

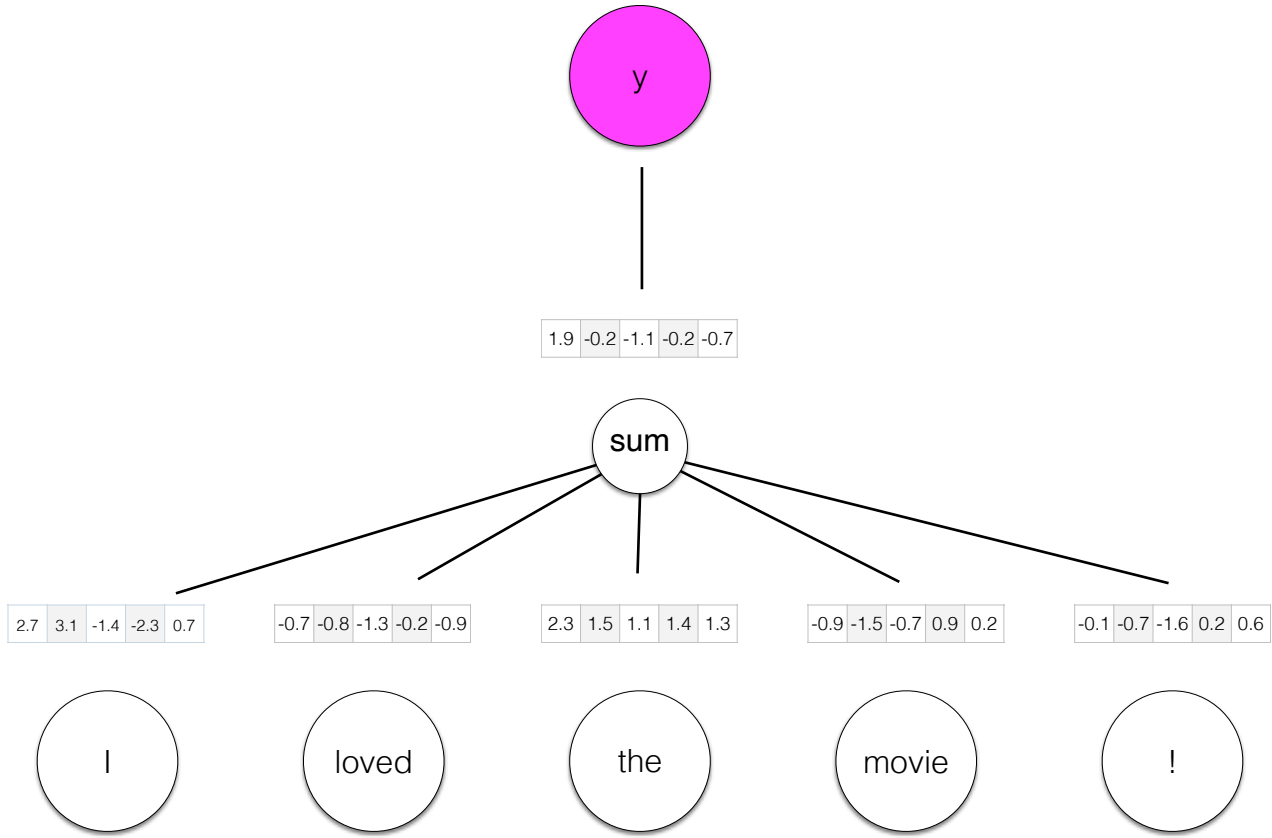
Applied Natural Language Processing

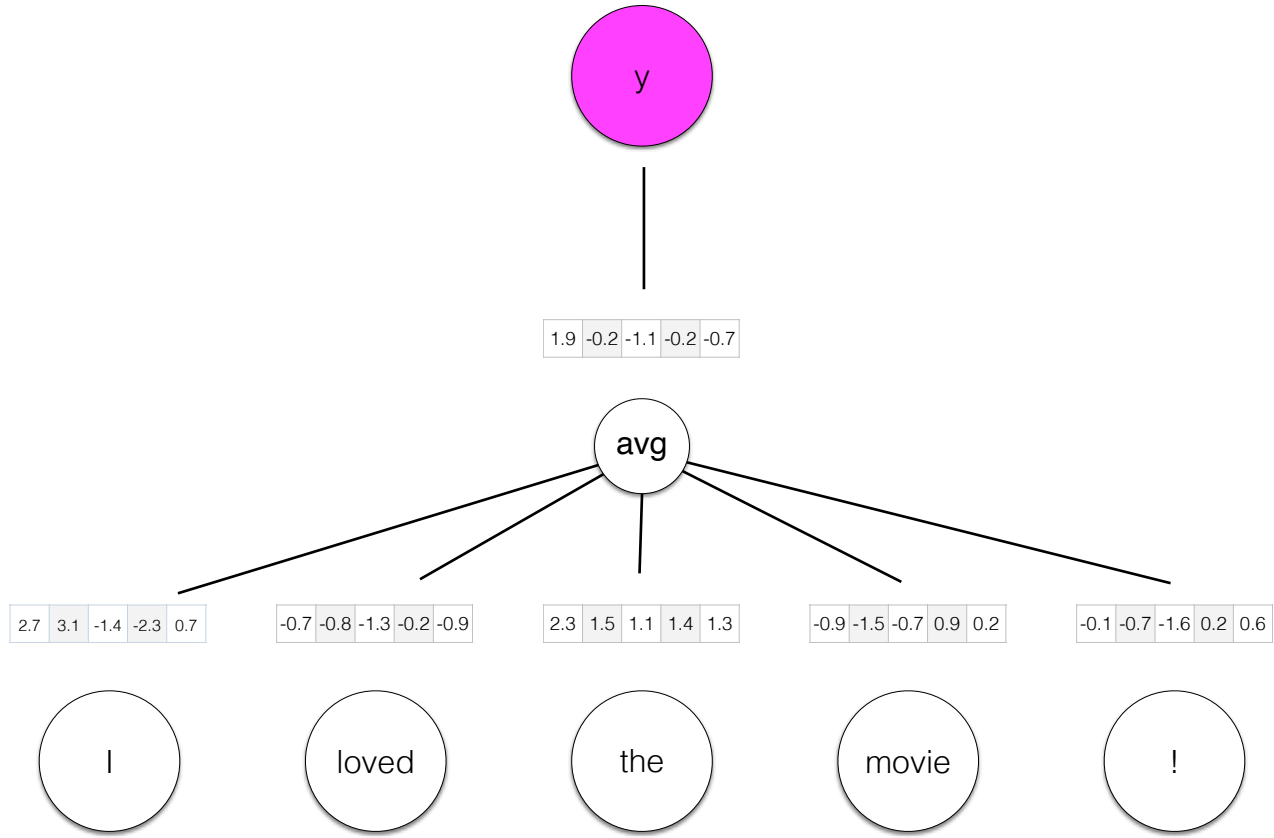
Info 256

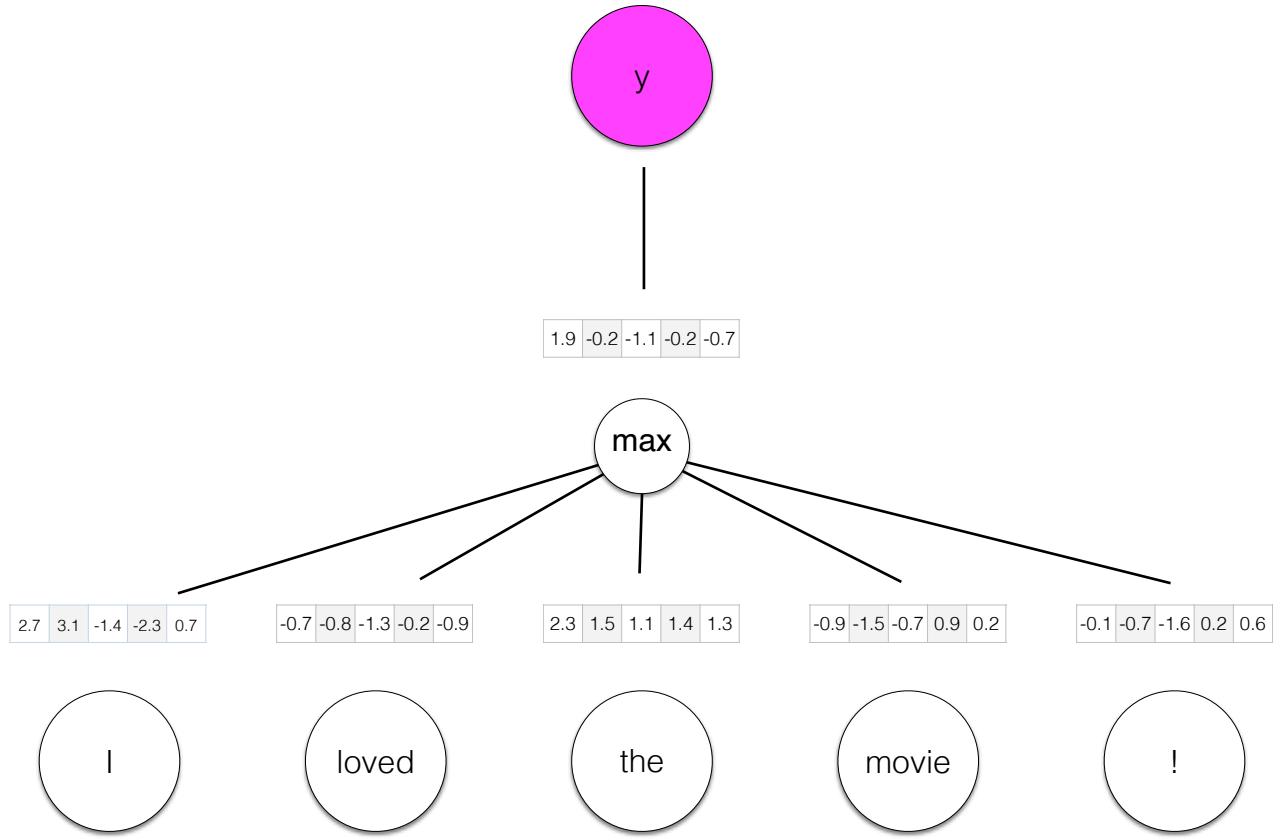
Lecture 5: Attention and transformers (Jan 31, 2023)

