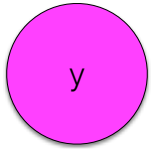
Applied Natural Language Processing

Info 256

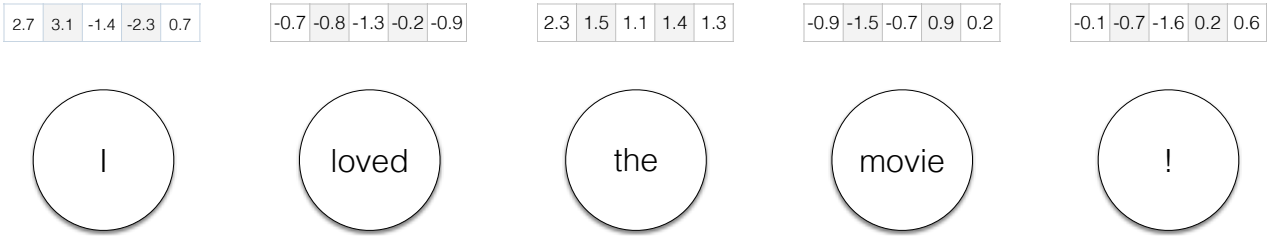
Lecture 12: Transformers (Oct 4, 2023)

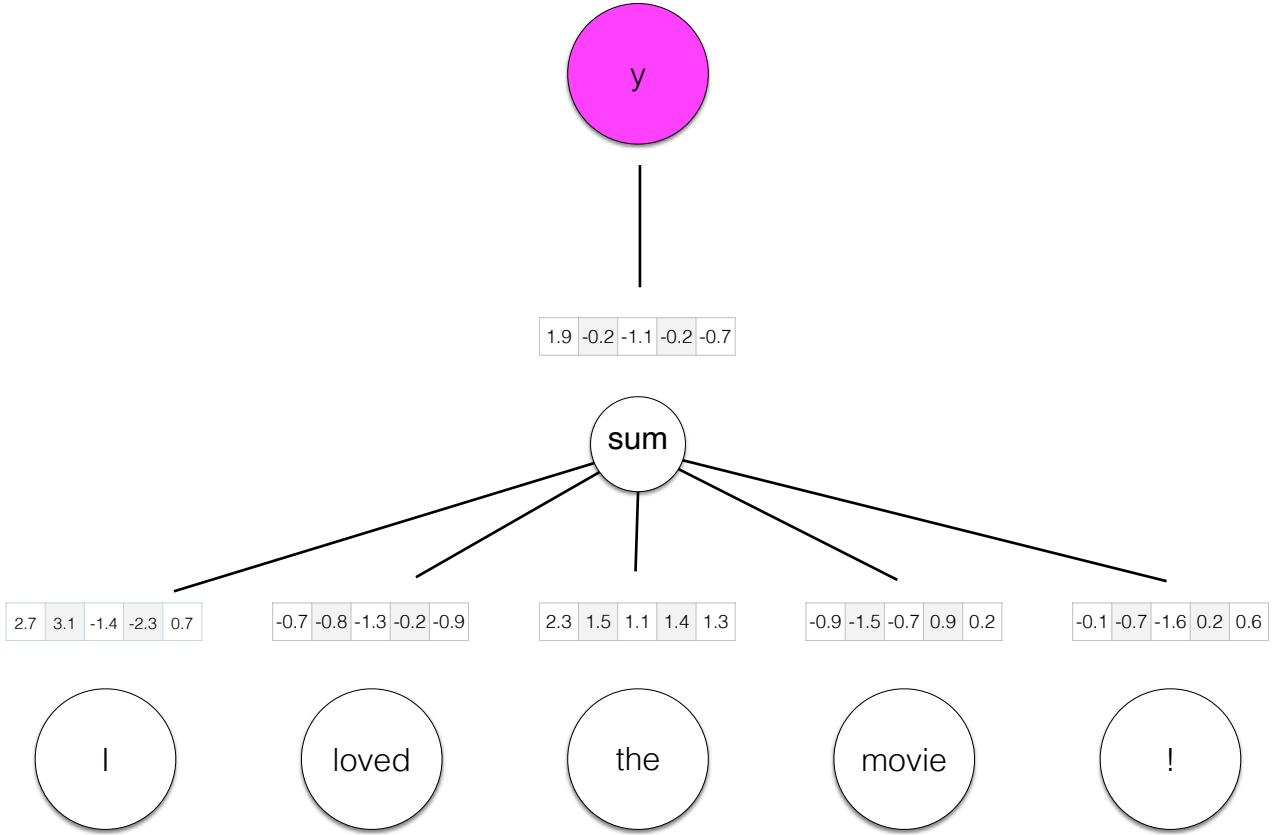
David Bamman, UC Berkeley

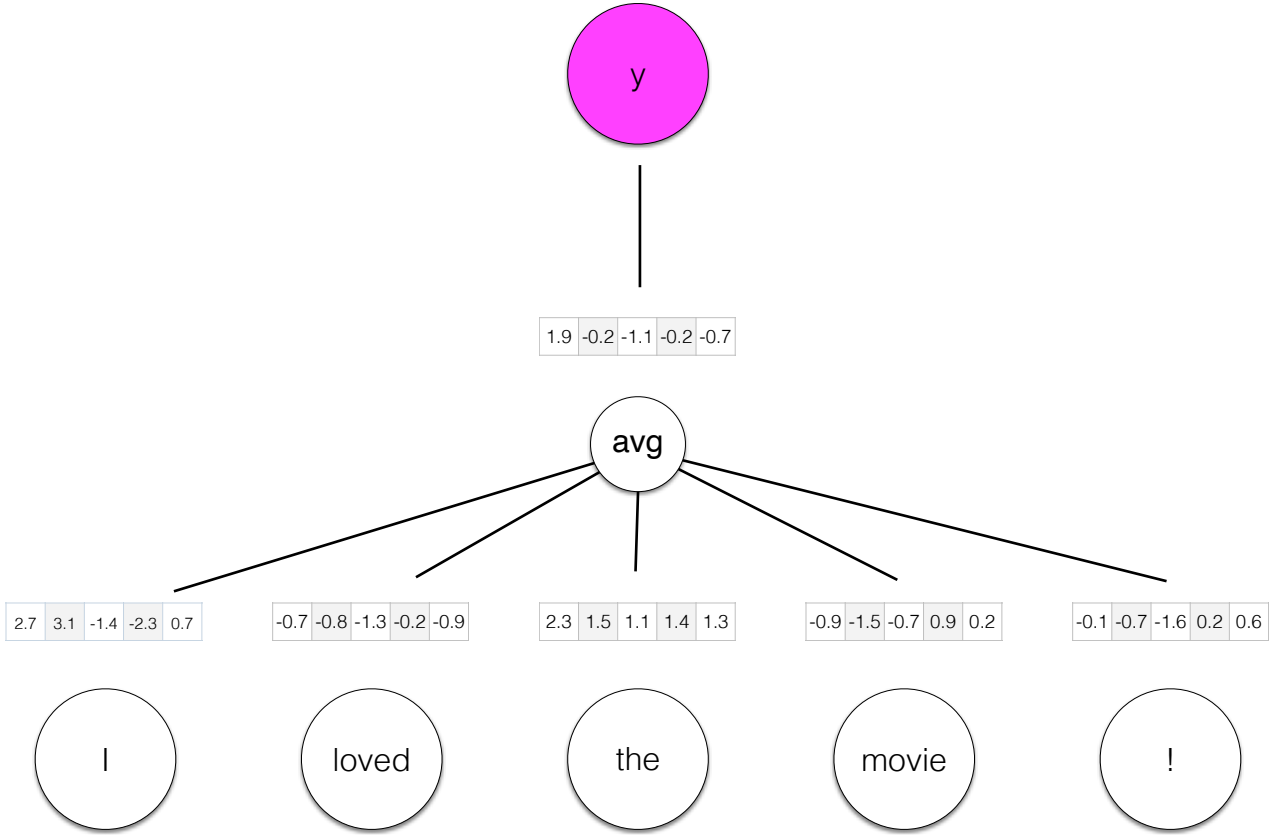
How do we use word embeddings for
document classification?


