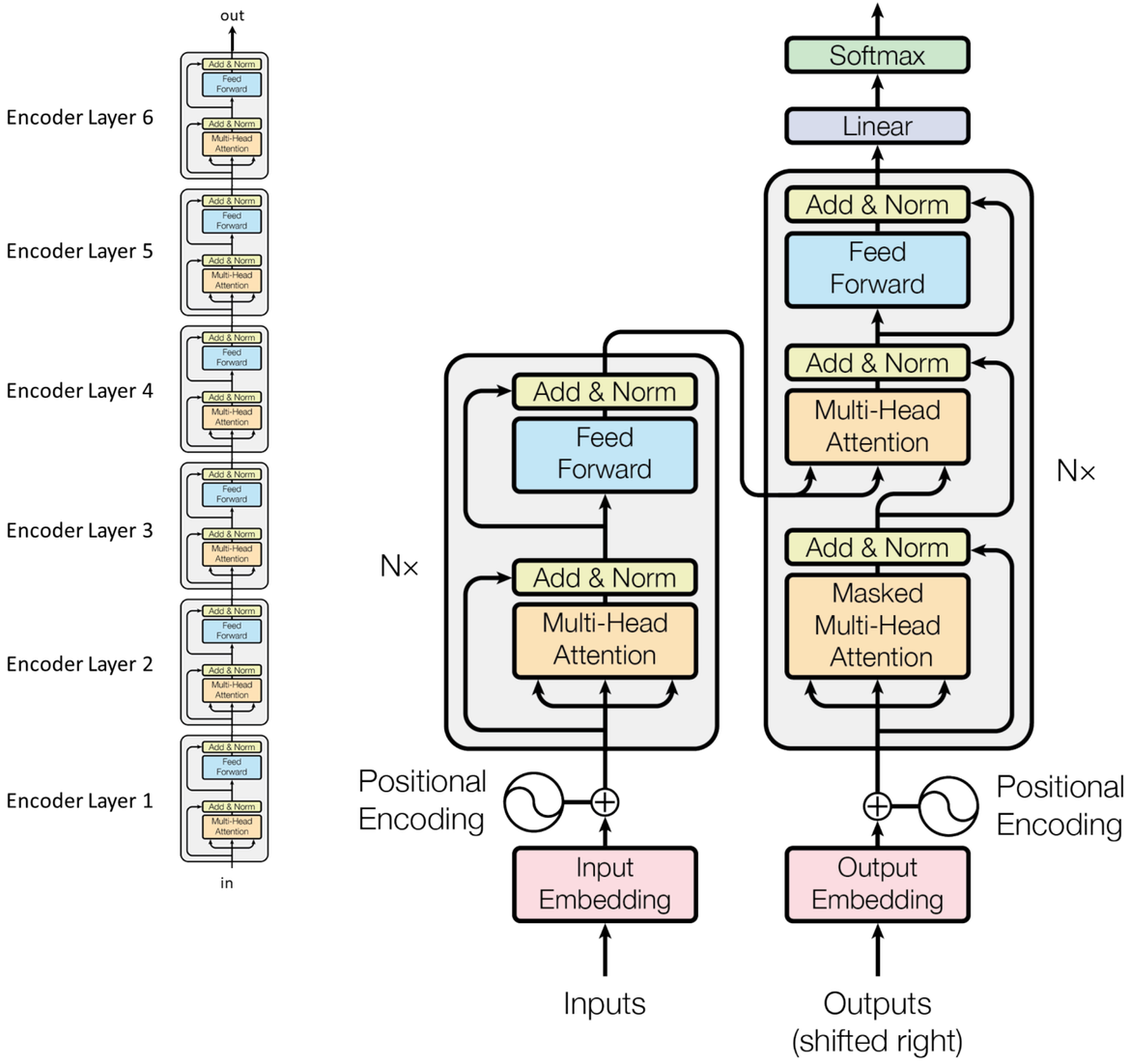




- Encoder and decoder are both transformers blocks

Vaswani et al. (2017)