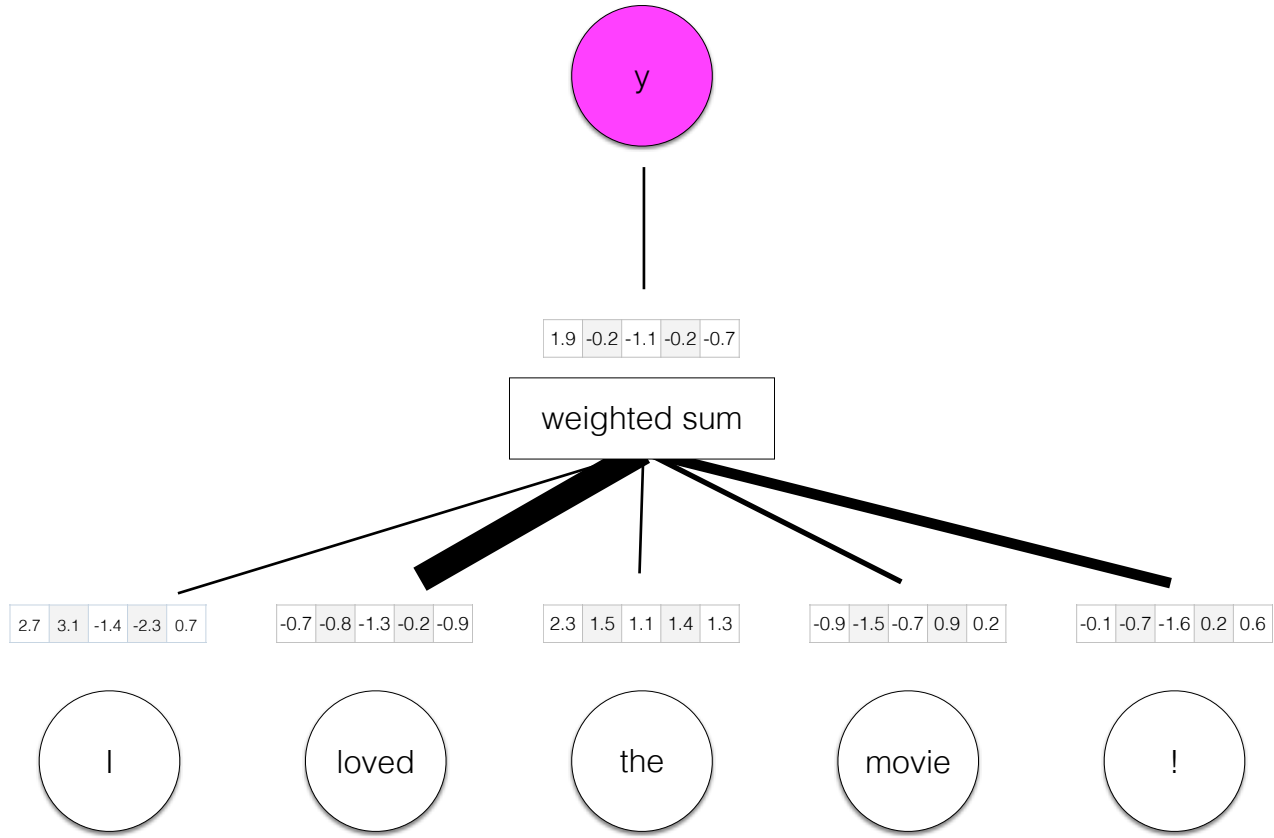
David Bamman, UC Berkeley

How do we use word embeddings for
document classification?









Attention

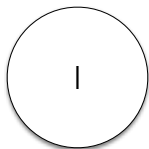
- Let's incorporate structure (and parameters) into a network that captures which elements in the input we should be **attending** to (and which we can ignore).

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

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

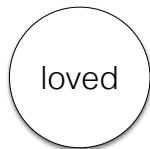
Define v to be a vector to be learned; think of it as an “important word” vector. The dot product here measures how similar each input vector is to that “important word” vector

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



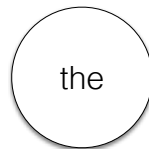
x_1

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



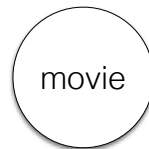
x_2

2.3	1.5	1.1	1.4	1.3
-----	-----	-----	-----	-----



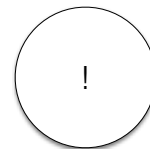
x_3

-0.9	-1.5	-0.7	0.9	0.2
------	------	------	-----	-----



x_4

-0.1	-0.7	-1.6	0.2	0.6
------	------	------	-----	-----



x_5

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

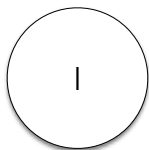
2.7	3.1	-1.4	-2.3	0.7
-----	-----	------	------	-----

