TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2019

Lecture 5:
Contextualized Word Embeddings, Encoders, and Attention

Roadmap

- intro (1 lecture)
- **deep learning for NLP (5 lectures)**
- structured prediction: sequence labeling, syntactic and semantic parsing, dynamic programming (4 lectures)
- generative models, latent variables, unsupervised learning, variational autoencoders (2 lectures)
- Bayesian methods in NLP (2 lectures)
- Bayesian nonparametrics in NLP (2 lectures)
- review & other topics (1 lecture)

Today

- contextualized word embeddings
- sentence encoders & attention

Today

- contextualized word embeddings
- sentence encoders & attention

Recap

- last Wednesday we discussed methods for subword modeling for both word embeddings (RNNs, CNNs, character *n*-grams) and for generation (BPE)
- we also talked about multisense word embeddings

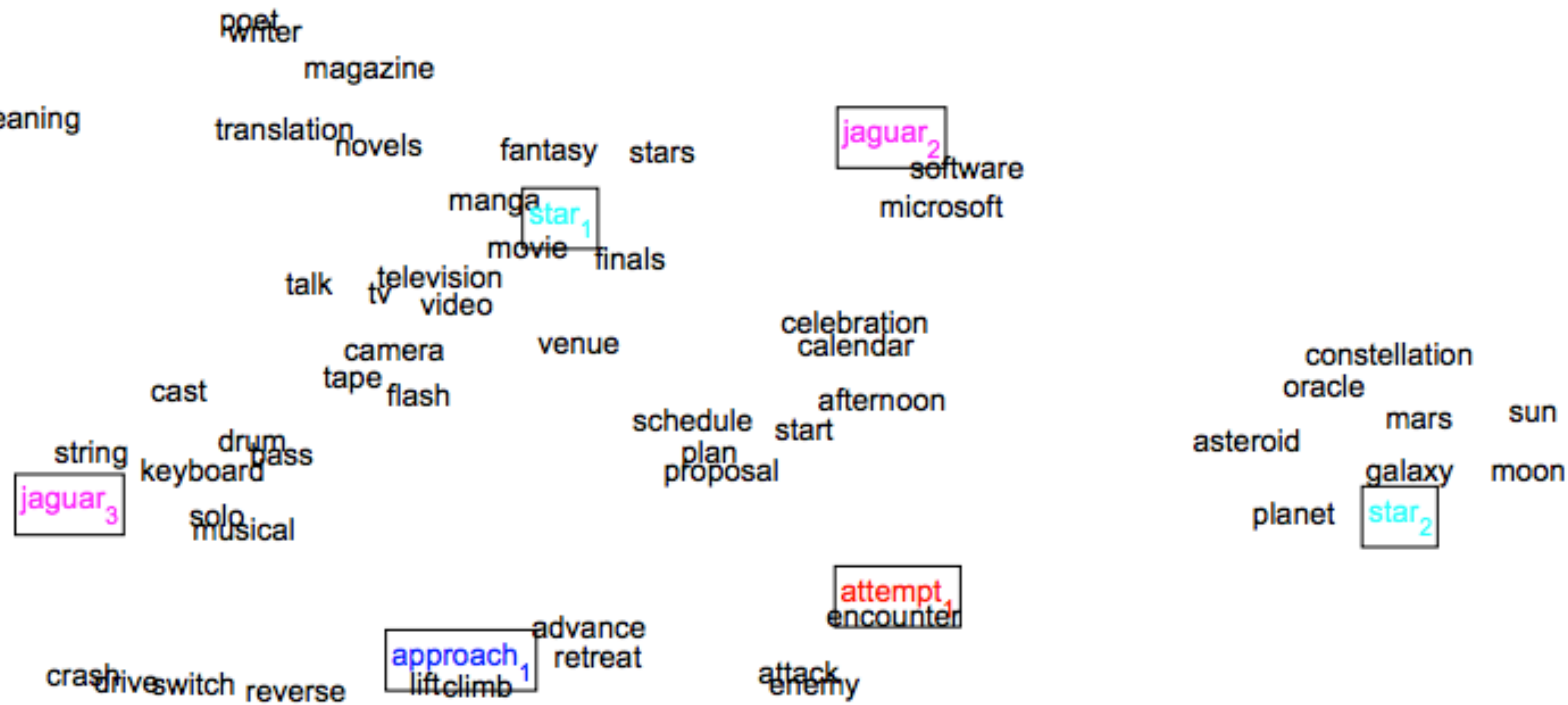
Other Work on Word Embeddings

- using subword information (e.g., characters) in word embeddings

- multiple embeddings for a single word type corresponding to different word senses

- tailoring embeddings using particular resources or for particular NLP tasks

Multisense Word Embeddings



Huang et al. (2012): *Improving Word Representations Via Global Context And Multiple Word Prototypes*

Multisense Word Embeddings

- limitations:
  - need a way to label senses or cluster word tokens in training data (and for downstream tasks)
  - fragments training data, so more may be needed for estimating word embeddings
  - unlikely to get good clusters for rare word types
  - unclear if sense-specific embeddings are useful for downstream tasks

Contextualized Word Embeddings

key idea:

define word embedding function based on context, e.g.:

i am **so** thrilled about this

fighting off a headache **so** i can work

does not need sense inventory or clustering

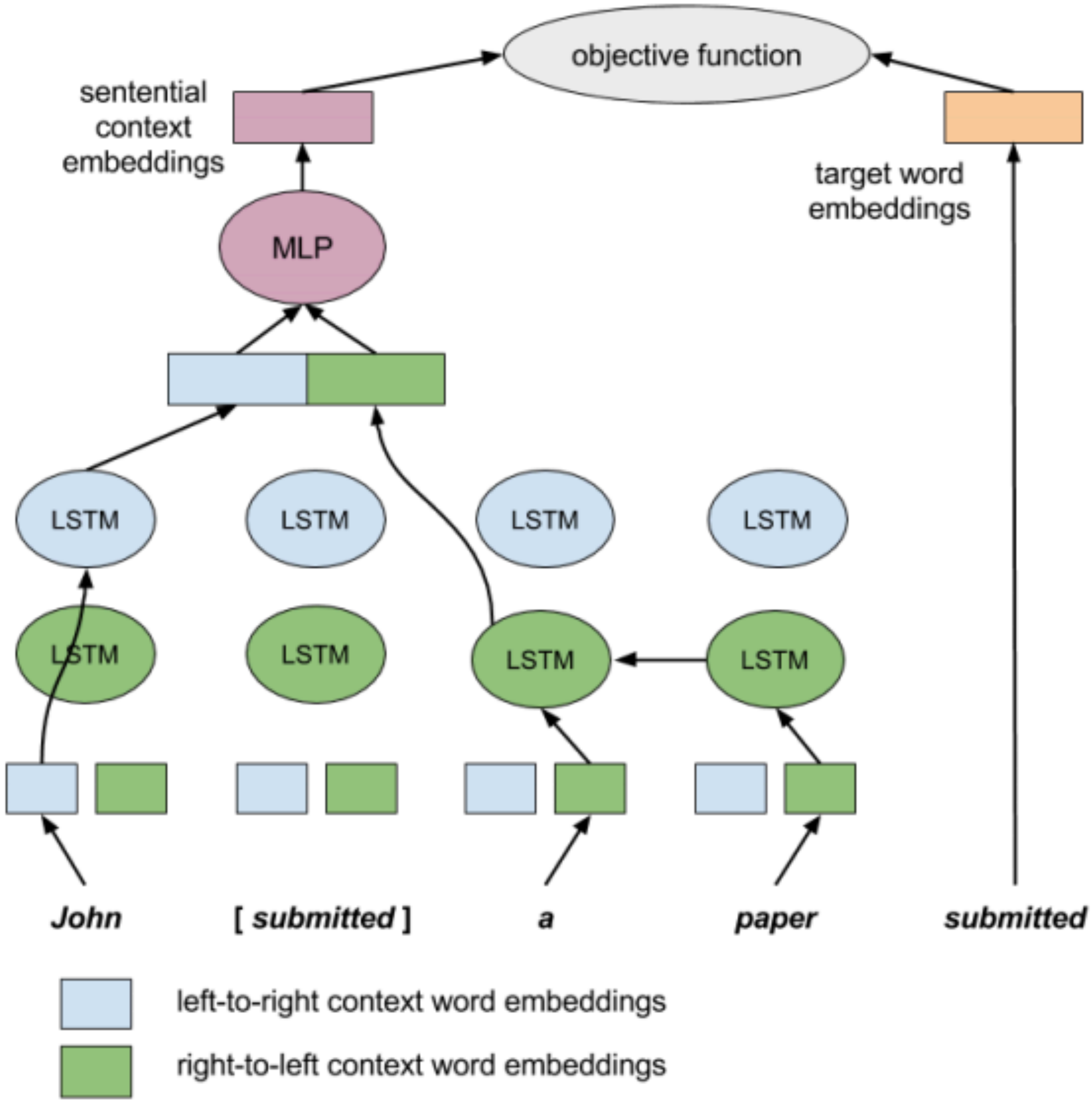
Contextualized Word Embeddings

- architectures vary, but typically RNNs used to encode a sentence, then hidden vector for word used as "contextualized" embedding

- learned from parallel text (sentences & translations):
  - Kawakami & Dyer (2015), McCann et al. (2017)

- learned from monolingual text:
  - Melamud et al. (2016), Peters et al. (2017), Tu et al. (2017)

context2vec



Melamud et al. (2016): *context2vec: Learning Generic Context Embedding with Bidirectional LSTM*

CoVe (context vectors)

- train English→German neural translation model
- use hidden vectors of English encoder as contextualized word embeddings

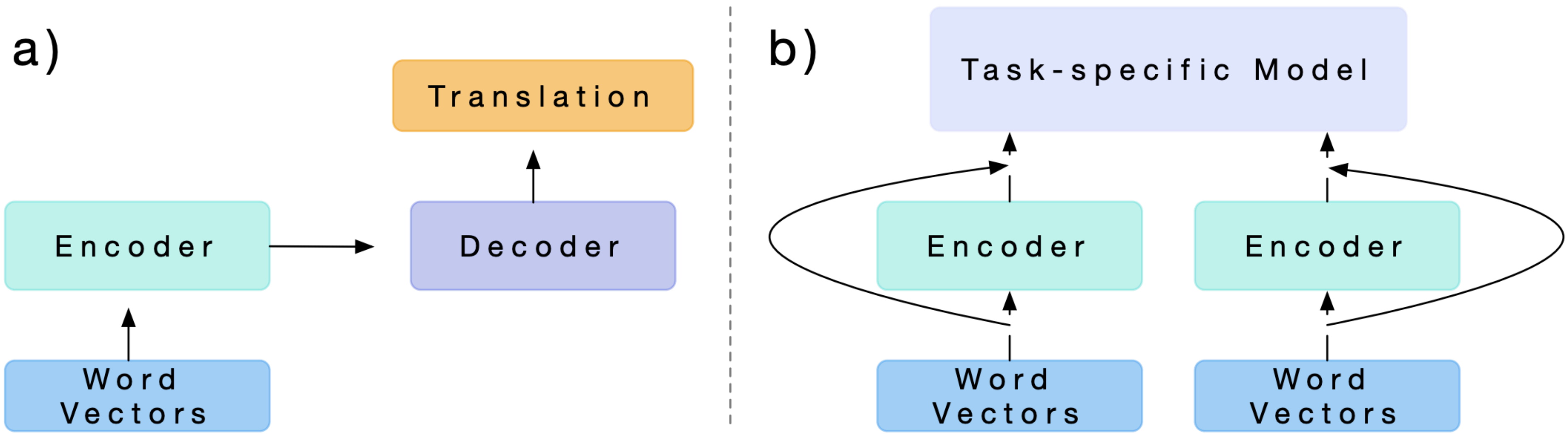


Figure 1: We a) train a two-layer, bidirectional LSTM as the encoder of an attentional sequence-to-sequence model for machine translation and b) use it to provide context for other NLP models.