y

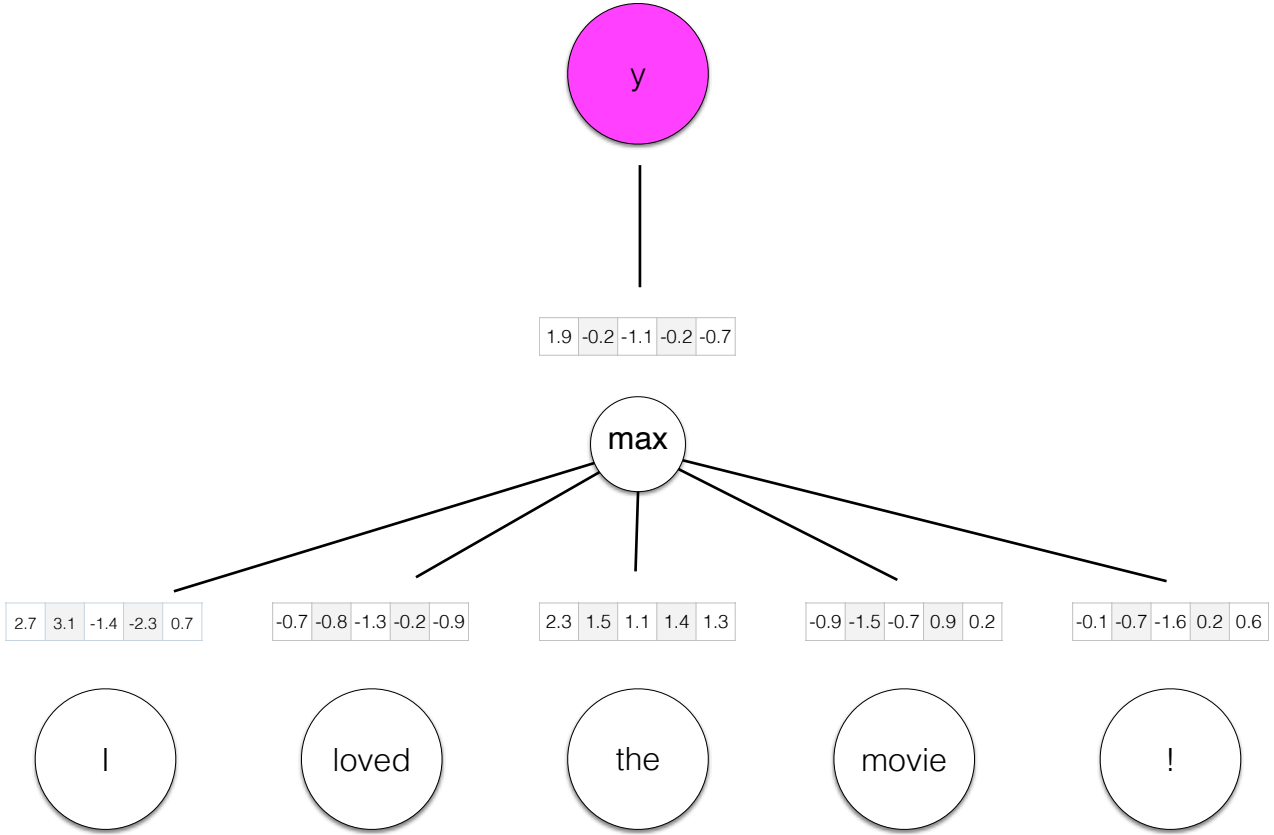
???