-3.4

$$r_1 = v^\top x_1$$

|

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



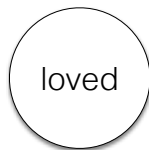
x_1

2.4

$$r_2 = v^\top x_2$$

|

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



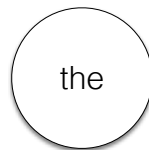
x_2

-0.8

$$r_3 = v^\top x_3$$

|

2.3	1.5	1.1	1.4	1.3
-----	-----	-----	-----	-----



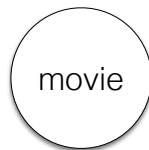
x_3

-1.2

$$r_4 = v^\top x_4$$

|

-0.9	-1.5	-0.7	0.9	0.2
------	------	------	-----	-----



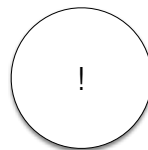
x_4

1.7

$$r_5 = v^\top x_5$$

|

-0.1	-0.7	-1.6	0.2	0.6
------	------	------	-----	-----



x_5

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

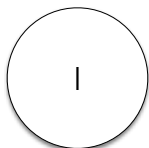
$$a = \text{softmax}(r)$$

a	0	0.64	0.02	0.02	0.32
r	-3.4	2.4	-0.8	-1.2	1.7

$$r_1 = v^\top x_1$$

|

2.7 3.1 -1.4 -2.3 0.7

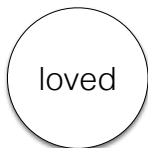


x_1

$$r_2 = v^\top x_2$$

|

-0.7 -0.8 -1.3 -0.2 -0.9

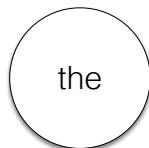


x_2

$$r_3 = v^\top x_3$$

|

2.3 1.5 1.1 1.4 1.3

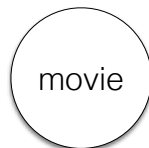


x_3

$$r_4 = v^\top x_4$$

|

-0.9 -1.5 -0.7 0.9 0.2

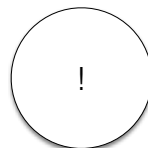


x_4

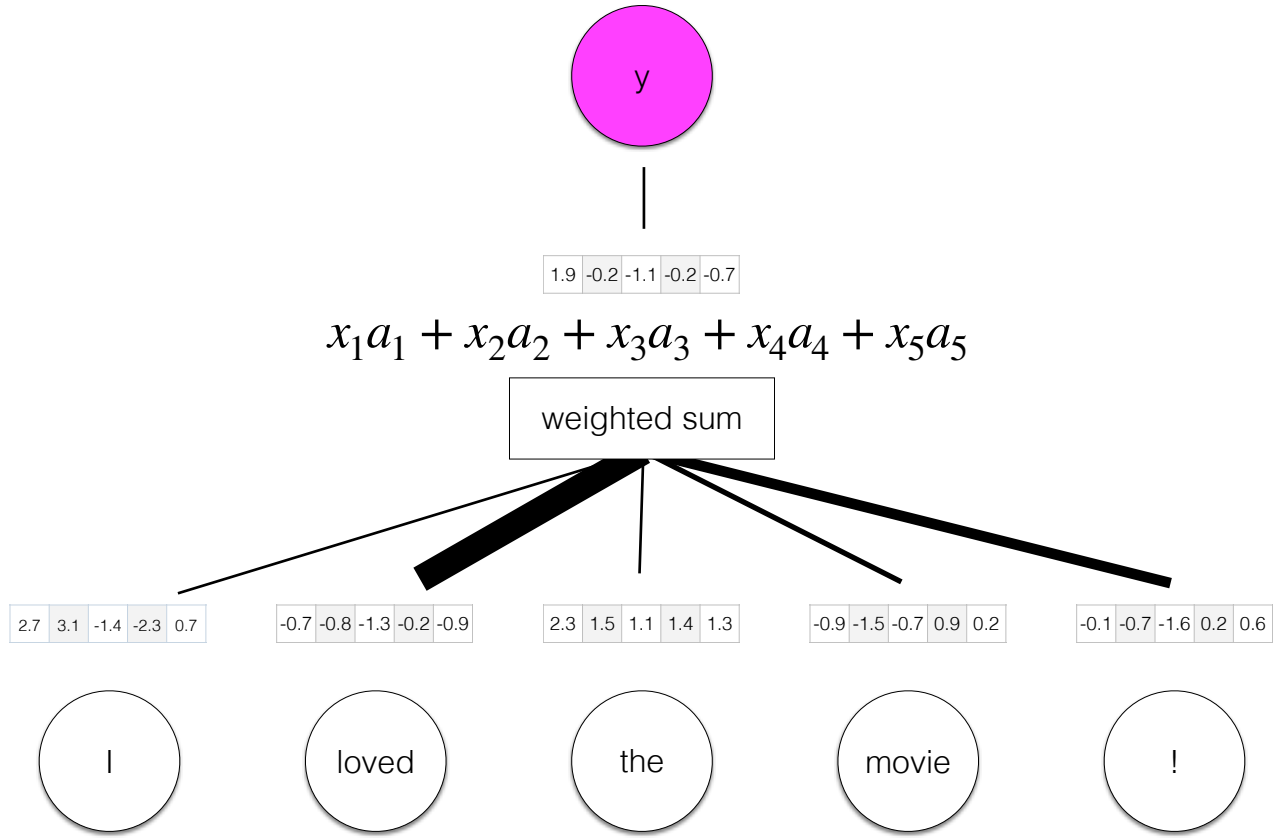
$$r_5 = v^\top x_5$$

|

-0.1 -0.7 -1.6 0.2 0.6



x_5



Attention

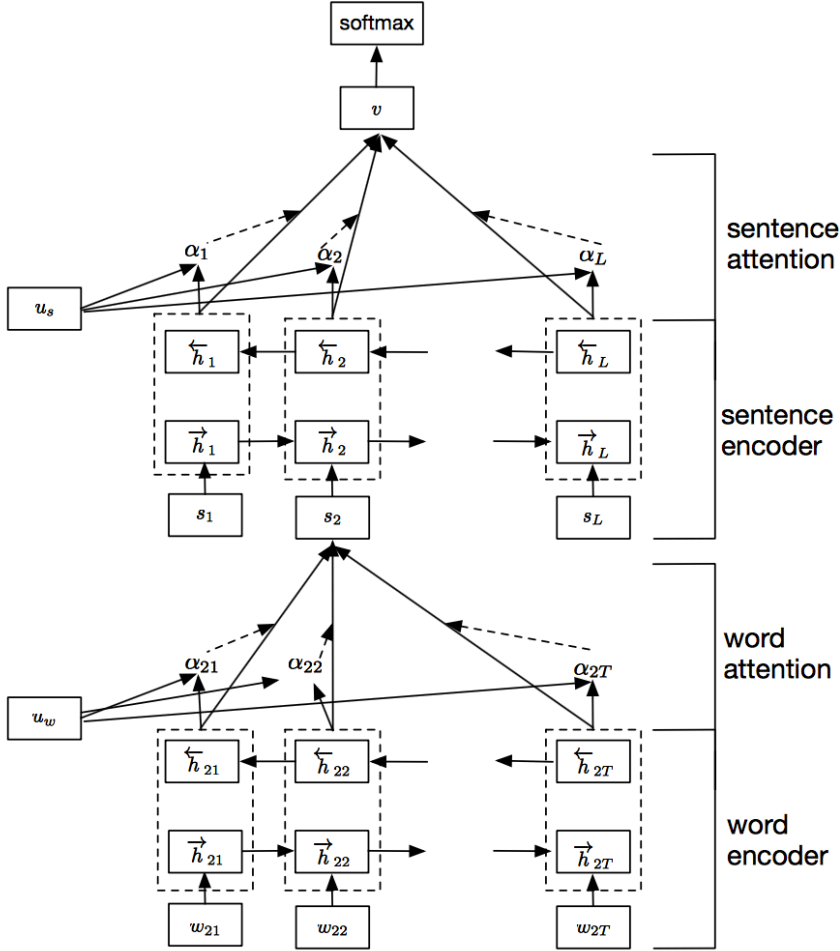
- Lots of variations on attention:
 - Linear transformation of x into before dotting with v
 - Non-linearities after each operation.
 - “Multi-head attention”: multiple v vectors to capture different phenomena that can be attended to in the input.
 - Hierarchical attention (sentence representation with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

attention over sentences

bidirectional GRU over sentence representations

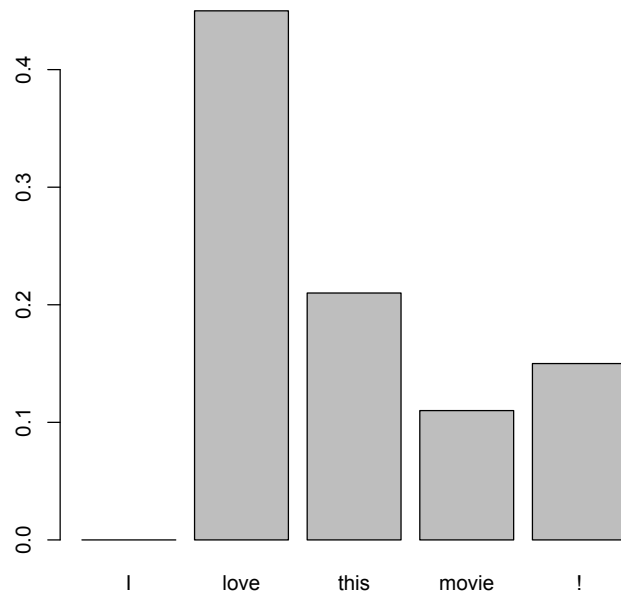
attention over words

bidirectional GRU over word representations



Attention

- Attention gives us a normalized weight for every token in a sequence that tells us how important that word was for the prediction
- This can be useful for visualization



Attention

after 15 minutes watching the movie i was asking myself what to do leave the theater sleep or try to keep watching the movie to see if there was anything worth i finally watched the movie what a waste of time maybe i am not a 5 years old kid anymore

original α

$$f(x|\alpha, \theta) = 0.01$$

after 15 minutes watching the movie i was asking myself what to do leave the theater sleep or try to keep watching the movie to see if there was anything worth i finally watched the movie what a waste of time maybe i am not a 5 years old kid anymore

adversarial $\tilde{\alpha}$

$$f(x|\tilde{\alpha}, \theta) = 0.01$$

Base model	brilliant	and	moving	performances	by	tom	and	peter	finch
Jain and Wallace (2019)	brilliant	and	moving	performances	by	tom	and	peter	finch
Our adversary	brilliant	and	moving	performances	by	tom	and	peter	finch

Figure 2: Attention maps for an IMDb instance (all predicted as positive with score > 0.998), showing that in practice it is difficult to learn a distant adversary which is consistent on all instances in the training set.

[Submitted on 26 Feb 2019 (v1), last revised 8 May 2019 (this version, v3)]

Attention is not Explanation

Sarthak Jain, Byron C. Wallace

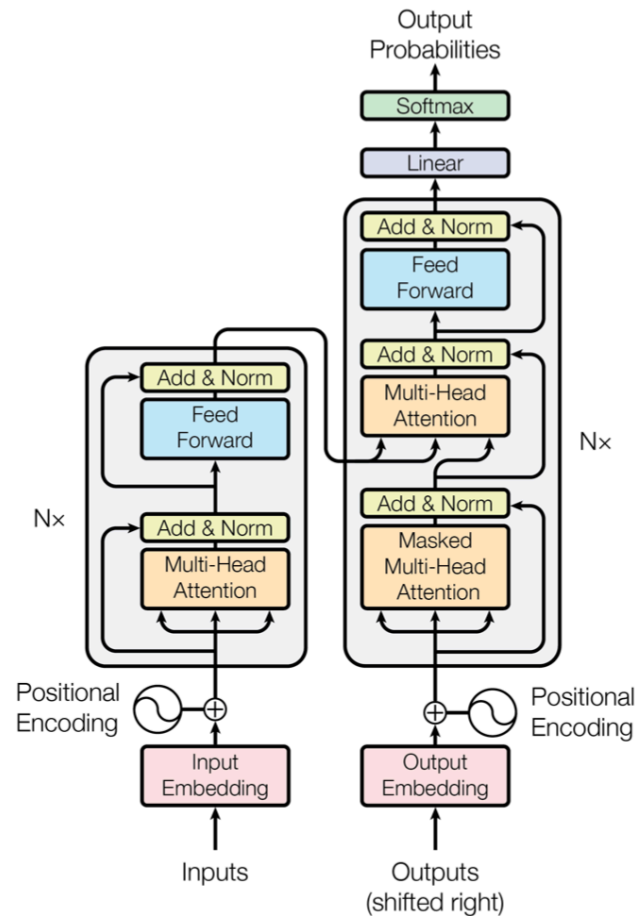
[Submitted on 13 Aug 2019 (v1), last revised 5 Sep 2019 (this version, v2)]

Attention is not not Explanation

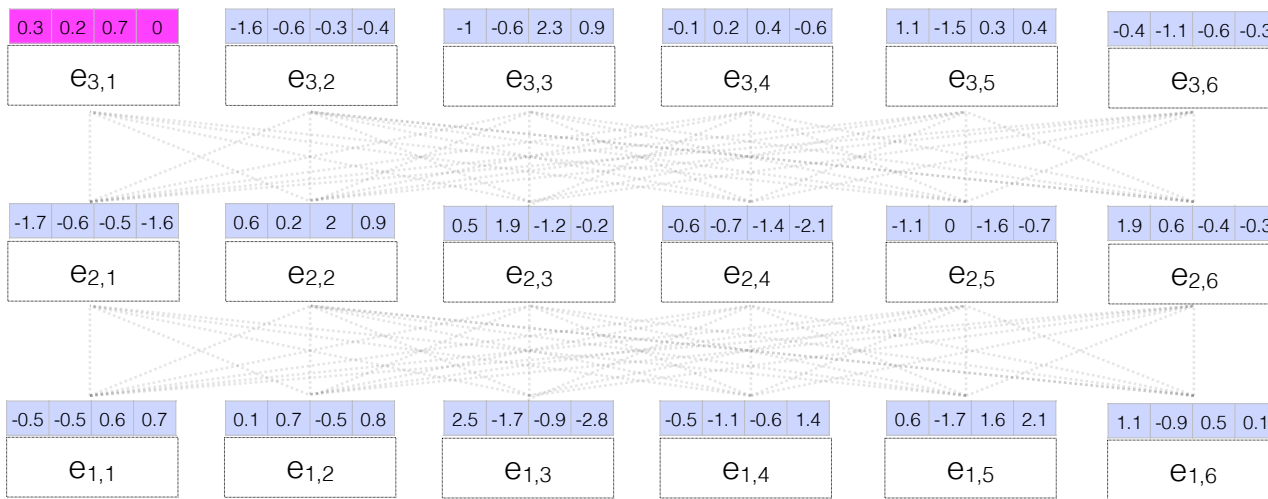
Sarah Wiegrefe, Yuval Pinter

Transformers

- Vaswani et al. 2017, “Attention is All You Need”
- Transforms map an input **sequence** of vectors to an output **sequence** of vectors of the same dimensionality



positive
sentiment



[CLS]

The

movie

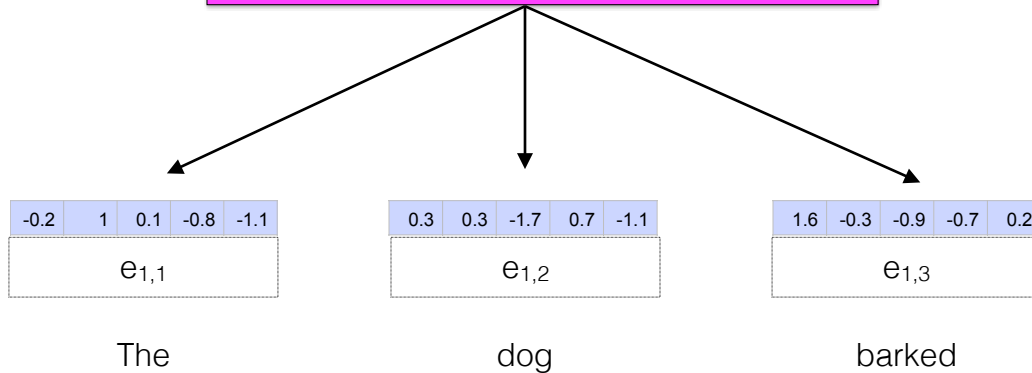
was

amazing

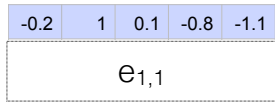
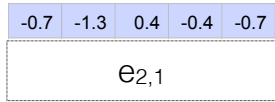
[SEP]

Self-Attention

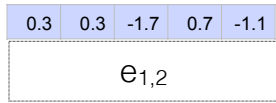
Let's assume (for the moment) that our input vectors are static word2vec embeddings of words.