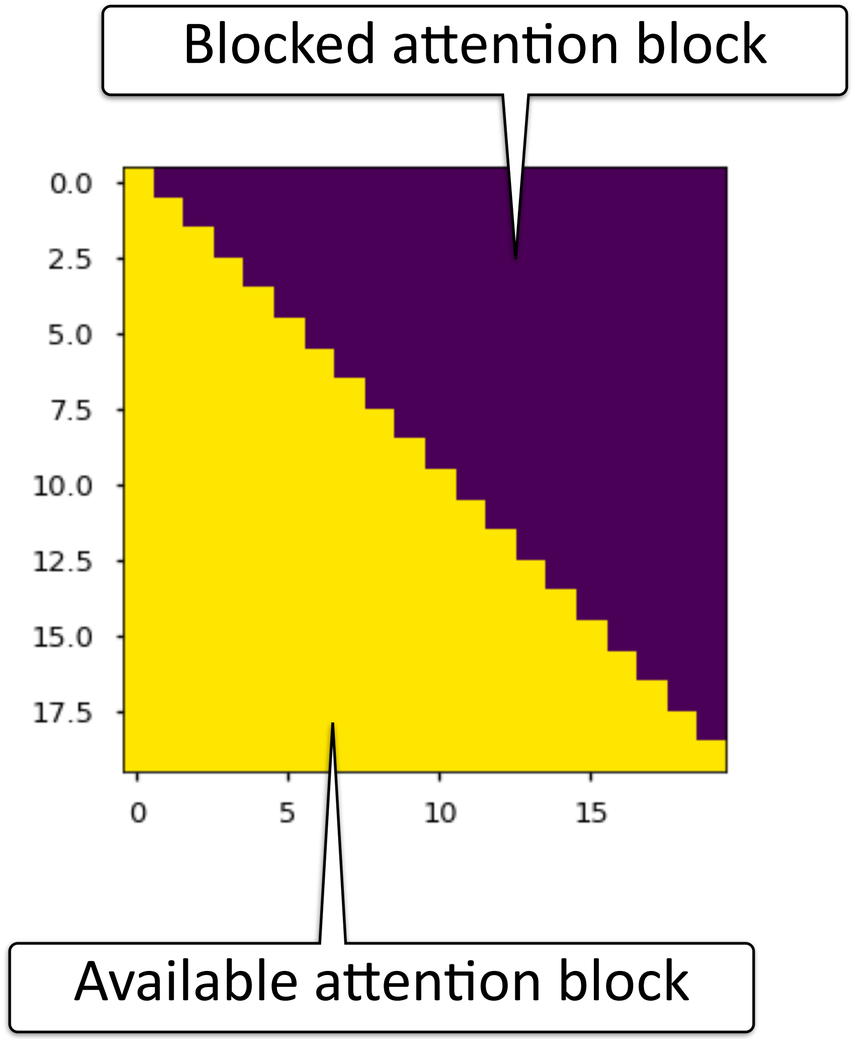
Transformers



- Encoder and decoder are both transformers blocks
- Decoder has self-attention, followed by attention over the encoder outputs (similar to vanilla encoder-decoder)

Vaswani et al. (2017)

Decoding with Transformers



- Decoder consumes the previous generated token (and attends to input), without *recurrent state*
- Words are blocked for attending to future words

Performances of Transformers: MT

- big = 6 layers, 1000 dim for each token, 16 heads
- base = 6 layers + other params halved

Model	BLEU	
	EN-DE	EN-FR
ByteNet [18]	23.75	
Deep-Att + PosUnk [39]		39.2
GNMT + RL [38]	24.6	39.92
ConvS2S [9]	25.16	40.46
MoE [32]	26.03	40.56
Deep-Att + PosUnk Ensemble [39]		40.4
GNMT + RL Ensemble [38]	26.30	41.16
ConvS2S Ensemble [9]	26.36	**41.29**
Transformer (base model)	27.3	38.1
Transformer (big)	**28.4**	**41.8**

Vaswani et al. (2017)