McCann et al. (2017): *Learned in Translation: Contextualized Word Vectors*

Contextualized Word Embeddings with Autoencoders

- encode a window of text to a vector (using a feed-forward or recurrent net), decode words

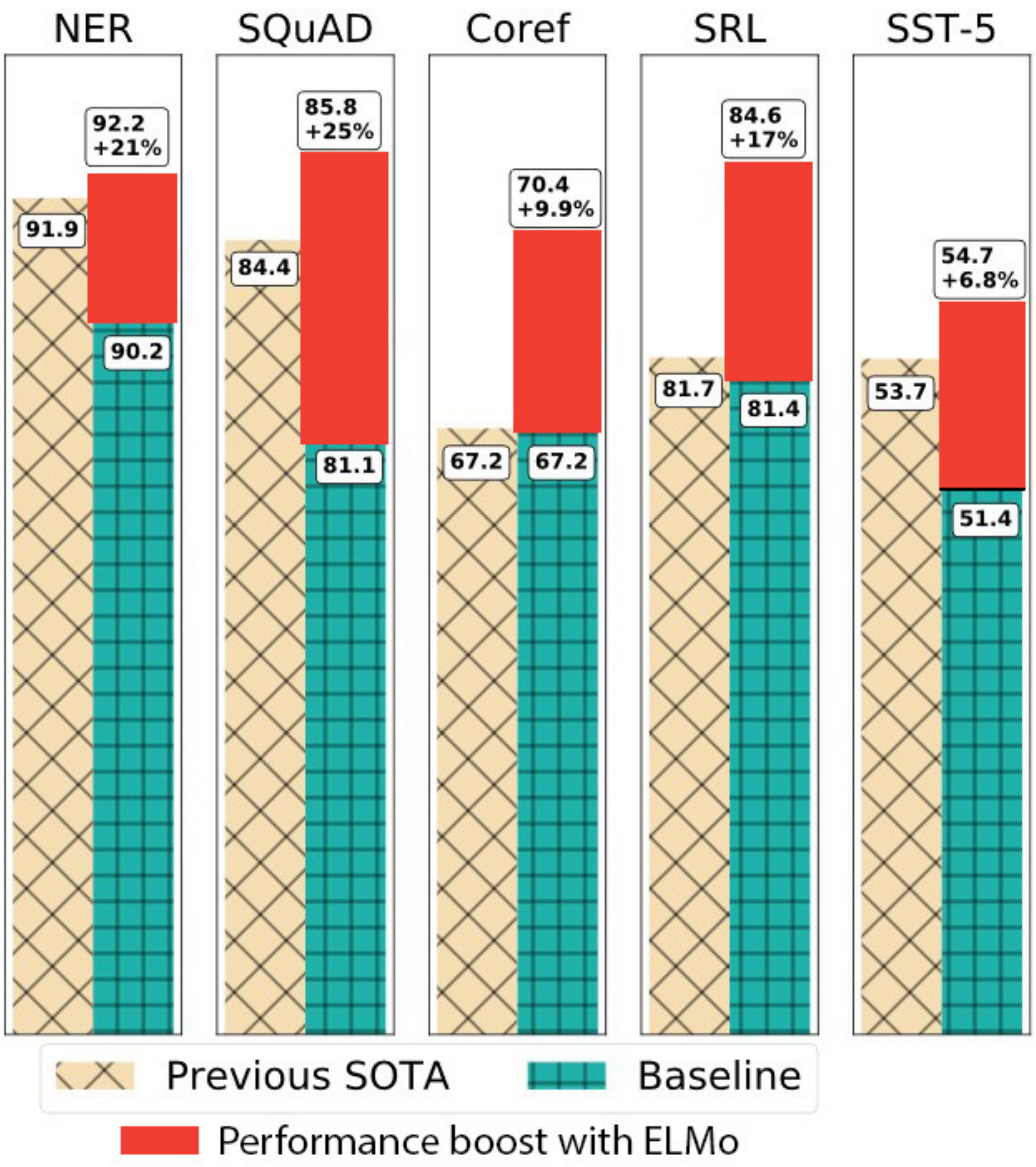
Q	my first **one was like 2 minutes long and** has	Q	**jus listenin 2 mr hudson and** drake crazyness
1	my fav **place- was there 2 years ago and** am	1	@mention **deadddddd u go 2 mlk high up** n
2	thought it **was more like 2 ..... either way** , i	2	only a **cups tho tryin 2 feed the whole** family
3	to **backup everything from 2 years before i**	3	bored on mars **i kum down 2 earth ... yupp** !!
4	i **slept for like 2 sec lol .** freakin chessy	4	i miss **you i trying 2 looking oud my** mind girl
Q	the lines **: i am so thrilled about this** . may	Q	fighting **off a headache so i can work** on my
1	and work **. i am so glad you asked** . let	1	im **on my phone so i cant see** who @mention
2	**i was so excited to sleep** in tomorrow	2	did some **things that hurt so i guess i** was doing
3	**@mention that is so funny ! i** know which	3	my **phone keeps beeping so i know ralph** must
4	little girl **! i was so touched when she** called	4	randomly obsessed **with this song so i bought it**

Table 1: Query tokens of two polysemous words and their four nearest neighboring tokens. The target token is underlined and the encoder context (3 words to either side) is shown in bold. See text for details.

Tu et al. (2017): *Learning to Embed Words in Context for Syntactic Tasks*

ELMo

(Embeddings from Language Models)



Peters et al. (2018): *Deep contextualized word representations*

ELMo

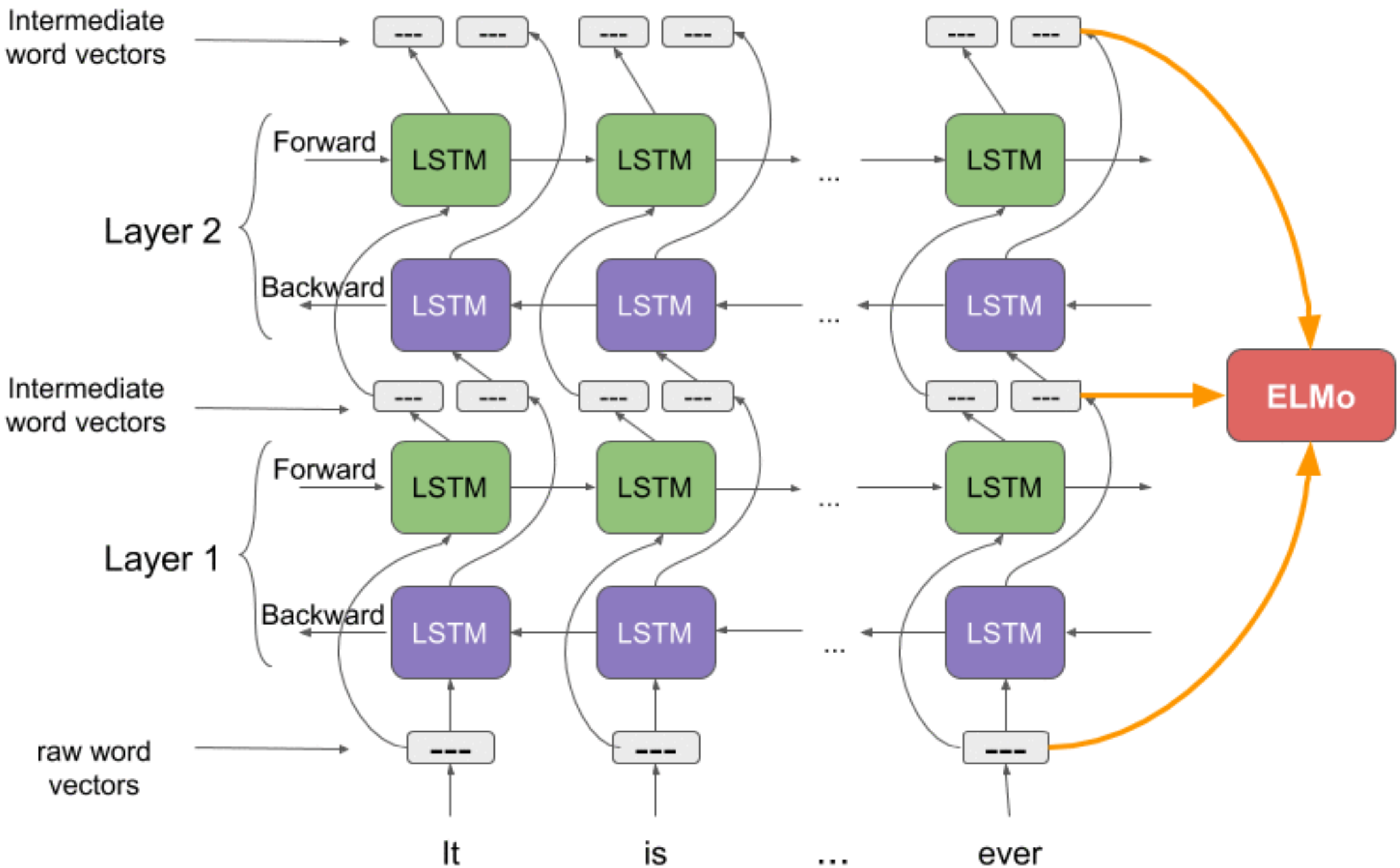


Figure credit: Analytics Vidhya
www.analyticsvidhya.com/blog/2019/03/learn-to-use-elmo-to-extract-features-from-text/

ELMo Details

- character CNN to encode each word (no word embeddings used)
- forward and backward LSTMs trained as language models
- some tied parameters:
  - character CNN parameters tied across directions
  - softmax output parameters tied across directions
  - LSTM parameters separate for each direction

Peters et al. (2018): *Deep contextualized word representations*

More Details

- 2 LSTM layers, 4096 units in hidden vectors
- residual connection
- 512-dimensional projection layers
- word representation module:
  - 2048 character n-gram convolutional filters
  - 2 highway layers
  - linear projection down to a 512-dim representation for a word

Peters et al. (2018): *Deep contextualized word representations*

Using ELMo for Tasks

For inclusion in a downstream model, ELMo collapses all layers in R into a single vector, $\mathbf{ELMo}_k = E(R_k; \mathbf{\Theta}_e)$. In the simplest case, ELMo just selects the top layer, $E(R_k) = \mathbf{h}_{k,L}^{LM}$, as in TagLM (Peters et al., 2017) and CoVe (McCann et al., 2017). More generally, we compute a task specific weighting of all biLM layers:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}. \quad (1)$$

In (1), $\mathbf{s}^{task}$ are softmax-normalized weights and the scalar parameter γ^{task} allows the task model to scale the entire ELMo vector. γ is of practical importance to aid the optimization process (see supplemental material for details).

How do the layers differ?