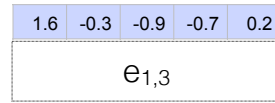
The value for time step j at layer i is the result of attention over all time steps in the previous layer $i-1$



The



dog



barked

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

query

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

key

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

value

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

original value

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

$e_{1,1}$

The

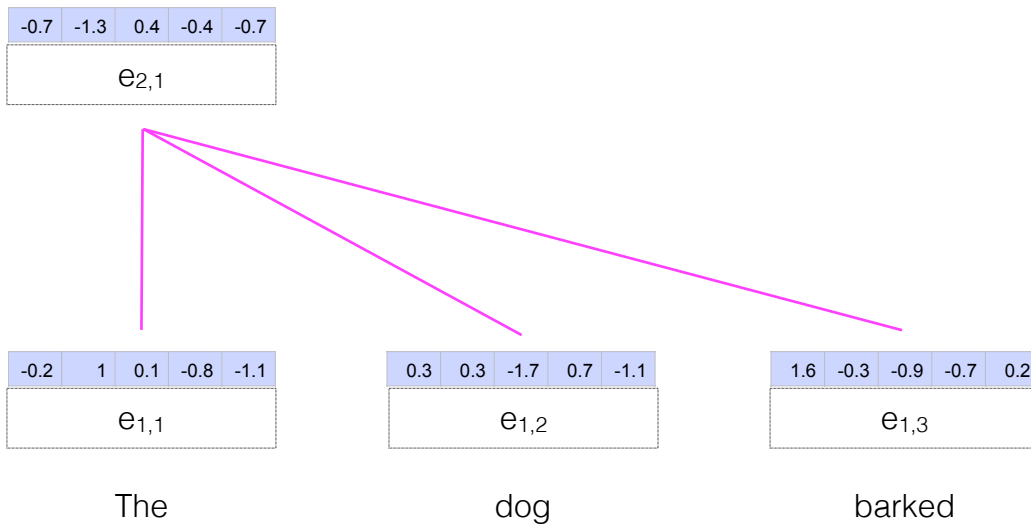
$$W^Q \in \mathbb{R}^{100 \times 37}$$

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

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

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

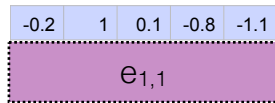
Self attention **from** “The” at position 1 to every token in the sentence



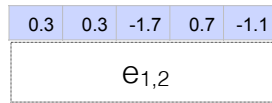
- The compatibility score between two words is the dot product between their respective **query** and **key** vectors.

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

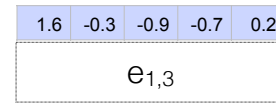
<i>a</i>	0.07	0.58	0.35	$a = \text{softmax}(\text{scores})$
<i>scores</i>	-1.4	0.64	0.14	
	$q_1 \cdot k_1$	$q_1 \cdot k_2$	$q_1 \cdot k_3$	



The



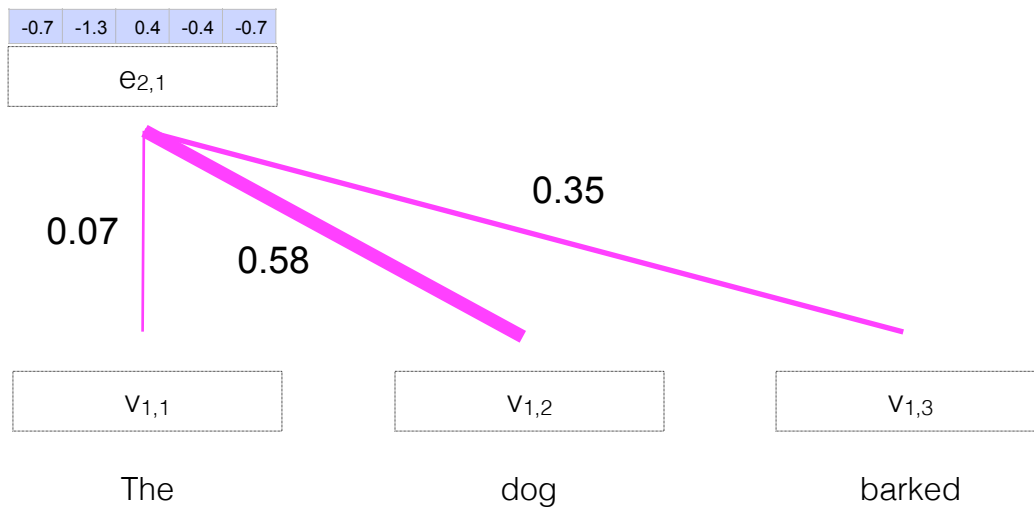
dog



barked

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

If the dimensionality of v is 100, how big is this vector?



Multihead attention

- Attention in transformers is essentially a set of learned parameters (W^Q , W^K , W^V) and a mathematical expression for how an input is transformed into an output through operations involving those parameters.

$$Q = XW^Q; K = XW^K; V = XW^V$$

$$\text{SelfAttention}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

Scaled by the dimensionality of
the key vectors ($\sqrt{d_k}$)

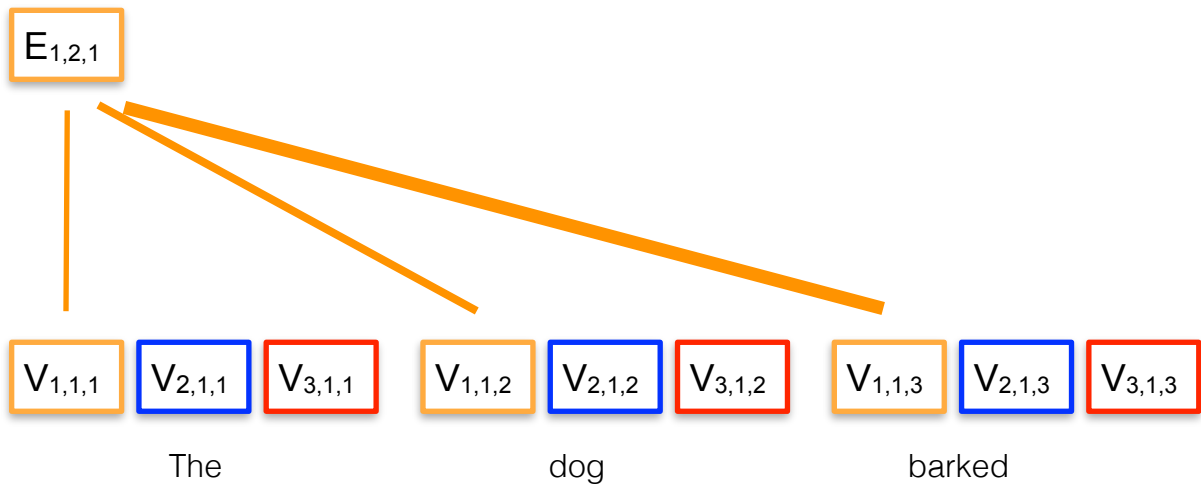
Multihead attention

- Attention provides one view on the data; just like we use multiple filters in CNNs to provide multiple perspectives (by learning separate parameters for each one), so too can we learn multiple perspectives in a transformer by learning multiple (W^Q , W^K , W^V) sets.

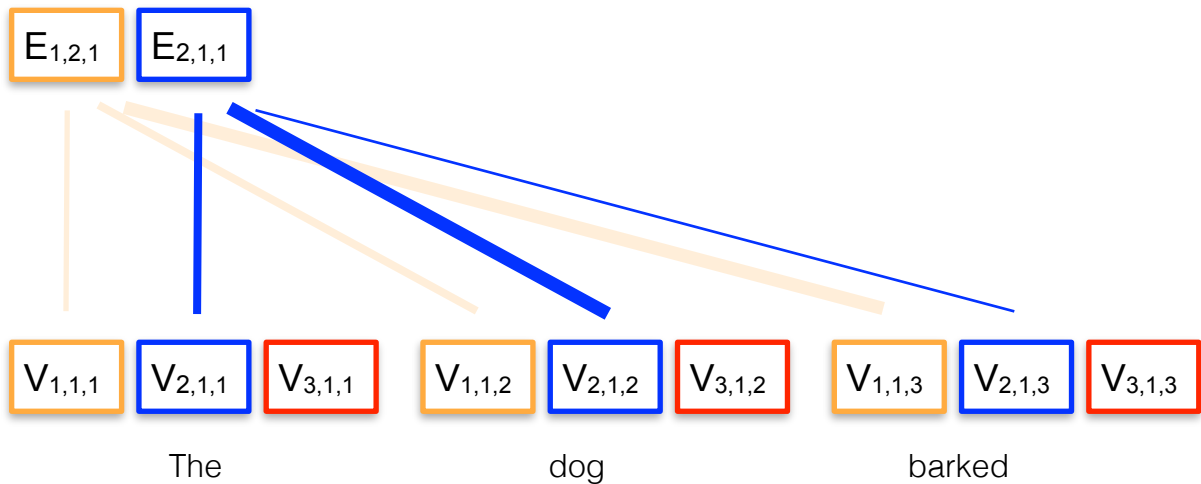
$$Q_i = XW_i^Q; K = XW_i^K; V = XW_i^V$$

$$\text{SelfAttention}(Q_i, K_i, V_i) = \text{softmax} \left(\frac{Q_i K_i^T}{\sqrt{d_k}} \right) V_i$$

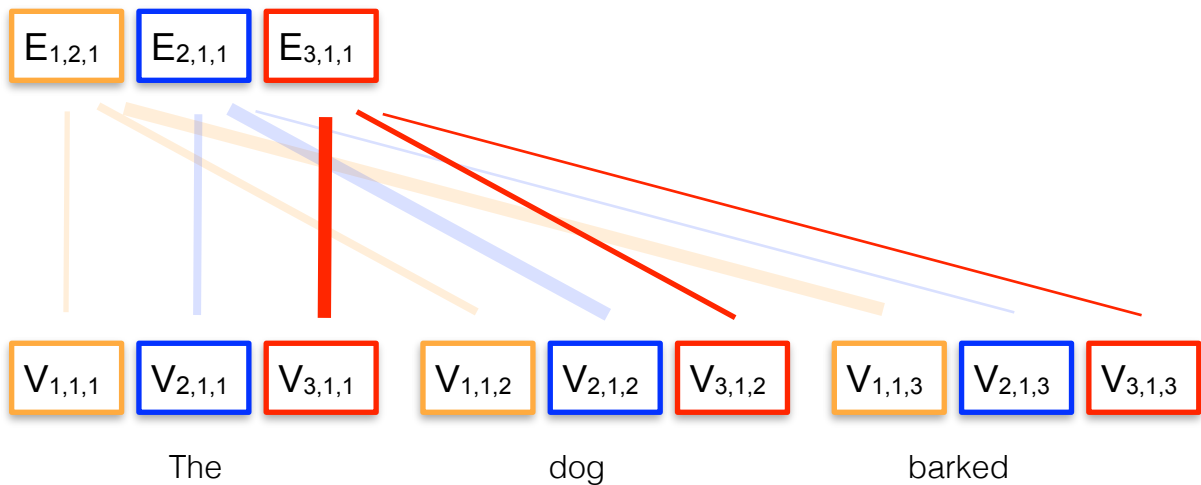
- With multihead attention, each attention head generates its own output vector i based on its own W_i^Q , W_i^K and W_i^V