Visualization

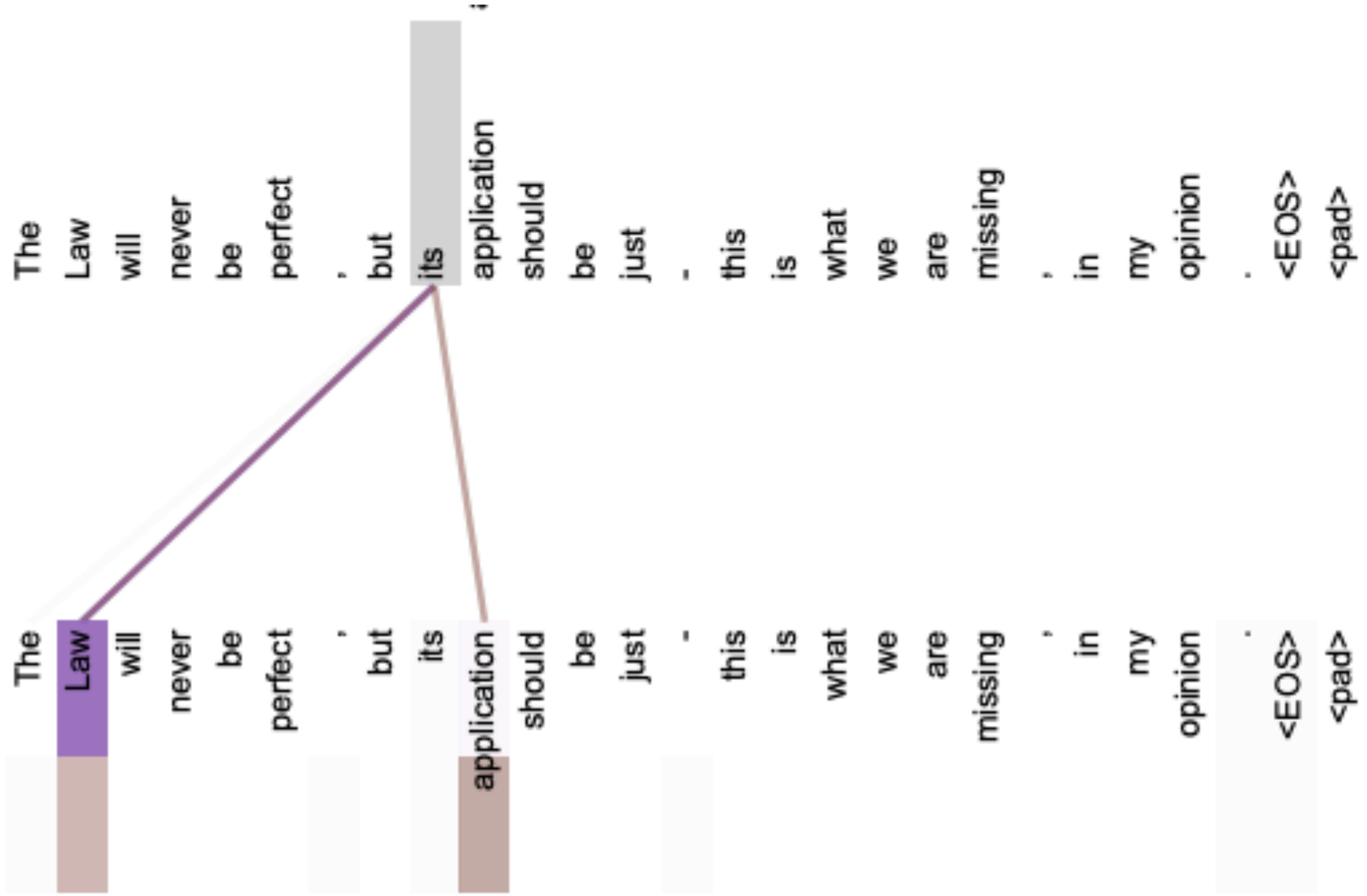
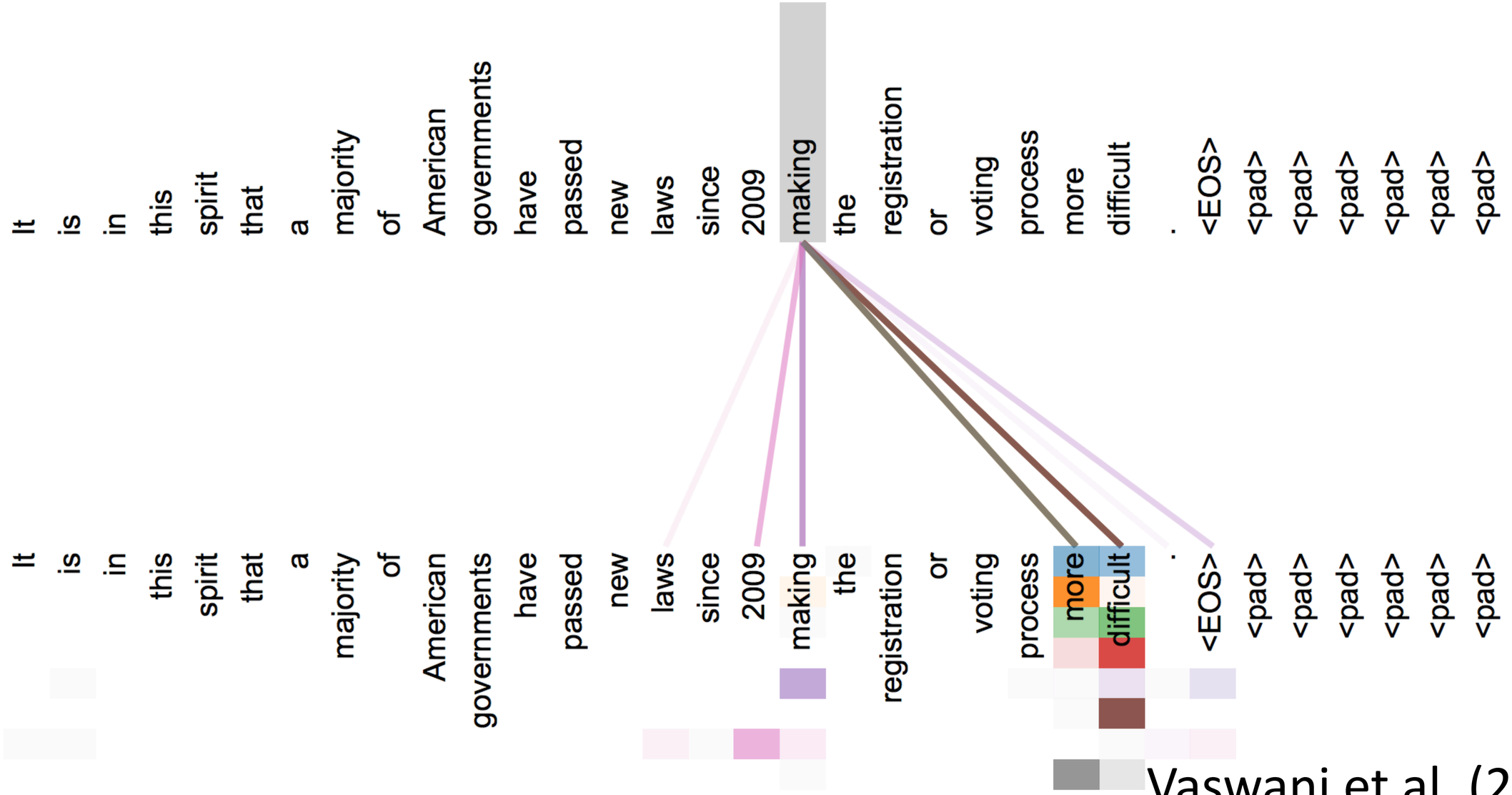


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

et al. (2017)