- first layer better at POS tagging
- second layer better for word sense prediction

Model	**F_1**
WordNet 1st Sense Baseline	65.9
Raganato et al. (2017a)	69.9
Iacobacci et al. (2016)	**70.1**
CoVe, First Layer	59.4
CoVe, Second Layer	*64.7*
biLM, First layer	67.4
biLM, Second layer	*69.0*

Table 5: All-words fine grained WSD F_1. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

Model	**Acc.**
Collobert et al. (2011)	97.3
Ma and Hovy (2016)	97.6
Ling et al. (2015)	**97.8**
CoVe, First Layer	*93.3*
CoVe, Second Layer	92.8
biLM, First Layer	*97.3*
biLM, Second Layer	96.8

Table 6: Test set POS tagging accuracies for PTB. For CoVe and the biLM, we report scores for both the first and second layer biLSTMs.

Peters et al. (2018): *Deep contextualized word representations*

Today

- contextualized word embeddings
- sentence encoders & attention

Encoders

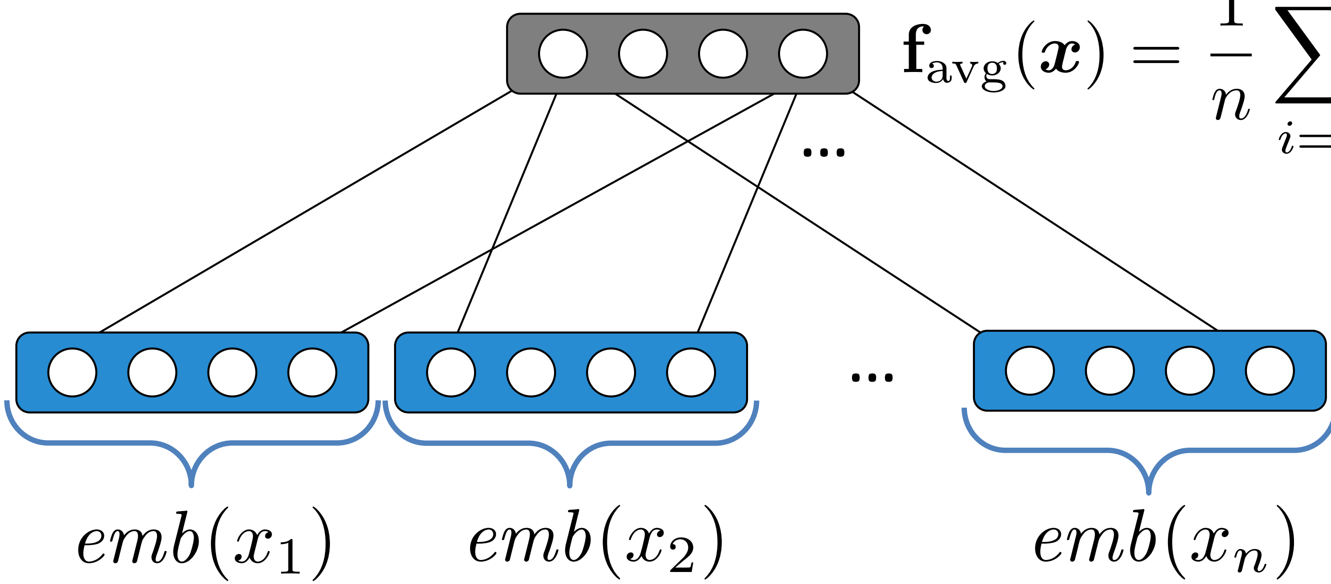
- many NLP tasks require us to form fixed-length representations of sentences (or paragraphs, documents, etc.)

- encoder = neural network compositional functional architecture that represents a sequence as a vector

A Simple Encoder: Word Averaging

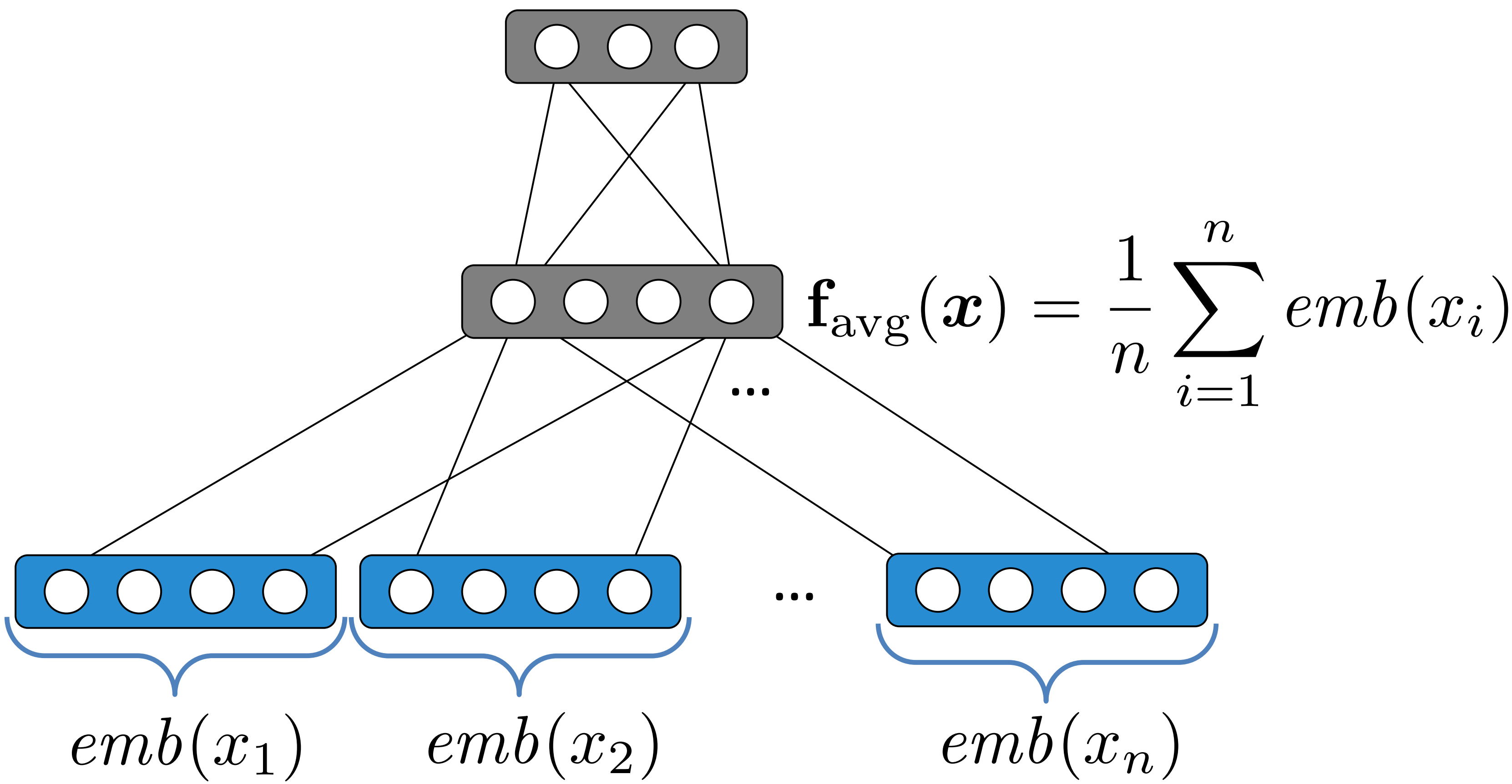
- represent word sequence $\boldsymbol{x}$ by averaging its word embeddings:

$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

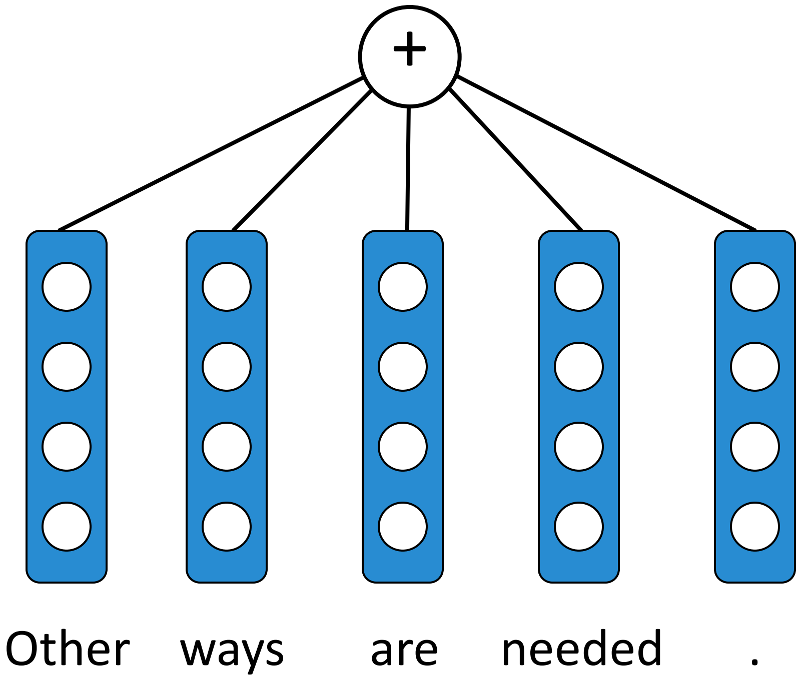


Adding Hidden Layers

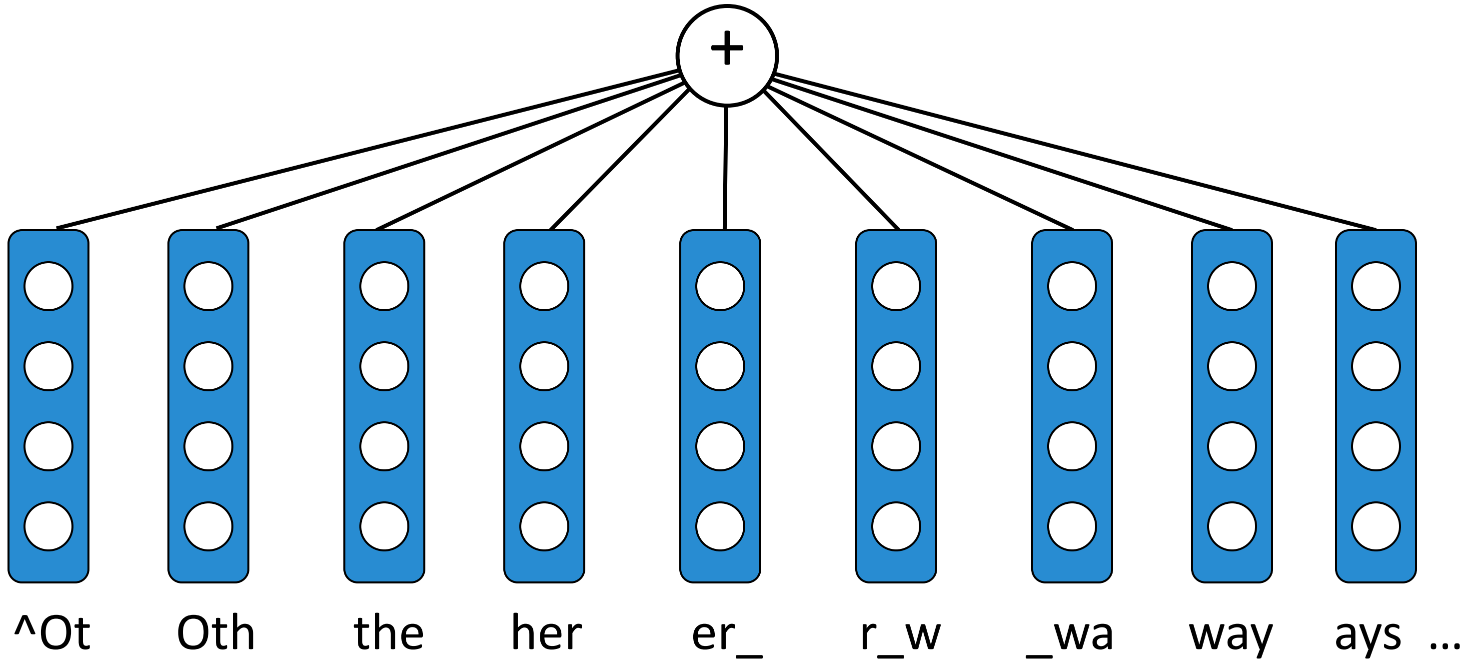
- deep averaging network (DAN; Iyyer et al., 2015)