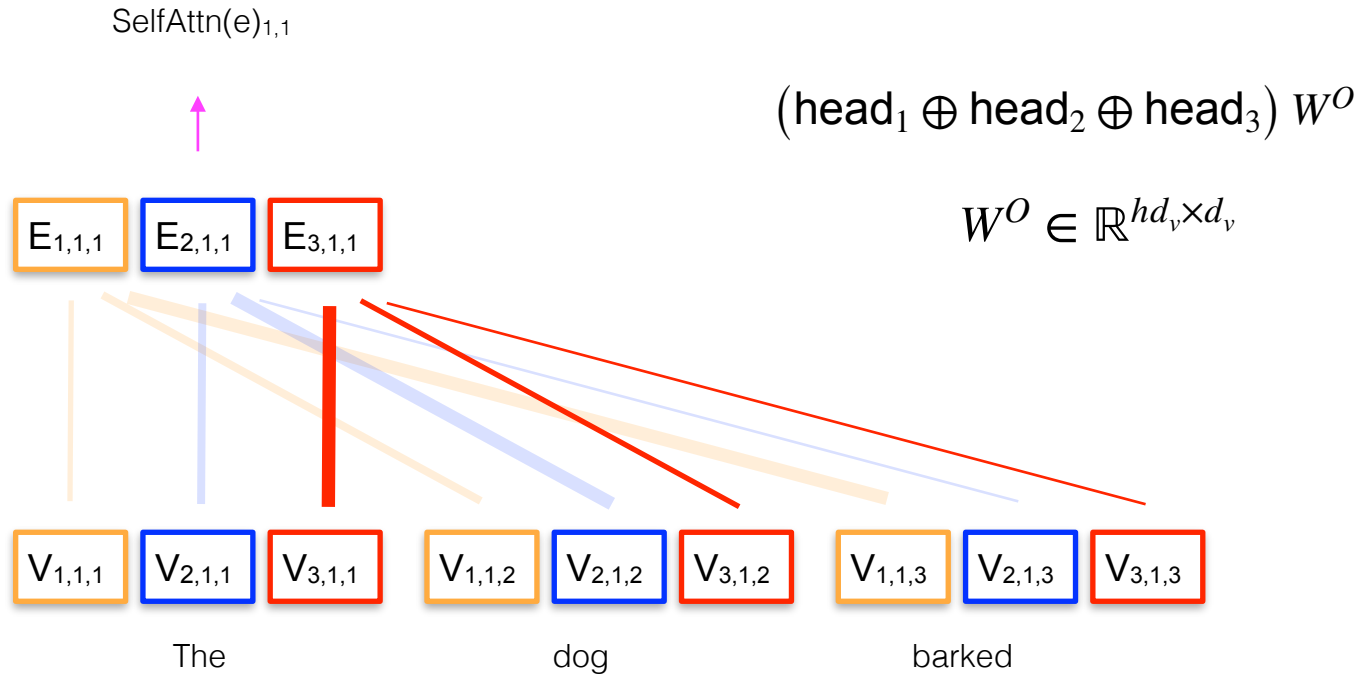
- With multihead attention, each attention head generates its own output vector i based on its own W_i^Q , W_i^K and W_i^V



- With multihead attention, each attention head generates its own output vector i based on its own W_i^Q , W_i^K and W_i^V



- These h separate output heads are then concatenated together and linearly transformed back to the original dimensionality d_v



-0.7 -1.3 0.4 -0.4 -0.7

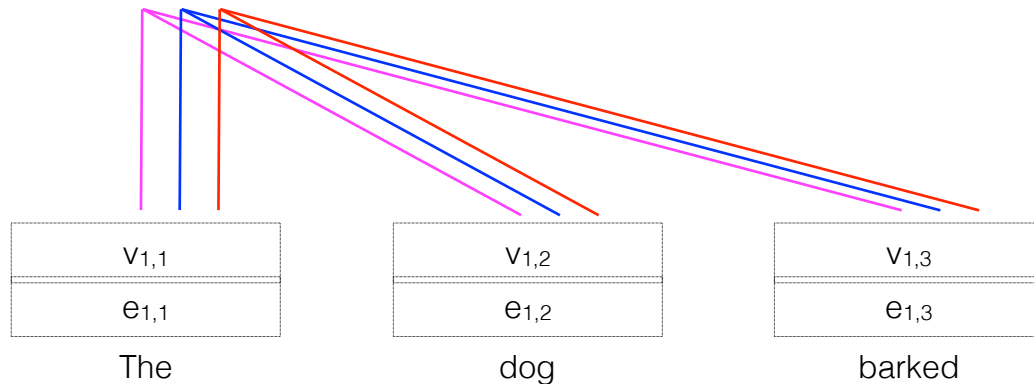
$\in \mathbb{R}^{100}$

$e_{2,1}$

$y = \text{LayerNorm}(z + \text{FFNN}(z)) \in \mathbb{R}^{100}$

$z = \text{LayerNorm}(e + \text{SelfAttn}(e)) \in \mathbb{R}^{100}$

$\text{SelfAttn}(e)_{1,1} \in \mathbb{R}^{100}$



Layer Normalization

- Transform each output from a layer d_h into its z-score, with two learnable parameters γ and β .

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i \quad \sigma = \sqrt{\frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2}$$

$$\hat{x} = \frac{x - \mu}{\sigma}$$

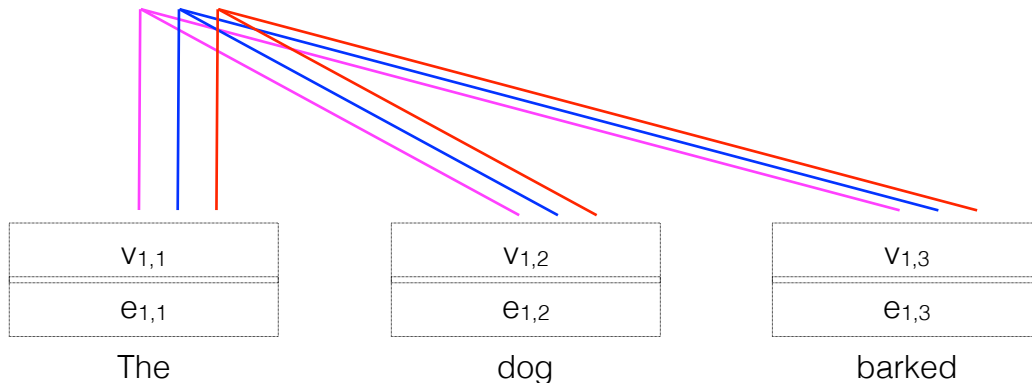
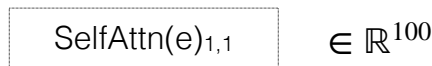
$$\text{LayerNorm} = \gamma \hat{x} + \beta$$

- Residual layers add a layer's **input** to its **output**, giving later layers access to unmediated information.



$y = \text{LayerNorm}(z + \text{FFNN}(z)) \in \mathbb{R}^{100}$

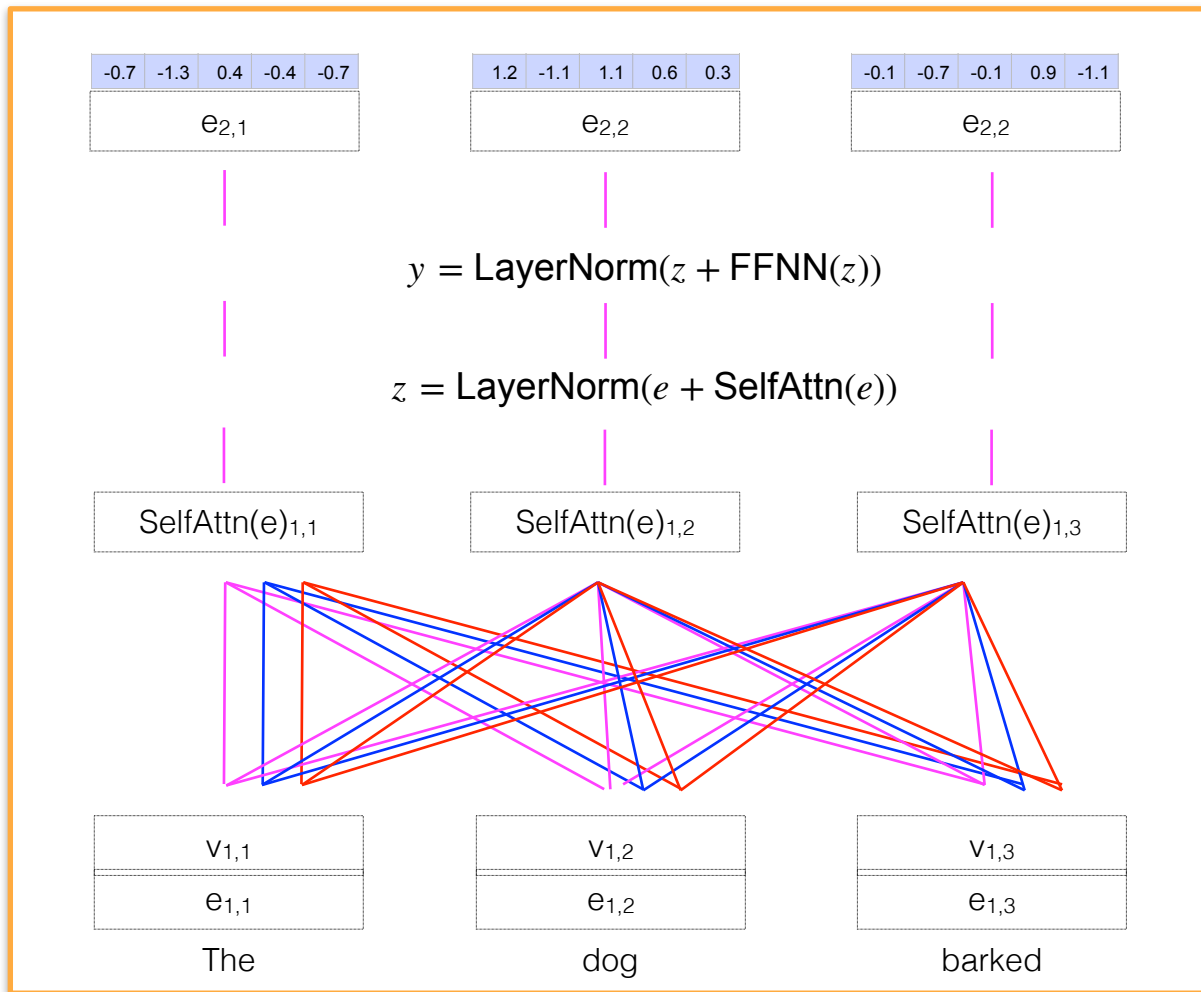
$z = \text{LayerNorm}(e + \text{SelfAttn}(e)) \in \mathbb{R}^{100}$