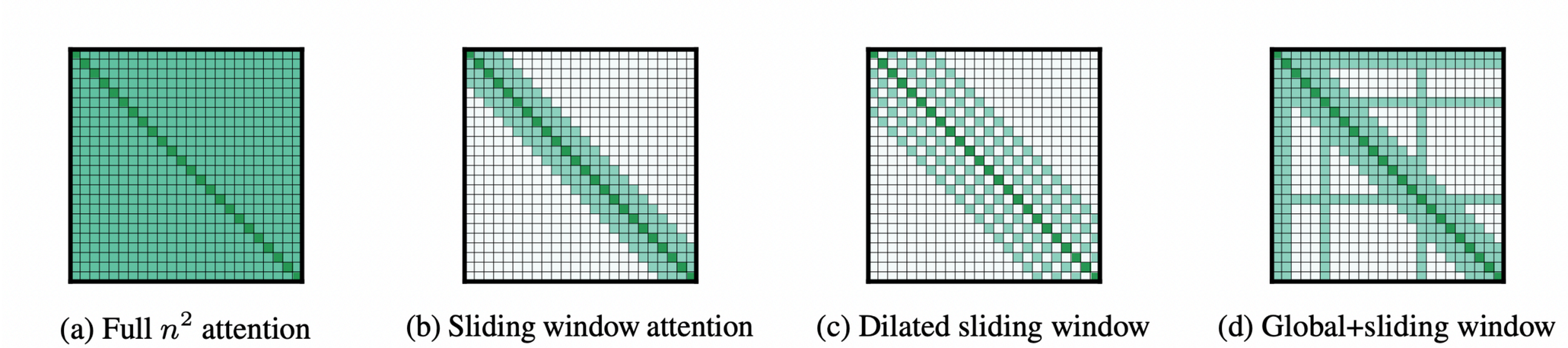
Visualization



Vaswani et al. (2017)

Limitations?

- O(n^2) self-attention computation can get expensive when the sequence is long (n=the length of the input sequence)



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

[Beltagy et al, 2020, Zaheer et al, 2020, many others..]

Takeaways

- RNN requires sequential processing, CNN enables parallel processing
- Both CNN and RNN assumes locality
- Transformers are strong, general models we'll see frequently
- Next: Contextualized word embedding
  - How language models, RNN, Transformers are used for many downstream tasks

Recap: Word embeddings

- Learn a ***continuous, dense*** vector for each word type.
- Always the same vector, regardless of in which context the word appears

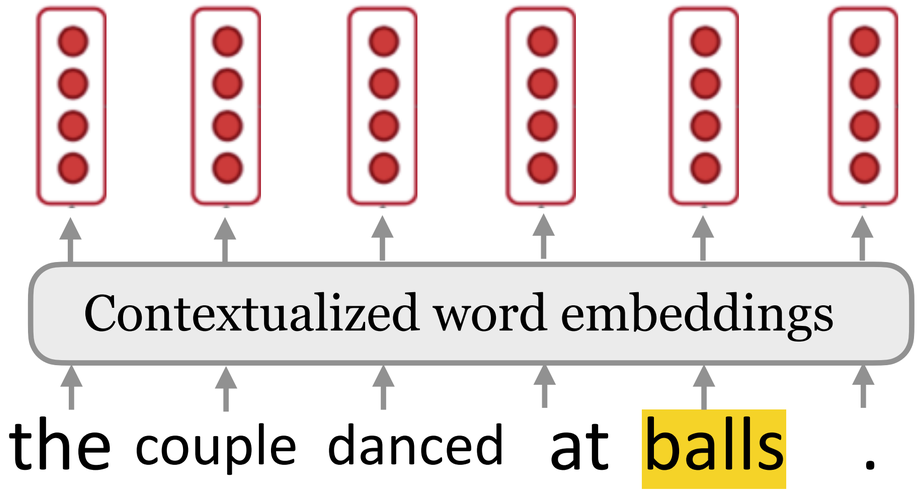
$$\text{employees} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 10.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

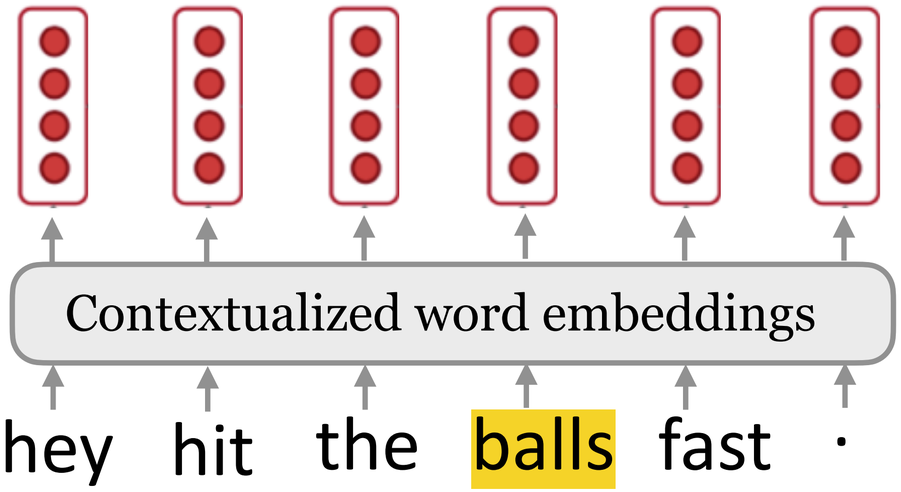
Motivation: Context-dependent Embeddings

- How to handle different word senses?

mouse^1 : .... a *mouse* controlling a computer system in 1968.
mouse^2 : .... a quiet animal like a *mouse*
bank^1 : ...a *bank* can hold the investments in a custodial account ...
bank^2 : ...as agriculture burgeons on the east *bank*, the river ...

- *Context-sensitive* word embeddings: depend on rest of the sentence





$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$

Peters et al. (2018)

Core idea

- Distributional Semantics:

  You know the meaning of the words by the company it keeps

C1: A bottle of ___ is on the table.

C2: Everybody likes ___.

C3: Don’t have ___ before you drive.

C4: We make ___ out of corn.