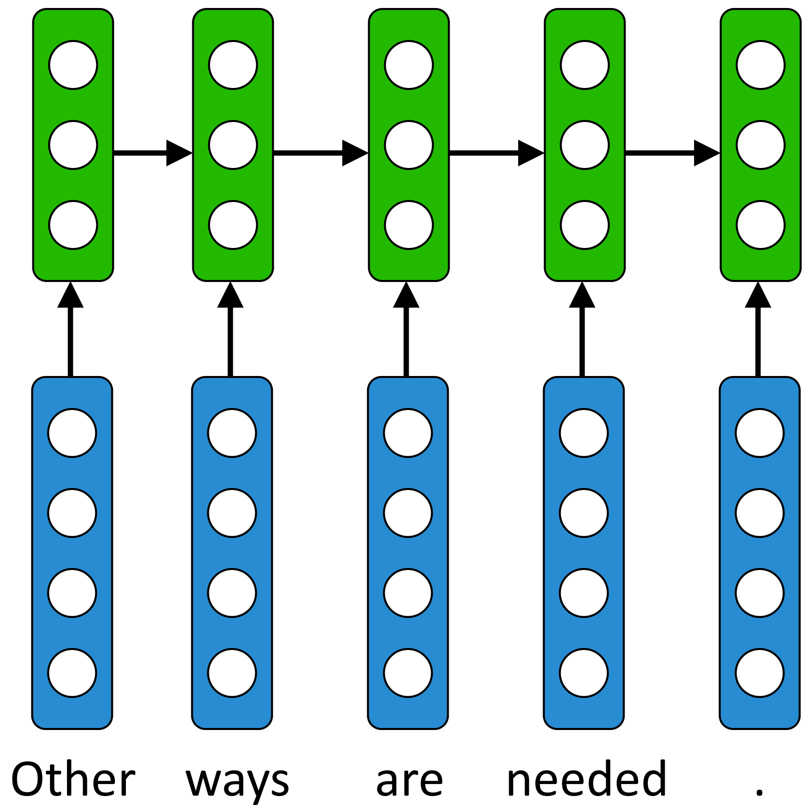
- Word embedding average:



- Character trigram embedding average:



- Recurrent Neural Networks:
  - run RNN over sequence
  - use average of hidden states or final hidden state as sequence representation



- Convolutional Neural Networks:
  - convolutional layers with *n*-gram filters followed by pooling

Recursive Neural Networks

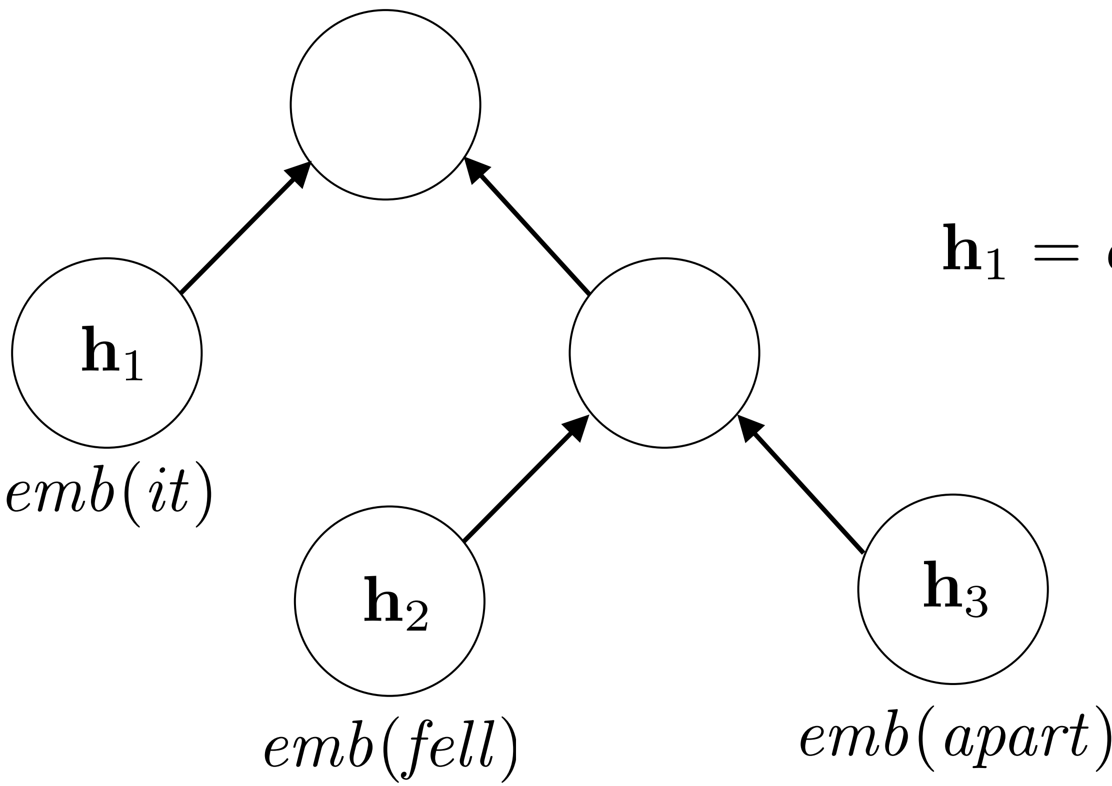
$x =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

Recursive Neural Networks

$\boldsymbol{x} =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

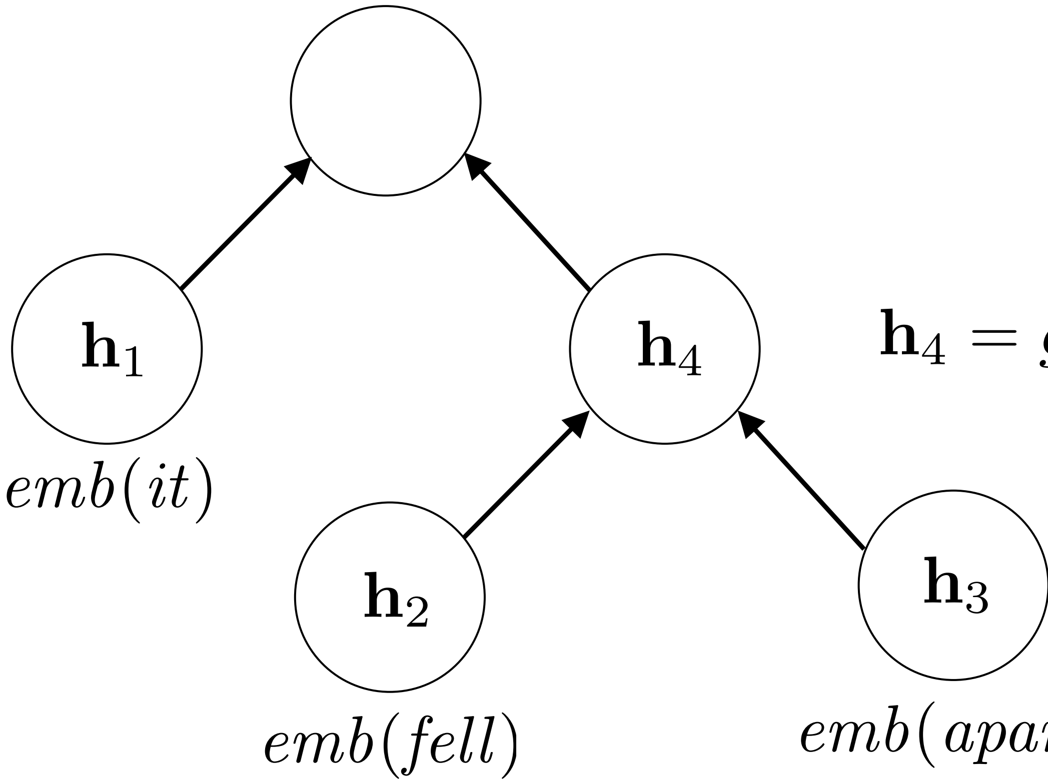


$$\mathbf{h}_1 = emb(it)$$

Recursive Neural Networks

$x =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

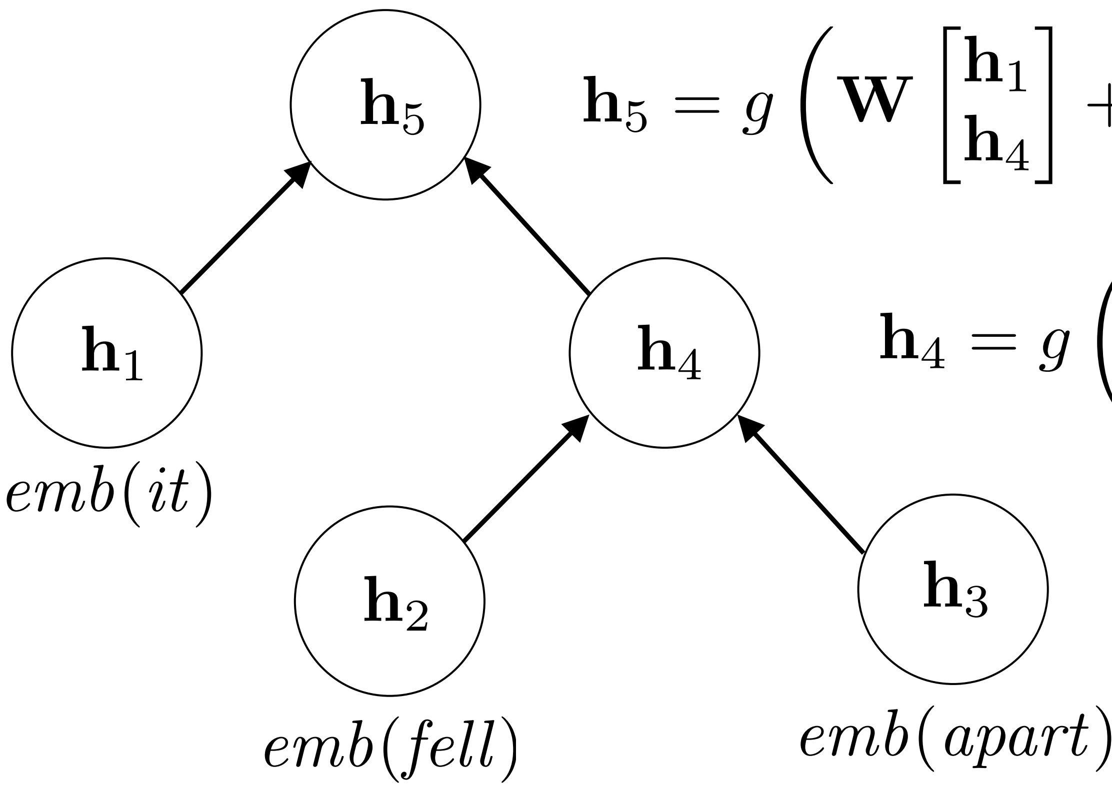


$$\mathbf{h}_4 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_2 \\ \mathbf{h}_3\end{bmatrix} + \mathbf{b}\right)$$