Output

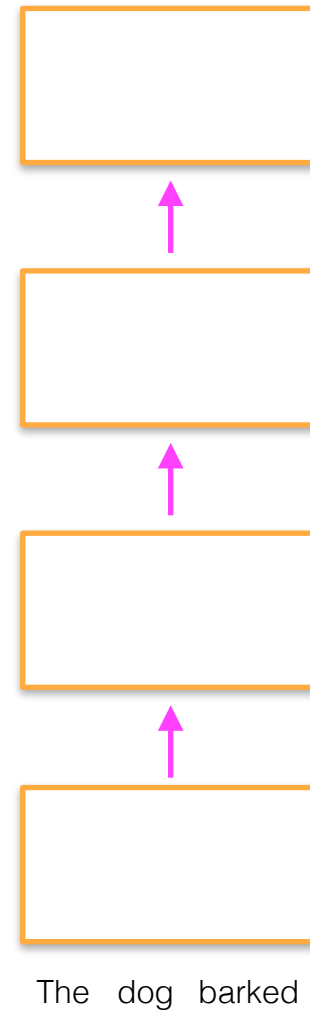
This whole process defines one attention **block**. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

Input

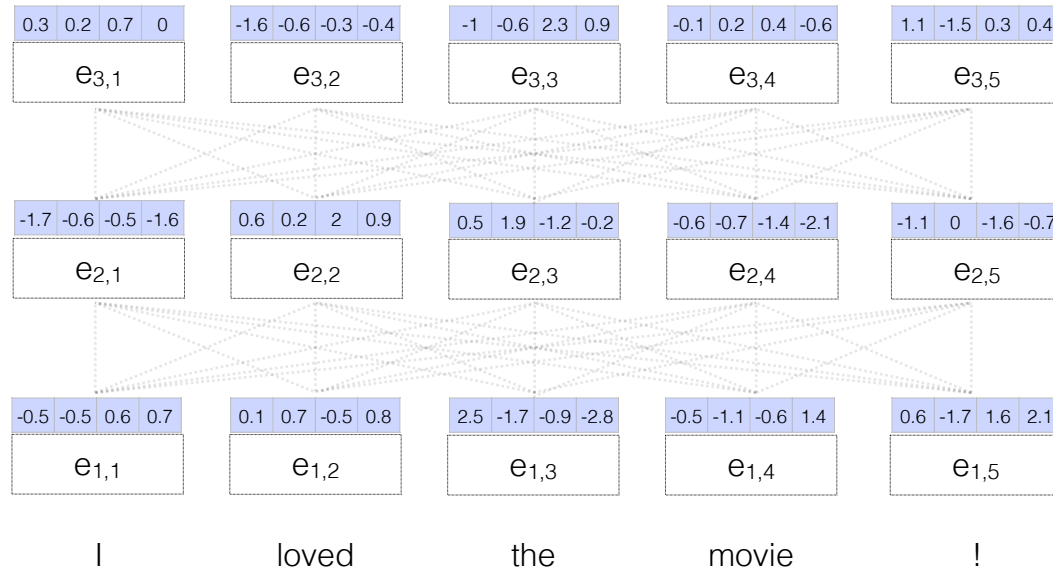


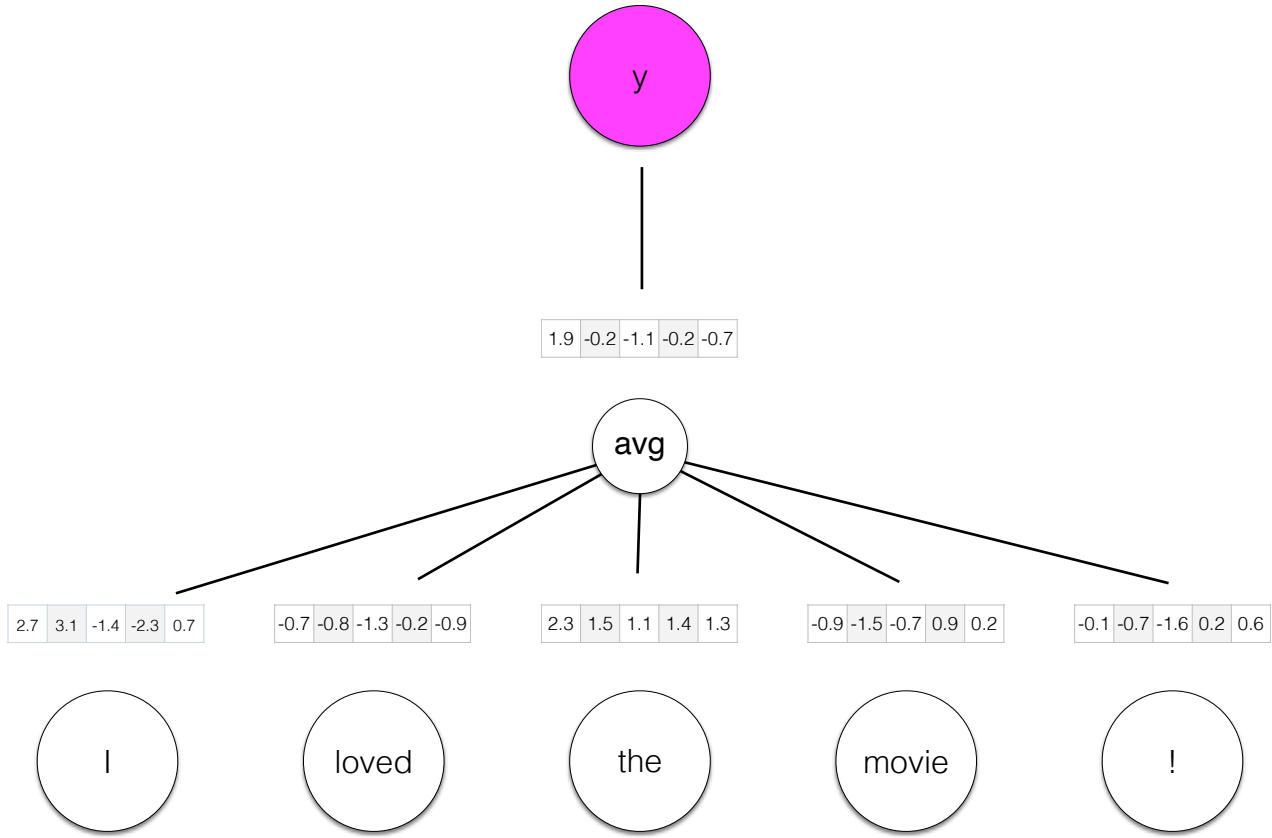
This whole process defines one attention **block**.
The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

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



- At this point, we have one representation for each token for each layer in our transformer. How do we use this for document classification?

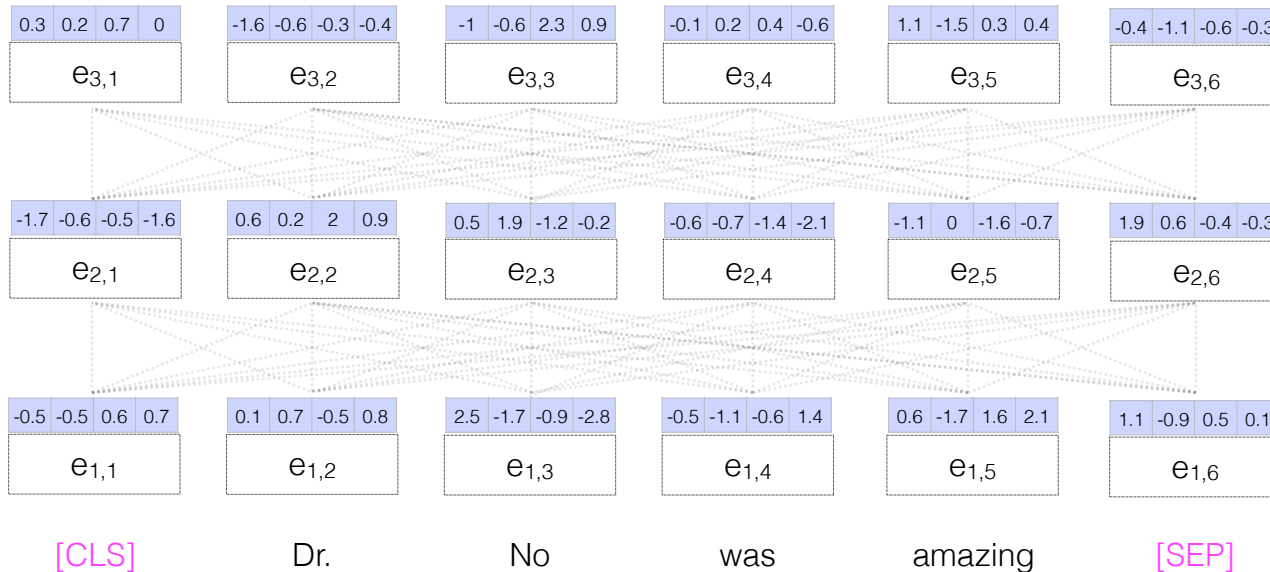




BERT

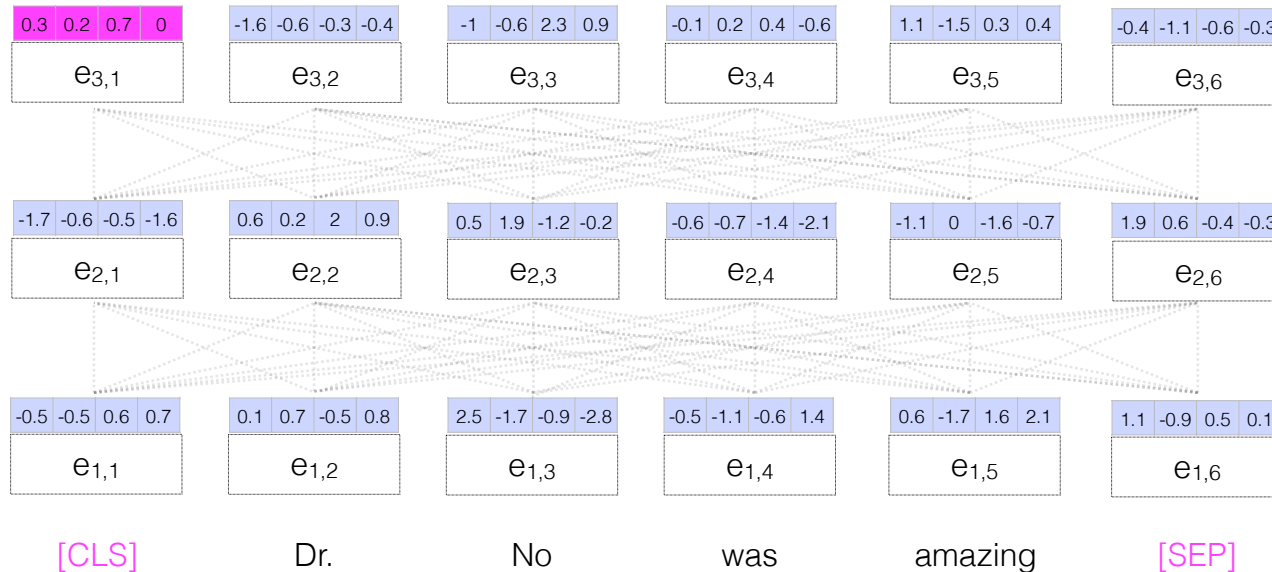
- Deep layers (12 for BERT base, 24 for BERT large)
- Large representation sizes (768 per layer)
- Pretrained on English Wikipedia (2.5B words) and BooksCorpus (800M words) — we'll cover this on [2/9](#) when we study contextual language models.