- Build context-dependent word embedding representation

Recall: Language Modeling

- Setup: Assume a (finite) vocabulary of words, (infinite) set of sentences.

$$\mathcal{V} = \{\text{the, a, man, telescope, Beckham, two, Madrid, ...}\}$$

$$\mathcal{V}^\dagger = \{\text{the, a, the a, the fan, the man, the man with the telescope, ...}\}$$

- Data: a training set of example sentences
- Task: estimate a probability distribution over sentences

$$\sum_{x \in \mathcal{V}^\dagger} p(x) = 1$$

and $p(x) \geq 0$ for all $x \in \mathcal{V}^\dagger$

$p(\text{the}) = 10^{-12}$

$p(\text{a}) = 10^{-13}$

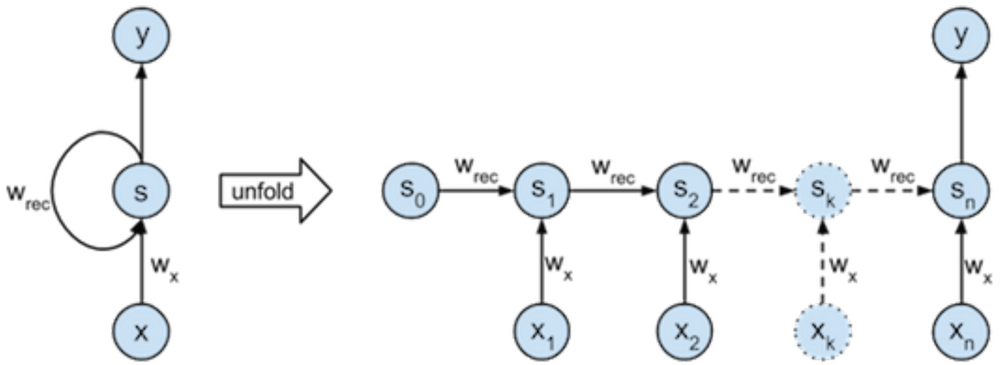
$p(\text{the fan}) = 10^{-12}$

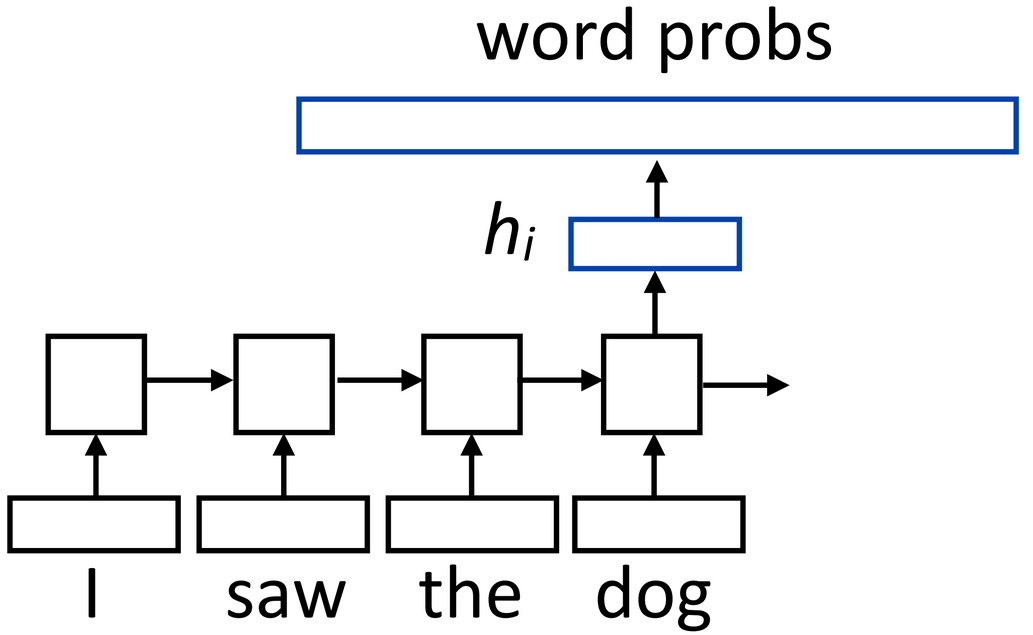
$p(\text{the fan saw Beckham}) = 2 \times 10^{-8}$

$p(\text{the fan saw saw}) = 10^{-15}$

...