Recursive Neural Networks

$\boldsymbol{x} =$ *it fell apart*

- run a syntactic parser on the sentence
- construct vector recursively at each split point:

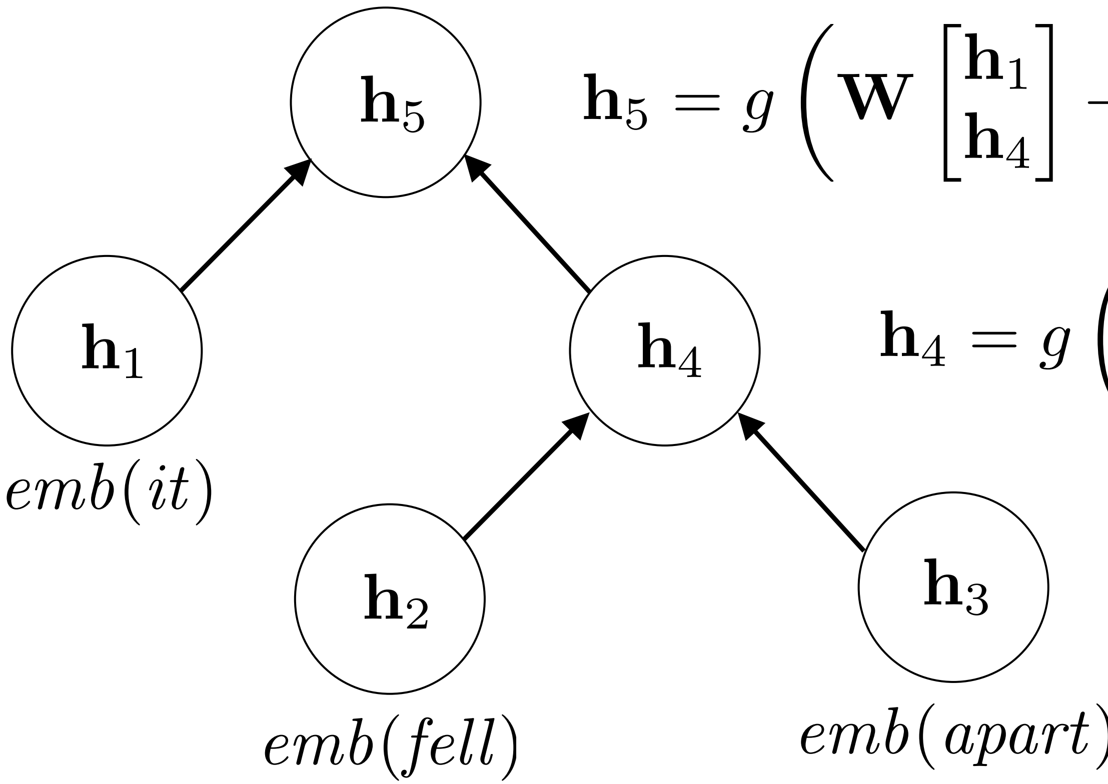


$$\mathbf{h}_5 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_1\\ \mathbf{h}_4\end{bmatrix} + \mathbf{b}\right)$$

$$\mathbf{h}_4 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_2\\ \mathbf{h}_3\end{bmatrix} + \mathbf{b}\right)$$

Recursive Neural Networks

- same parameters used at every split point
- order of children matters (different weights used for left and right child)



$$\mathbf{h}_5 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_1\\ \mathbf{h}_4\end{bmatrix} + \mathbf{b}\right)$$

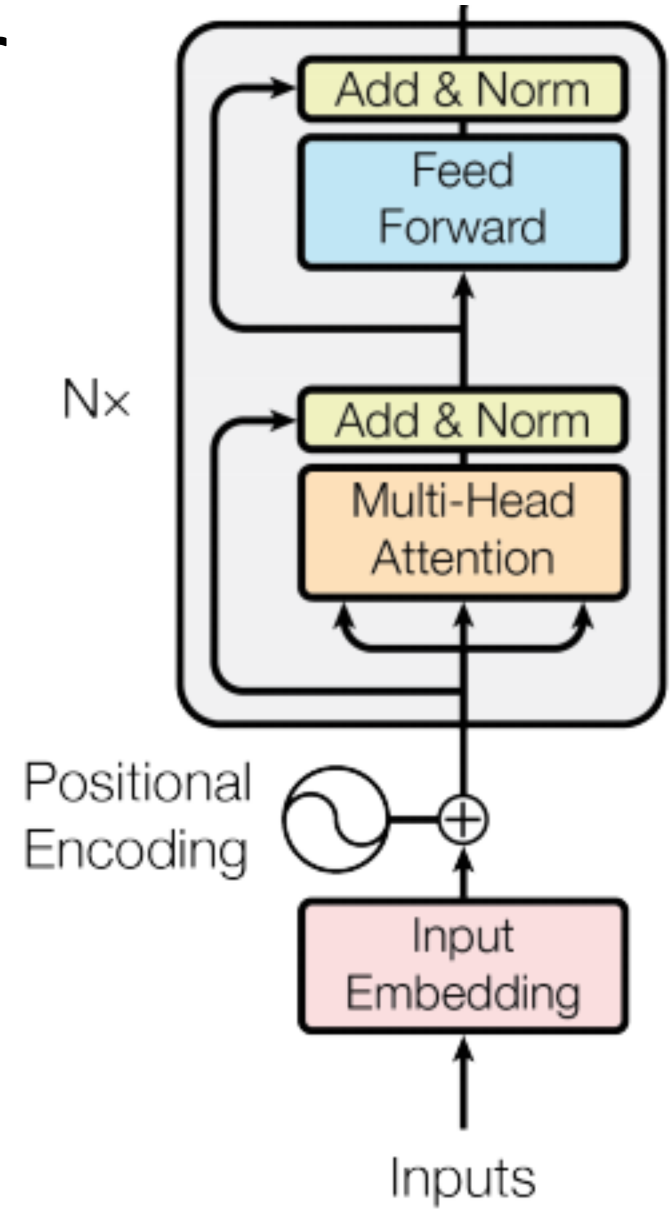
$$\mathbf{h}_4 = g\left(\mathbf{W}\begin{bmatrix}\mathbf{h}_2\\ \mathbf{h}_3\end{bmatrix} + \mathbf{b}\right)$$

Improvements to Recursive NNs

- gating in composition function (“tree LSTMs”)
- methods that automatically produce composition trees instead of requiring a parser

Transformer

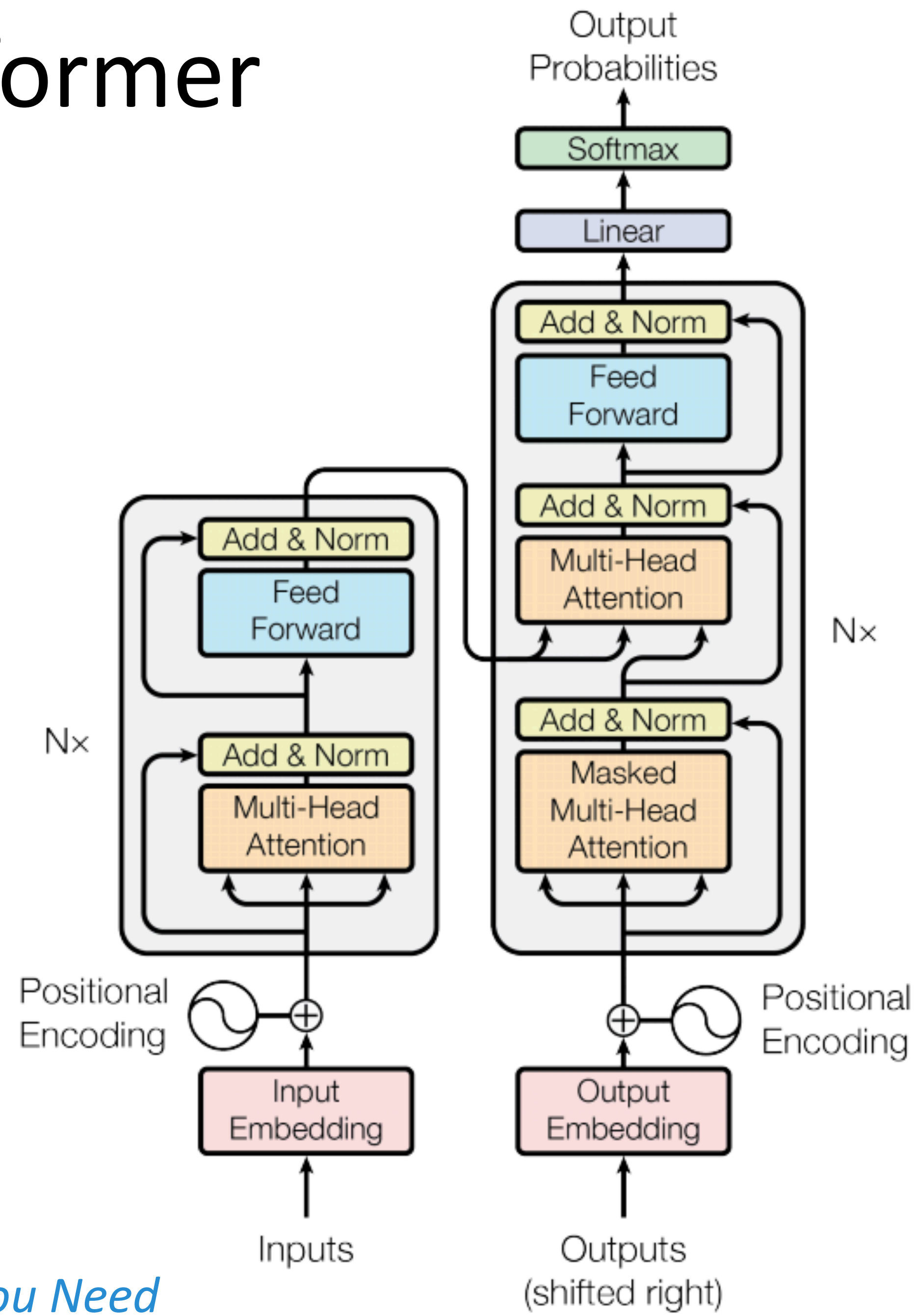
- effective encoder for text sequences (and other data)
- no recurrent/convolutional modules
- only attention (various forms)
- we'll discuss elements of attention-based neural architectures to build up to the transformer



Vaswani et al. (2017): *Attention Is All You Need*

Transformer

- initially developed for a setting with both encoding and decoding; we will discuss decoding on Wednesday



Vaswani et al. (2017): *Attention Is All You Need*

Attention

- attention is a useful generic tool
- often used to replace a sum or average with an attention-weighted sum

Attention

- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x})\, emb(x_i)$$

Attention

- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} \underbrace{att(x_i, i, \boldsymbol{x})}_{\text{"attention" function, returns a scalar}} emb(x_i)$$

Attention

- e.g., for a word averaging encoder:

$$\mathbf{f}_{\text{avg}}(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} emb(x_i)$$

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x})\, emb(x_i)$$

many attention functions are possible!
often assume:

$$\sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) = 1$$

Example Attention Function

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) emb(x_i)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\{\mathbf{w}^\top emb(x_i)\}$$

- introduces a new parameter vector **w** which is learned along with the word embeddings
- attention is normalized over the sentence length

Queries, Keys, and Values

- we can often think of attention functions in terms of these abstractions
- query = what you use to search
- key = the field that you're comparing to
- value = the field that you return

Vaswani et al. (2017): *Attention Is All You Need*

Analogy to Dictionaries

- query: key you are searching for
- dictionary contains <key, value> pairs

- look-up in a dictionary/hashmap can be interpreted as comparing the query to each key in the dictionary and returning the value for the key with the strongest match

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x})emb(x_i)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\{\mathbf{w}^\top emb(x_i)\}$$

- for this attention-weighted encoder,
  - query = ?
  - key = ?
  - value = ?