- 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)

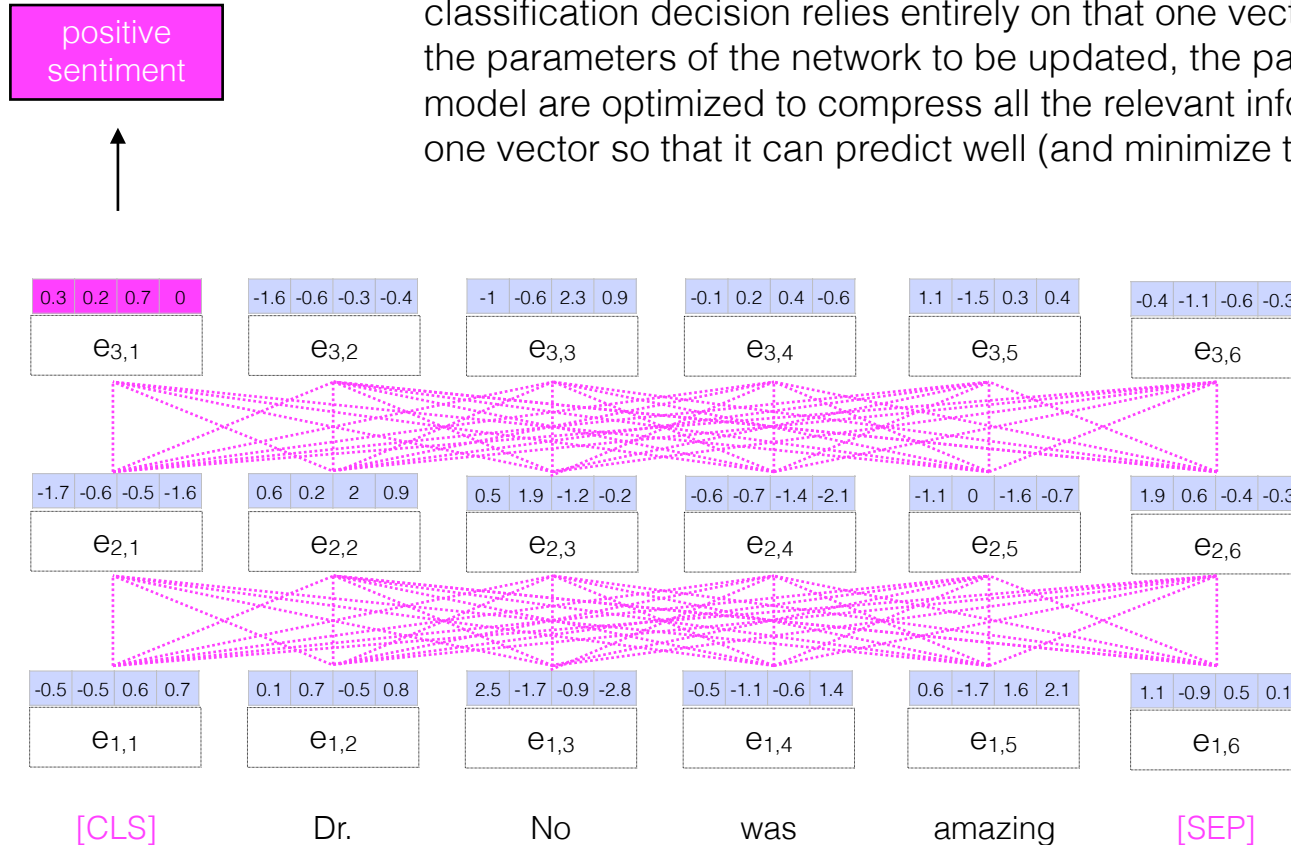


- We can represent the entire document with this *one* [CLS] vector
- Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all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

positive sentiment



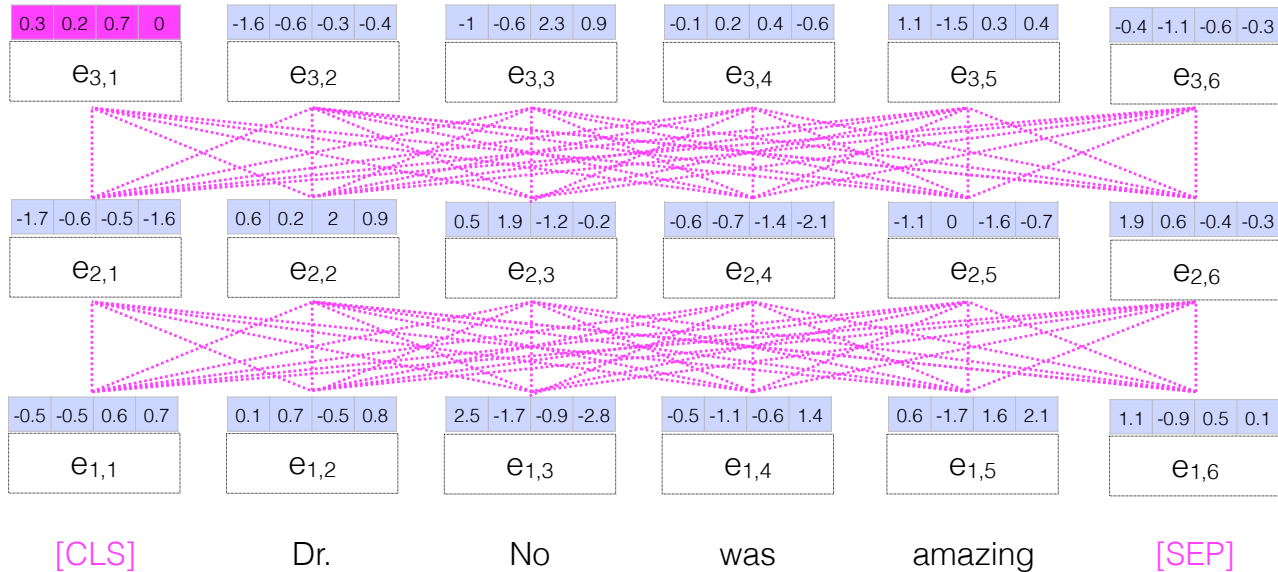
- We can represent the entire document with this *one* [CLS] vector
- Why does this work? When we design our network so that a classification decision relies entirely on that one vector and allow all the parameters of the network to be updated, the parameters of the model are optimized to compress all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



positive sentiment



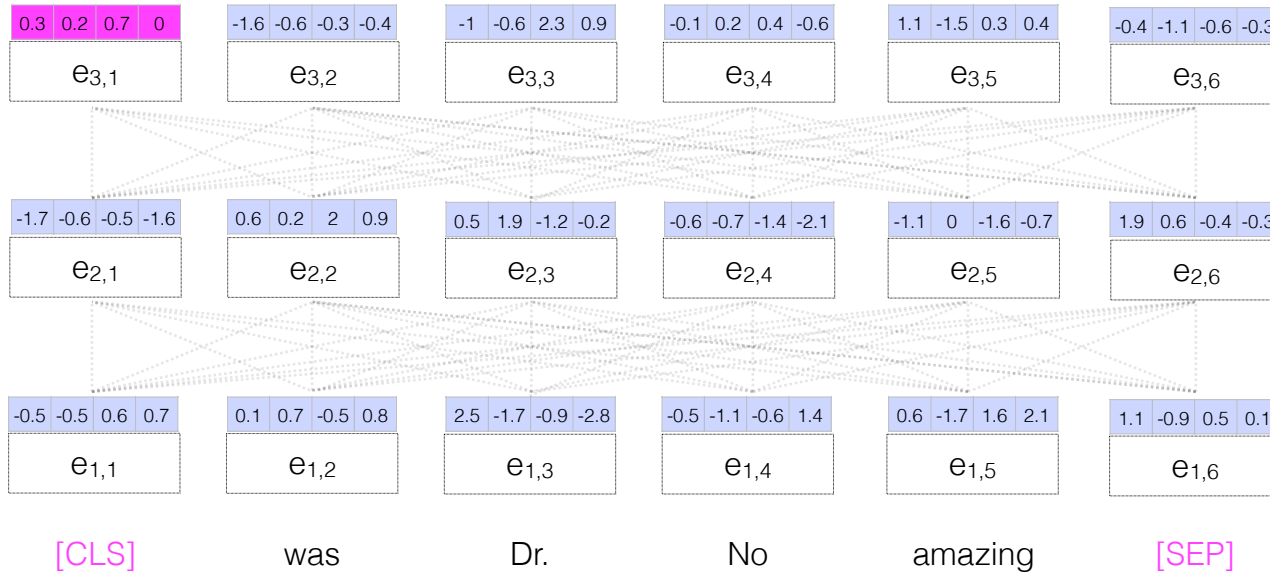
- CNNs can reason over text inputs of arbitrary length; FFNN require transformation into a fixed-dimensional feature vector.
- Can transformers reason over inputs of arbitrary length?



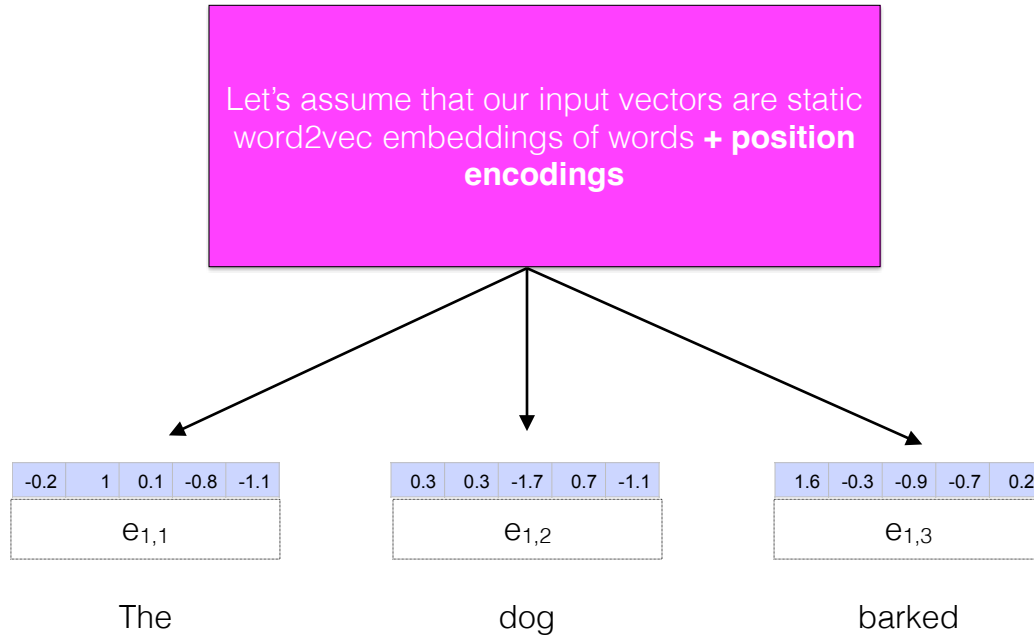
positive
sentiment



- Does a transformer encode any intrinsic information about the order of words within a sequence? Would the output probability for “Dr. No was amazing” be different from “was Dr. No amazing”?