Recurrent Neural Network LM



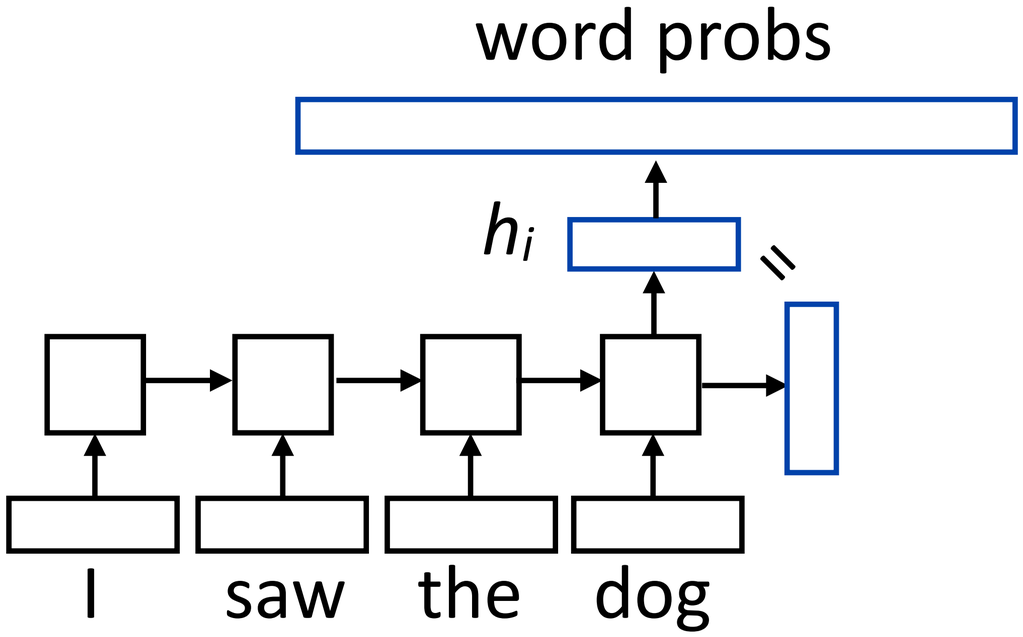


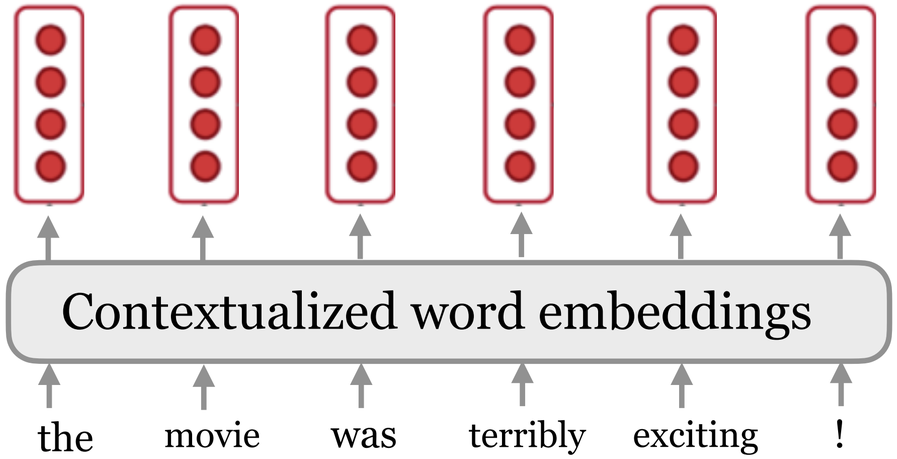
$$P(w_i \mid w_1, w_2 \ldots w_{i-1}) = \textbf{softmax}(Wh_{i-1})$$

Learned parameters:
vocab size x hidden dimension

Hidden state at position i-1

Embeddings from Language Models (ELMo)





$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$

Peters et al. (2018)

Why is language modeling a good objective?

- "Impossible" problem but bigger models seem to do better and better at distributional modeling (no upper limit yet)

- Successfully predicting next words requires modeling lots of different effects in text

Context: My wife refused to allow me to come to Hong Kong when the plague was at its height and –" "Your wife, Johanne? You are married at last ?" Johanne grinned. "Well, when a man gets to my age, he starts to need a few home comforts.
Target sentence: After my dear mother passed away ten years ago now, I became _____.
Target word: lonely

(Papernot et al., 2016)

Why is language modeling a good objective?

- "Impossible" … better at distrib…
- Succes… nt effects…

This week lecture:

Using language modeling objective to build a better representation for words

Use this new representation for all sorts of end tasks!

Context: M… Your wife, Johan… need a few home…
Target sent…
Target wor…

(Papernot et al., 2016)

Timeline of Pretrained LM

First general purpose LM: ELMo (2018)

Seq2Seq Pretraining: T5, BART(2019)

Efficient LM: ELECTRA / ALBERT (2020)

Even larger LM LM + search

Precursor to ELMo (2017) Using LM for sequence tagging

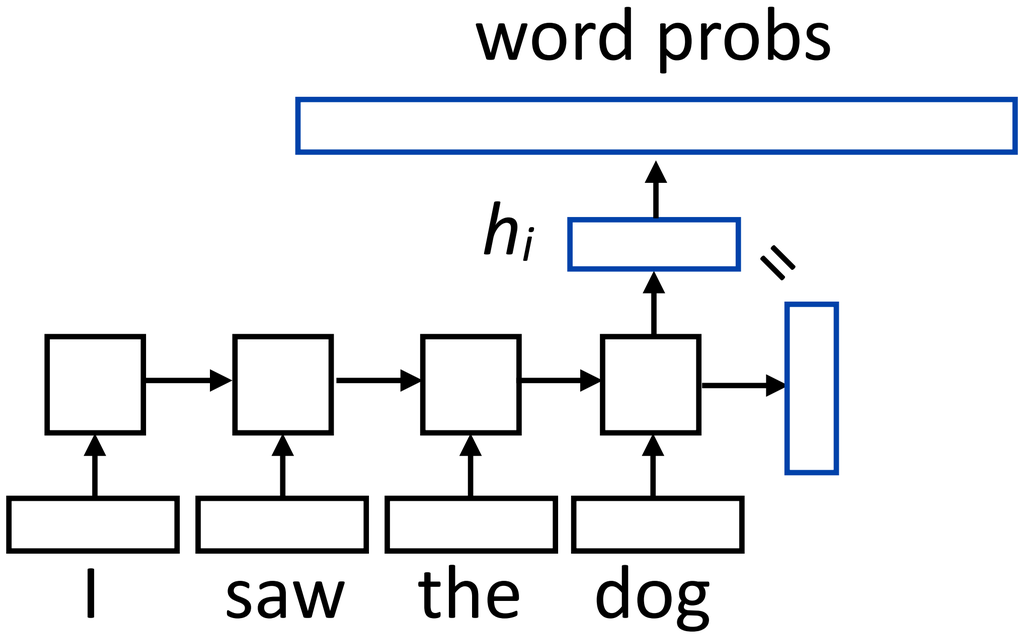
Masked LM: BERT (late 2018)