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) emb(x_i)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\{\mathbf{w}^\top emb(x_i)\}$$

- for this attention-weighted encoder,
  - query = $\mathbf{w}$
  - key = $emb(x_i)$
  - value = $emb(x_i)$

- considering attention as query/key/value suggests using different spaces for different roles
- e.g., we could use separate transformations of the embedding space for keys and values

- considering attention as query/key/value suggests using different spaces for different roles
- e.g., we could use separate transformations of the embedding space for keys and values:

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) \left(\mathbf{W}^{(v)} emb(x_i) \right)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\left\{ \mathbf{w}^\top \left(\mathbf{W}^{(k)} emb(x_i) \right) \right\}$$

- considering attention as query/key/value suggests using different spaces for different roles
- e.g., we could use separate transformations of the embedding space for keys and values:

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) \left(\mathbf{W}^{(v)} emb(x_i) \right)$$

value trans-
formation
matrix

$$att(x_i, i, \boldsymbol{x}) \propto \exp \left\{ \mathbf{w}^\top \left(\mathbf{W}^{(k)} emb(x_i) \right) \right\}$$

key trans-
formation
matrix

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x}) \left(\mathbf{W}^{(v)} emb(x_i)\right)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\left\{\mathbf{w}^\top \left(\mathbf{W}^{(k)} emb(x_i)\right)\right\}$$

- for this attention-weighted encoder,
  - query = $\mathbf{w}$
  - key = $\mathbf{W}^{(k)} emb(x_i)$
  - value = $\mathbf{W}^{(v)} emb(x_i)$

Multi-Head Attention

- we may want to learn multiple attention functions in parallel
- why? so that they can learn complementary functionality for the task

Vaswani et al. (2017): *Attention Is All You Need*

Multi-Head Attention

- we may want to learn multiple attention functions in parallel
- why? so that they can learn complementary functionality for the task

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{J} att_j(x_i, i, \boldsymbol{x}) \left(\mathbf{W}_j^{(v)} emb(x_i) \right)$$

$$att_j(x_i, i, \boldsymbol{x}) \propto \exp \left\{ \mathbf{w}_j^{\top} \left(\mathbf{W}_j^{(k)} emb(x_i) \right) \right\}$$

Vaswani et al. (2017): *Attention Is All You Need*

Multi-Head Attention

- we may want to learn multiple attention functions in parallel
- why? so that they can learn complementary functionality for the task

$$\mathbf{f}_{\text{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} \sum_{j=1}^{J} att_j(x_i, i, \boldsymbol{x}) \left(\mathbf{W}_j^{(v)} emb(x_i) \right)$$

$$att_j(x_i, i, \boldsymbol{x}) \propto \exp\left\{ \mathbf{w}_j^\top \left(\mathbf{W}_j^{(k)} emb(x_i) \right) \right\}$$

J attention heads

Vaswani et al. (2017): *Attention Is All You Need*

Multi-Head Attention

- in the transformer, each attention head uses projections to lower dimension, followed by concatenation of the outputs from each head

Vaswani et al. (2017): *Attention Is All You Need*

Transformer

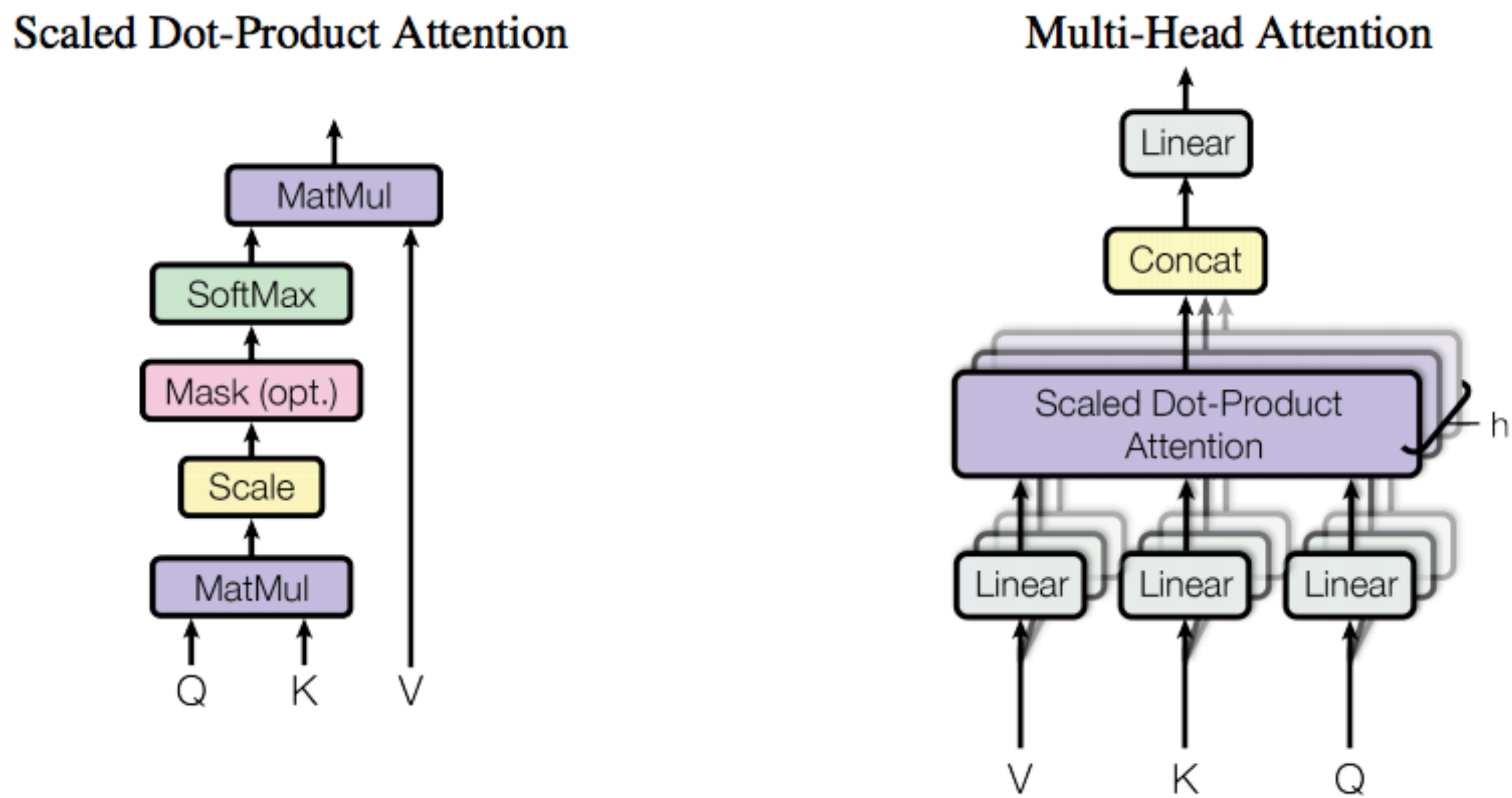


Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Vaswani et al. (2017): *Attention Is All You Need*

Self-Attention

- rather than learning attention weight vectors **w** to serve as query vectors, use the words themselves as the queries!

$$\mathbf{f}_{\mathrm{att}}(\boldsymbol{x}) = \sum_{i=1}^{n} att(x_i, i, \boldsymbol{x})\, emb(x_i)$$

$$att(x_i, i, \boldsymbol{x}) \propto \exp\left\{\sum_{j=1}^{n} emb(x_i)^\top emb(x_j)\right\}$$

Vaswani et al. (2017): *Attention Is All You Need*

Self-Attention

- many possibilities for self-attention functions
- intuitively, the following weights a word based on how similar it is to all other words in the sequence:

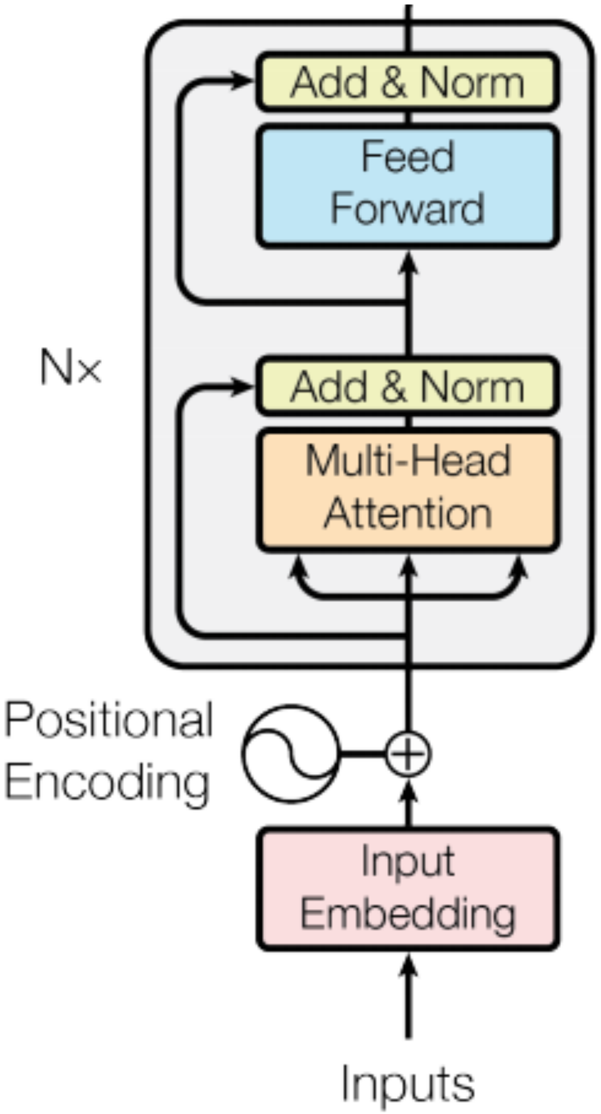
$$att(x_i, i, \boldsymbol{x}) \propto \exp\left\{\sum_{j=1}^{n} emb(x_i)^\top emb(x_j)\right\}$$

- can be combined with query/key/value-specific transformations and multiple heads

Vaswani et al. (2017): *Attention Is All You Need*

Word Position Information

- attention functions discussed so far do not use word position
- transformer uses embedding of word position that's added to word embedding in input
- compared predetermined & fixed sinusoidal positional embeddings to learned positional embeddings (similar performance)