Position encoding



Position embeddings

One option is to add learnable position embeddings $pe[i]$ to each word embedding e at position i (or concatenate them)

We can add two vectors if they're the same dimensionality

$$e_i = e_i + pe[i]$$

Or concatenate them if not

$$e_i = e_i \oplus pe[i]$$

0	2	-0.5	1.1	0.3	0.4	-0.5
1	-1.4	0.4	-0.2	-0.9	0.5	0.9
2	-1.1	-0.2	-0.5	0.2	-0.8	0
3	0.7	-0.3	1.5	-0.3	-0.4	0.1
4	-0.8	1.2	1	-0.7	-1	-0.4
5	0	0.3	-0.3	-0.9	0.2	1.4
6	0.8	0.8	-0.4	-1.4	1.2	-0.9
7	1.6	0.4	-1.1	0.7	0.1	1.6
...

position embeddings (pe)

Position encodings

Vaswani et al. 2017 use sinusoidal functions to deterministically create a **vector of position encodings the same dimensionality as the input** (d_{model}); the embedding dimensions progress from wavelength of 2π to $10000 * 2\pi$.

i = embedding
dimension

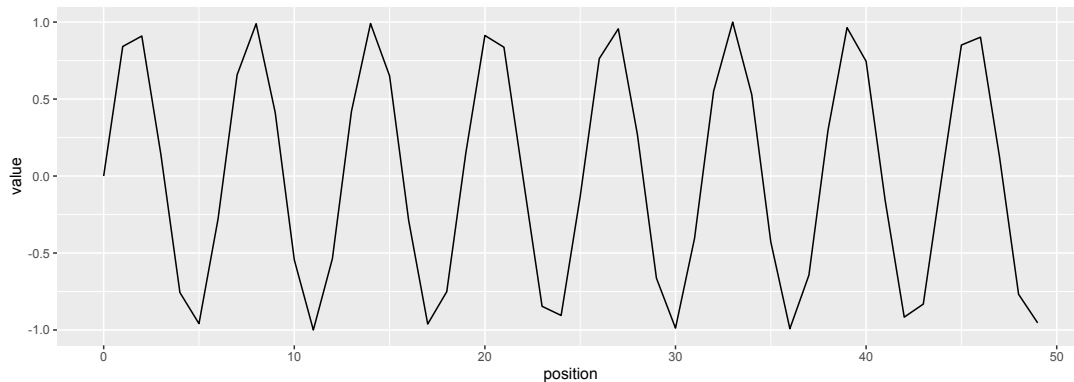
$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

pos = position in
sequence

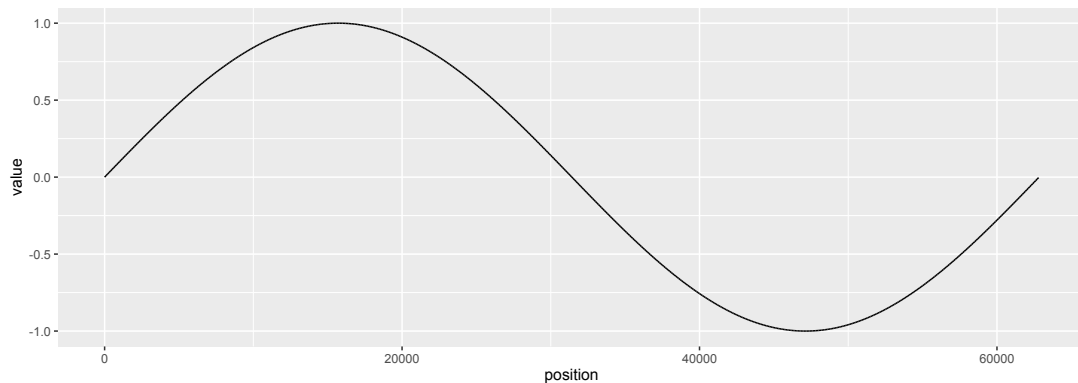
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Position encodings

Sine(x) values for integers x from positions 0 to 50, embedding index = 0 ($d_{\text{model}} = 100$)



Sine(x) values for integers x from positions 0 to 62,800, embedding index = 49 ($d_{\text{model}} = 100$)



Position encodings

We can see that the position embedding for word at position 20 is much more similar to the embedding for word at position 21 (closer) than that at position 98 (much further away).

dimension

position in sequence

	20	21	98
0	0.91	0.84	-0.57
10	-0.11	-0.49	0.25
20	-0.03	-0.19	0.18
30	0.30	0.24	1.00
40	0.48	0.50	0.63
50	0.98	0.98	0.56
60	0.08	0.08	0.38
70	1.00	1.00	0.99
80	0.01	0.01	0.06
90	1.00	1.00	1.00

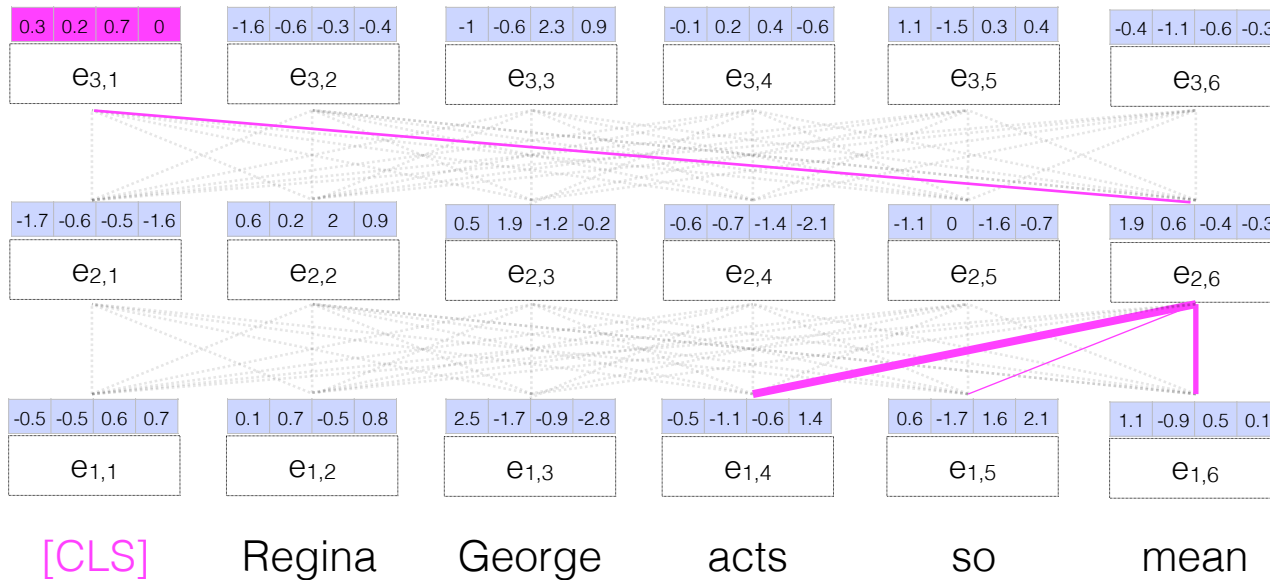
What does “mean” mean?

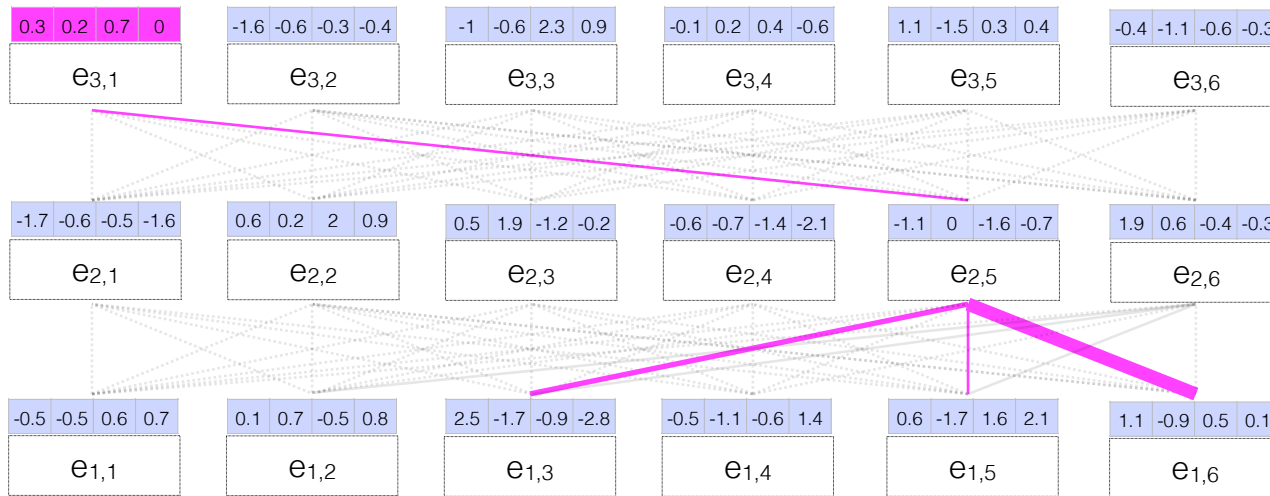
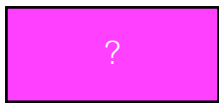
SPECIAL COLLECTOR'S EDITION

LINDSAY LOHAN AND TINA FEY
MEAN GIRLS



**"AWESOME, COOL AND
WICKED-GOOD!"**
- GENE SHALIT, TODAY





[CLS]

He

makes

a

mean

cocktail

Transformers

- Transformers have been extremely influential in NLP (Vaswani et al. 2017 has 35K citations!)
- We'll see them much more in this class in the context of specific applications:
 - Contextual language models, including causal self-attention (GPT), and bidirectional attention (BERT).
 - Machine translation
 - Text generation