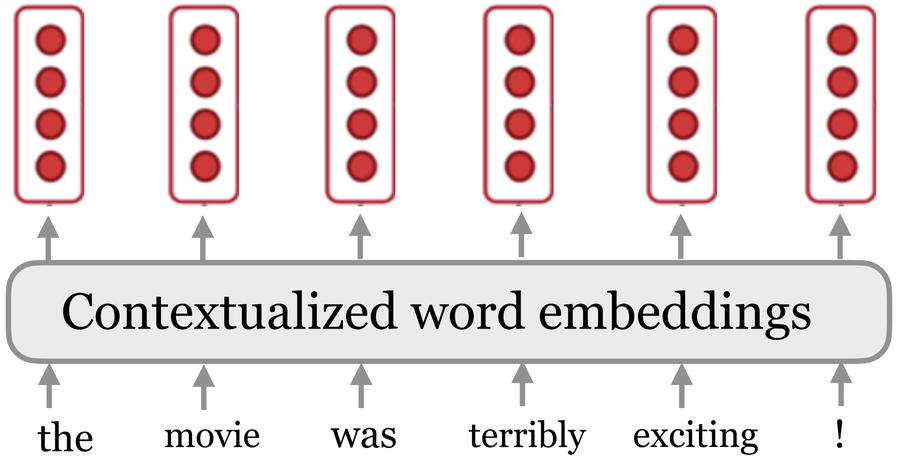
Multilingual Language Model: XLM (2019)

Larger LM: GPT3 (2020)

Multimodal LMs: Language / Vision / Audio (2019-)

Embeddings from Language Models (ELMo)

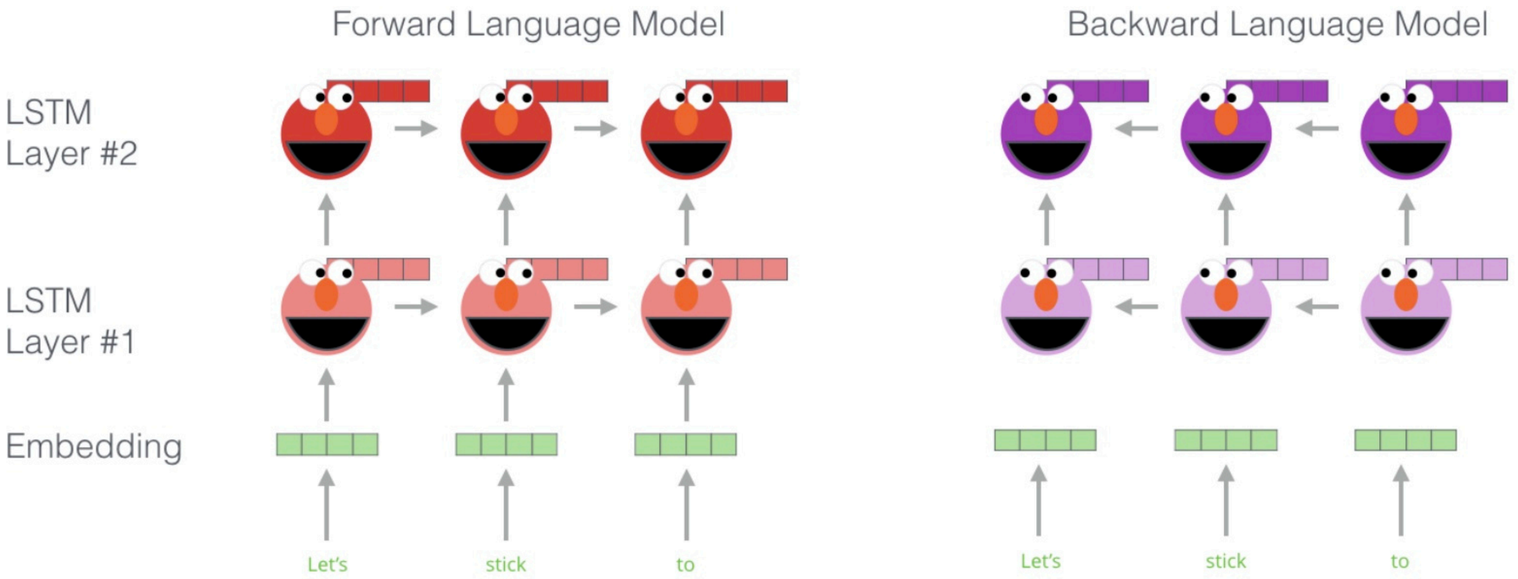




$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$

Peters et al. (2018)