Vaswani et al. (2017): *Attention Is All You Need*

Sinusoidal Word Position Encodings

In this work, we use sine and cosine functions of different frequencies:

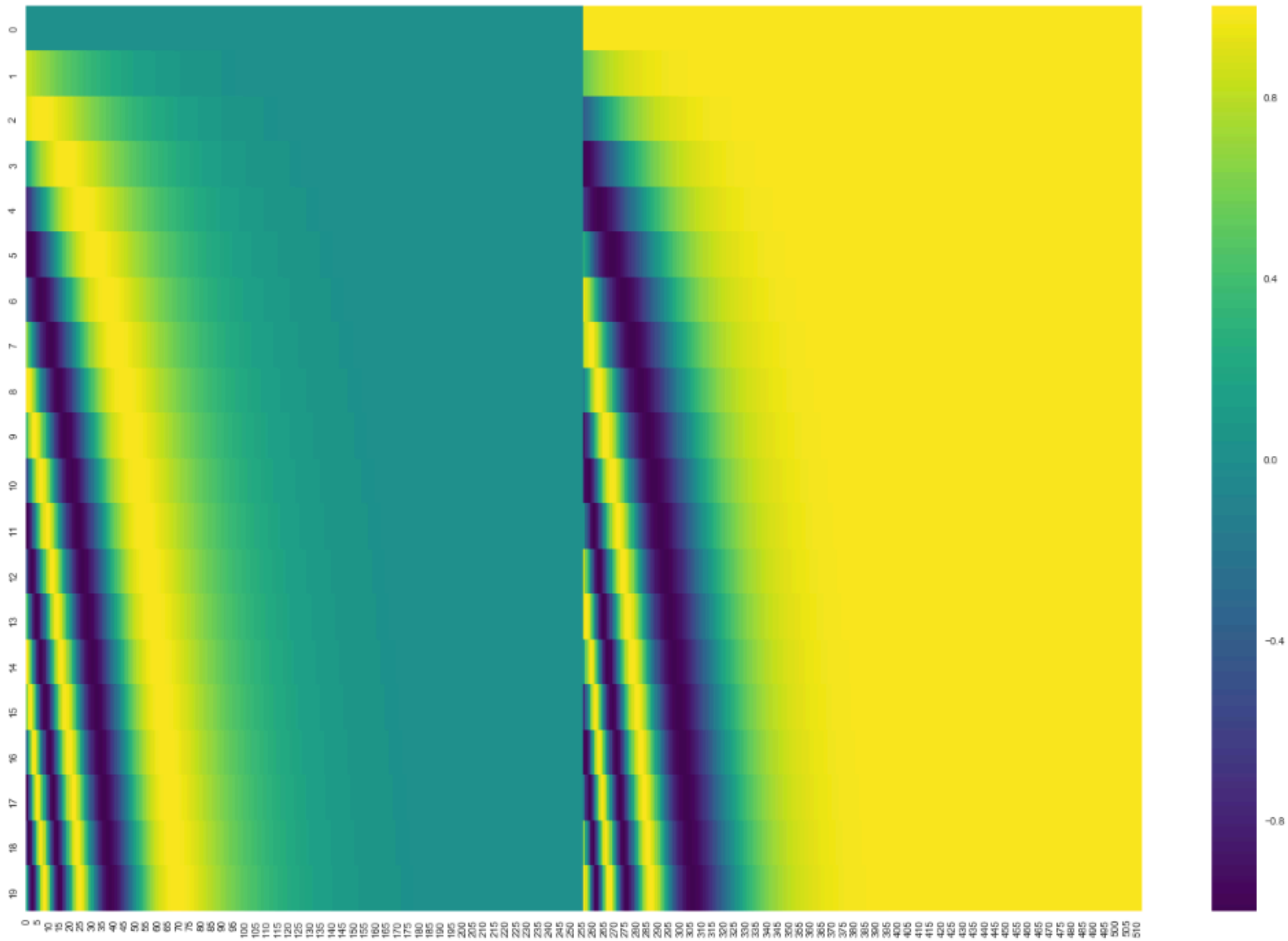
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, PE_{pos+k} can be represented as a linear function of PE_{pos}.

We also experimented with using learned positional embeddings [8] instead, and found that the two versions produced nearly identical results (see Table 3 row (E)). We chose the sinusoidal version because it may allow the model to extrapolate to sequence lengths longer than the ones encountered during training.

Vaswani et al. (2017): *Attention Is All You Need*

Sinusoidal Positional Encodings



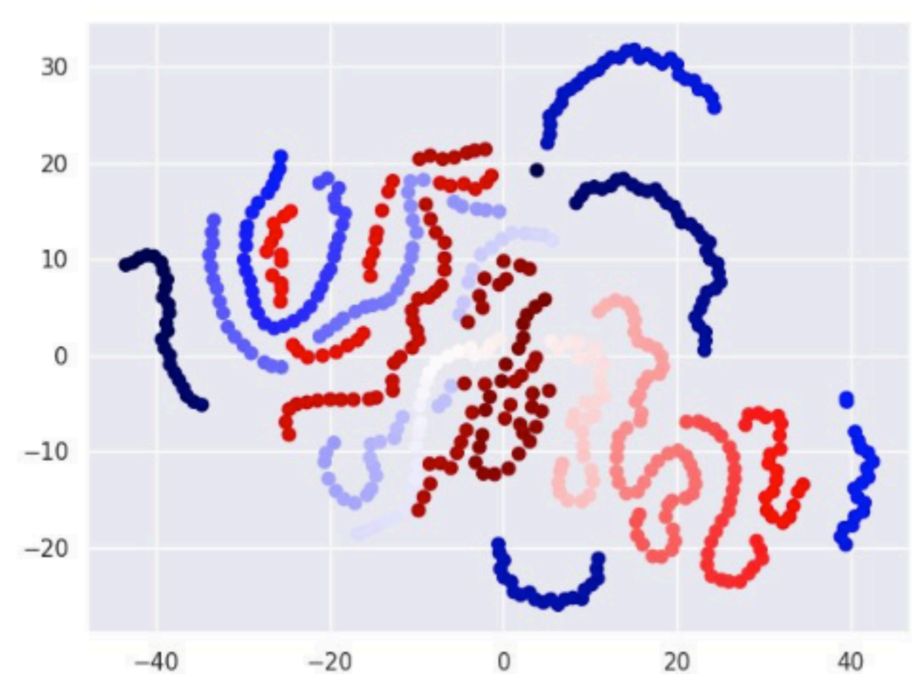
A real example of positional encoding for 20 words (rows) with an embedding size of 512 (columns). You can see that it appears split in half down the center. That's because the values of the left half are generated by one function (which uses sine), and the right half is generated by another function (which uses cosine). They're then concatenated to form each of the positional encoding vectors.

figure credit: Jay Alammar

Jack Hessel
@jmhessel

Follow

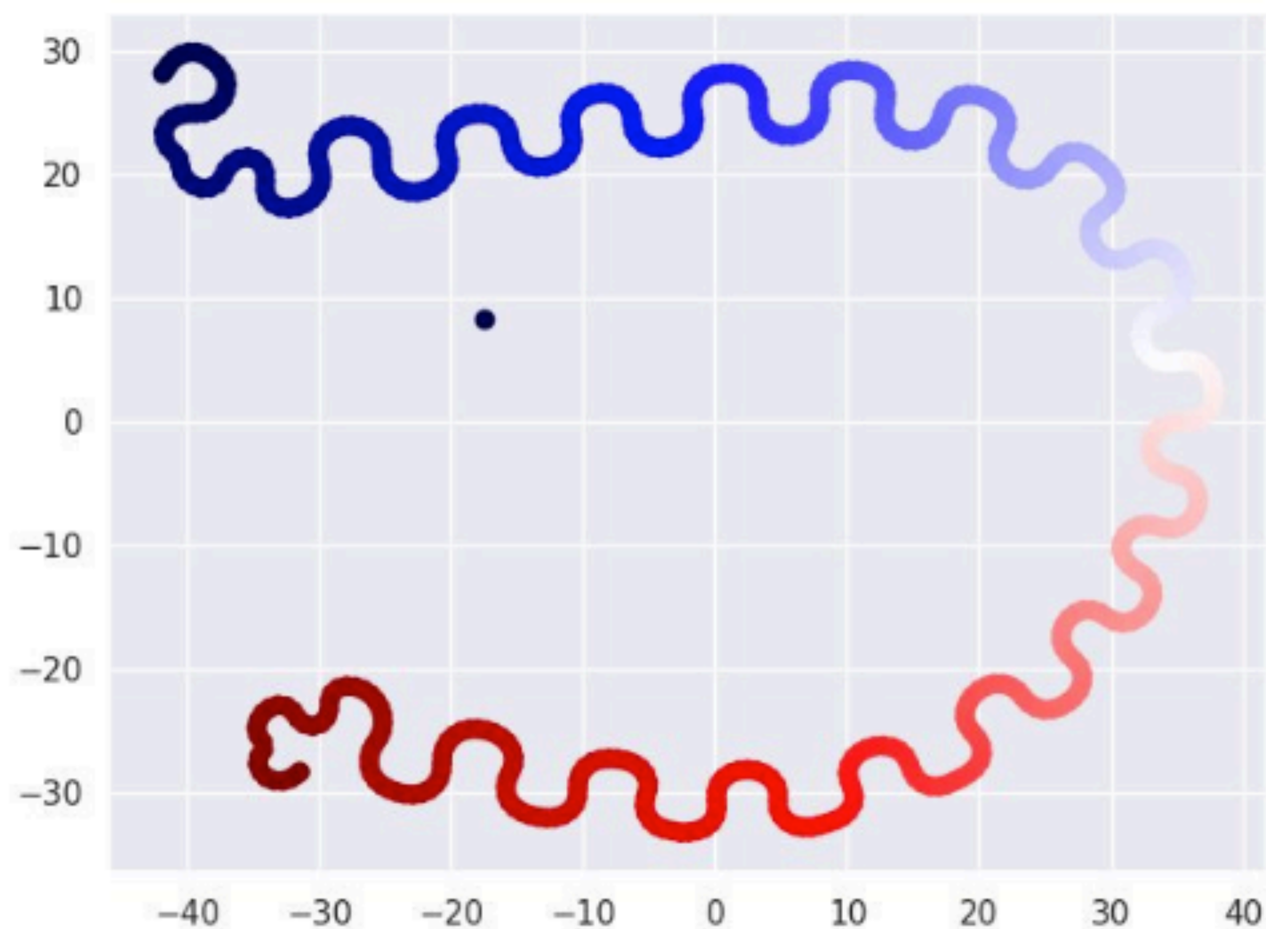
Transformer models, like BERT released by @GoogleAI today, contain an embedding for each sequence position to encode ordering information. But what the heck is a "position 3" embedding? I have no idea myself, but I TSNEed the learned embeddings (blue -> red is position 0 -> 512).