Embeddings from Language Models (ELMo)



words in the sentence

$$\sum_{k=1}^{N} \Big(\log p(t_k \mid t_1, \ldots, t_{k-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) + \log p(t_k \mid t_{k+1}, \ldots, t_N; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s) \Big)$$

input

softmax

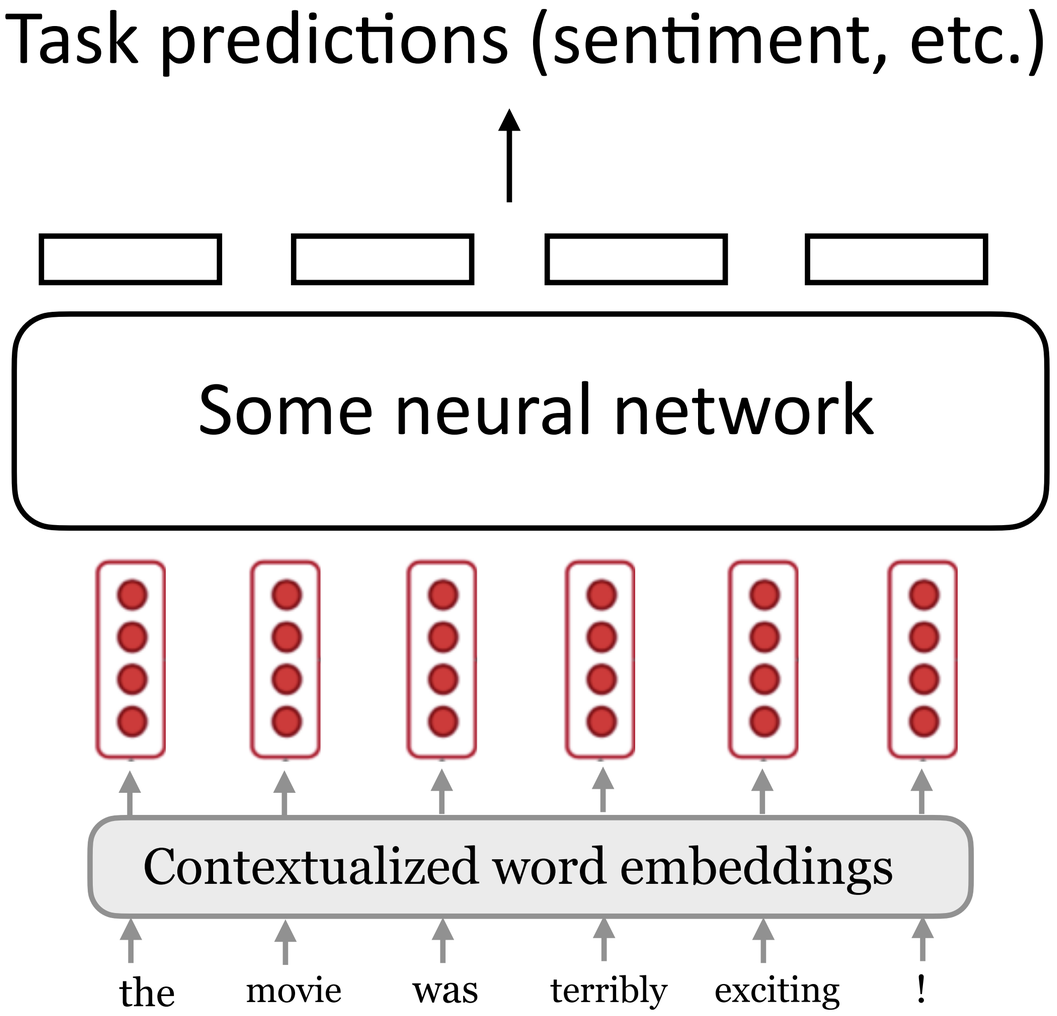
Peters et al. (2018)

How to apply ELMo?

- Take those embeddings and feed them into whatever architecture you want to use for your task
- *Frozen* embeddings: update the weights of your network but keep ELMo's parameters frozen
- *Plug ELMo into any (neural) NLP model: freeze all the LMs weights and change the input representation to:*

$$[\mathbf{x}_k; \mathbf{ELMo}_k^{task}]$$

(could also insert into higher layers)

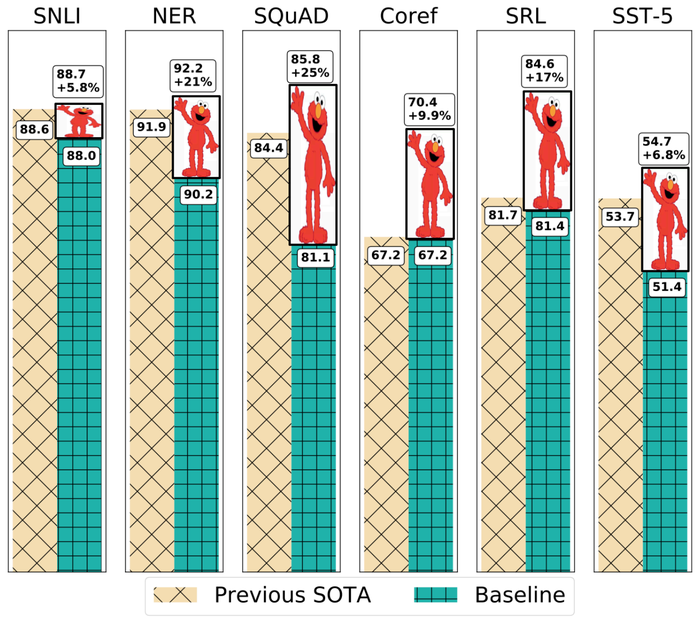


$$f : (w_1, w_2, \ldots, w_n) \longrightarrow \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$$

Experimental Results

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%

- SQuAD: question answering
- SNLI: natural language inference
- SRL: semantic role labeling
- Coref: coreference resolution
- NER: named entity recognition
- SST-5: sentiment analysis



ELMo architecture details

- Forward and backward LMs: 2 layers each
- Use character CNN to build initial word representation
  - 2048 char n-gram filters and 2 highway layers, 512 dim projection
- User 4096 dim hidden/cell LSTM states with 512 dim projections to next input
- A residual connection from the first to second layer
- Trained 10 epochs on **1B** Word Benchmark