4:08 PM - 31 Oct 2018

Jack Hessel @jmhessel · 31 Oct 2018

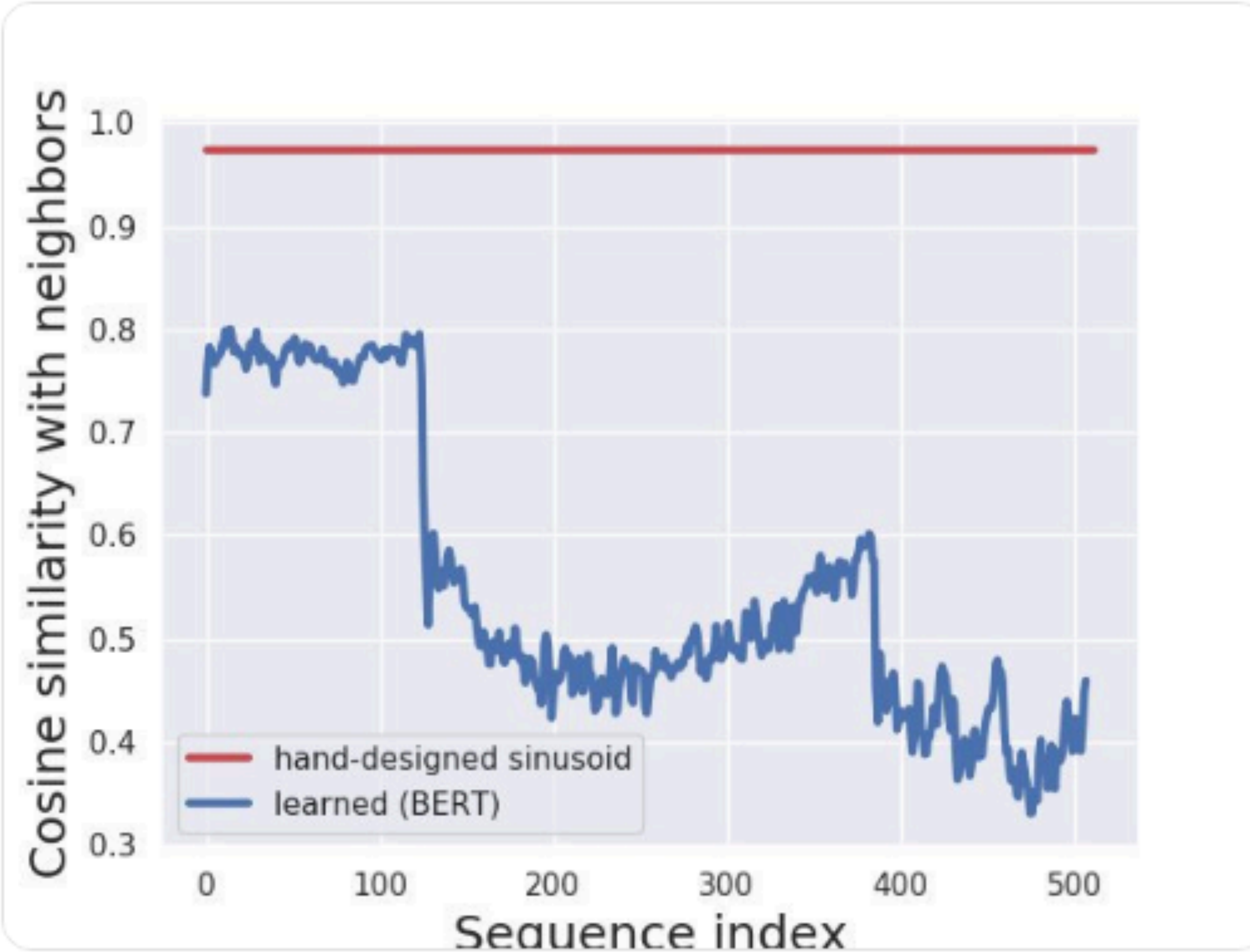
Compare this to a TSNE for the hand-crafted sinusoid pattern from the original transformer paper (again -- (blue -> red is position 0 -> 512)).



1 3 9

Jack Hessel @jmhessel · 31 Oct 2018

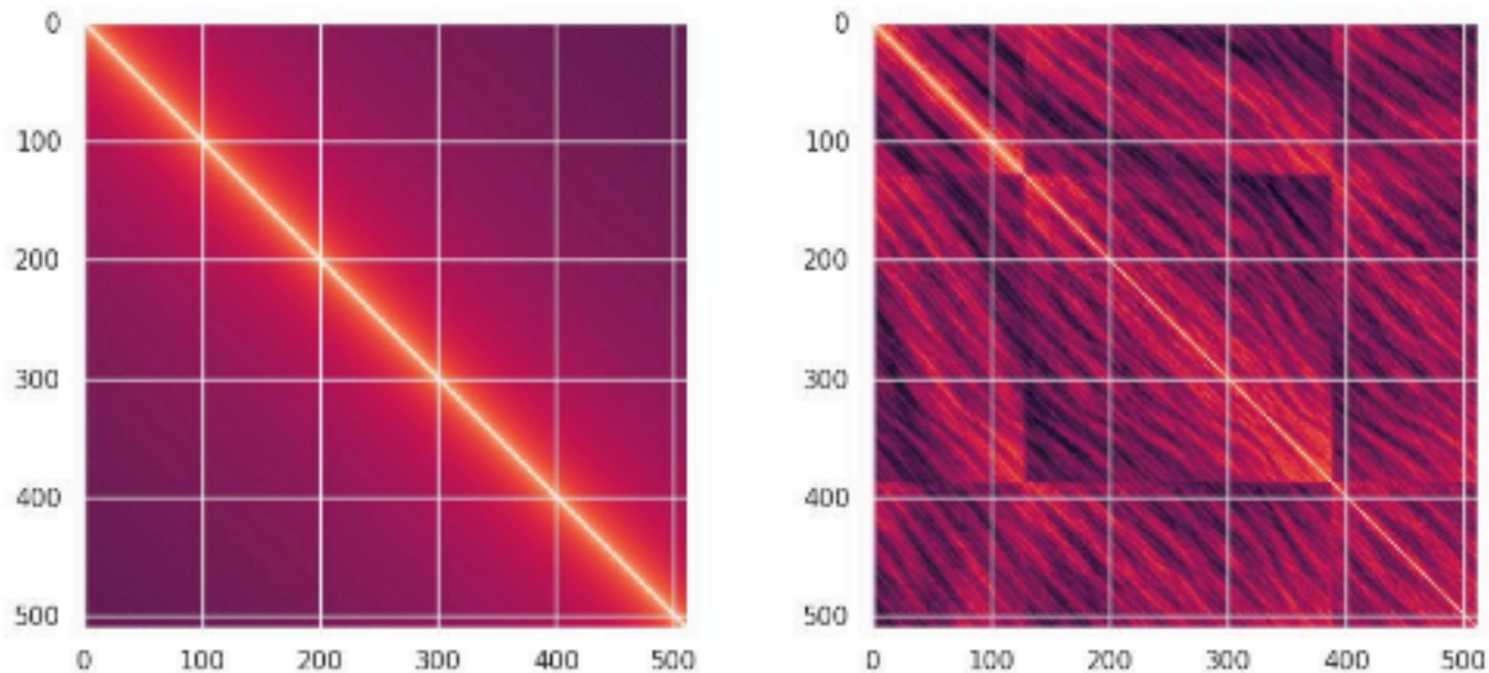
A better way of viewing these embeddings is via cosine similarity. We would expect that nearby positions are more similar to their neighbors. For the hand-crafted embeddings, neighbor sim is constant at .97. For learned, the story is way different (!)



1 10

Jack Hessel @jmhessel · 31 Oct 2018

Last tweet on this topic! Another way of exploring the similarity between position embeddings is to simply plot a heatmap of all pairwise cosine similarities. Here's what comes of that for the sinusoid embeddings and the learned embeddings. So many weird things in the learned emb



2 2 21

Attention Visualizations

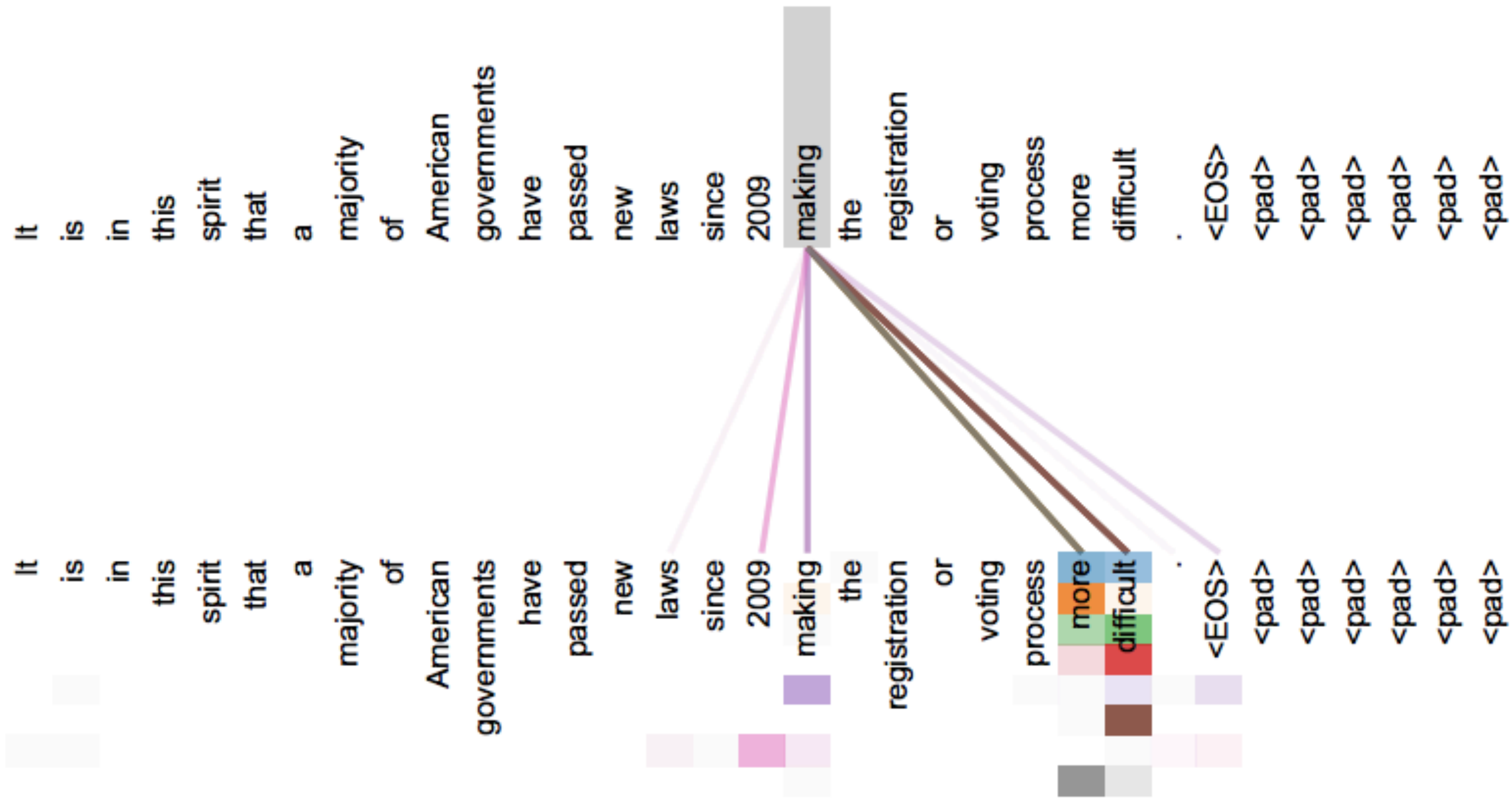


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.