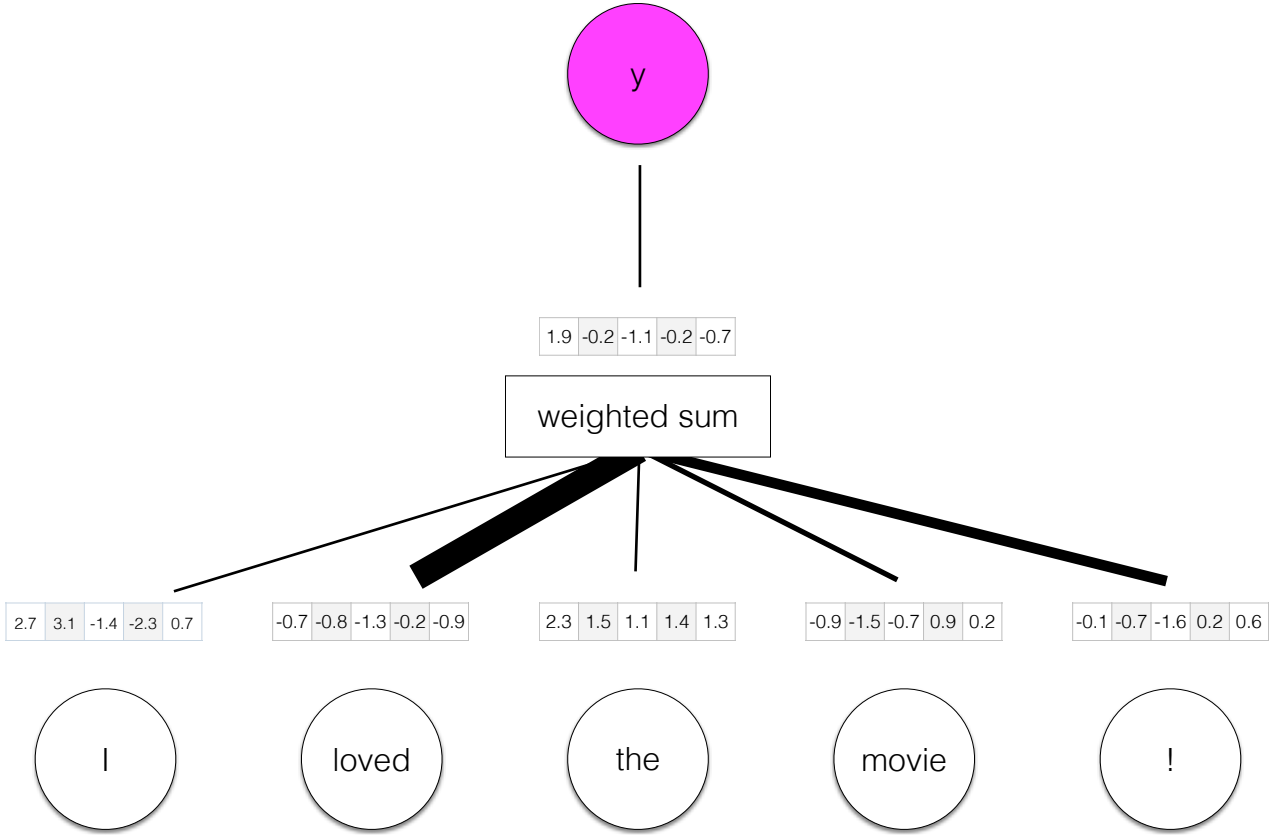

2.7 3.1 -1.4 -2.3 0.7
-0.7 -0.8 -1.3 -0.2 -0.9
2.3 1.5 1.1 1.4 1.3
-0.9 -1.5 -0.7 0.9 0.2
-0.1 -0.7 -1.6 0.2 0.6
I
loved
the
movie
!


y
1.9 -0.2 -1.1 -0.2 -0.7
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
!



Iyyer et al. (2015), “Deep Unordered Composition Rivals Syntactic Methods for Text Classification” (ACL)


y
1.9 -0.2 -1.1 -0.2 -0.7
max
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
!


y
1.9 -0.2 -1.1 -0.2 -0.7
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
!

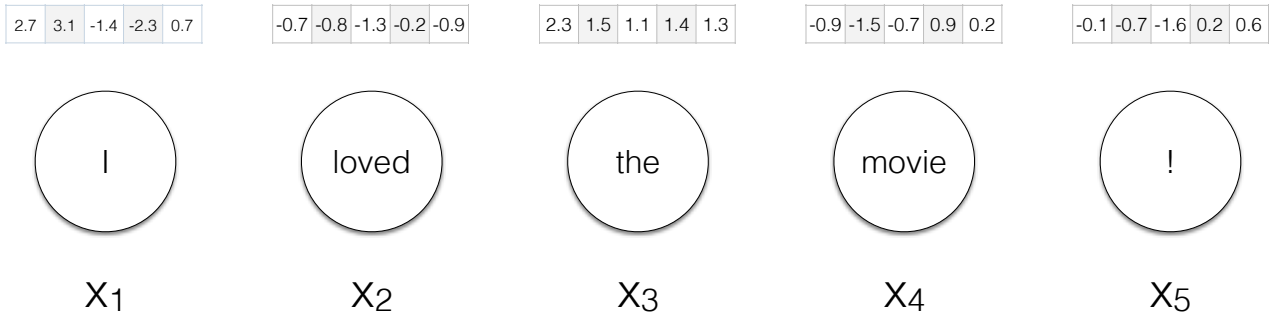
Attention

- 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



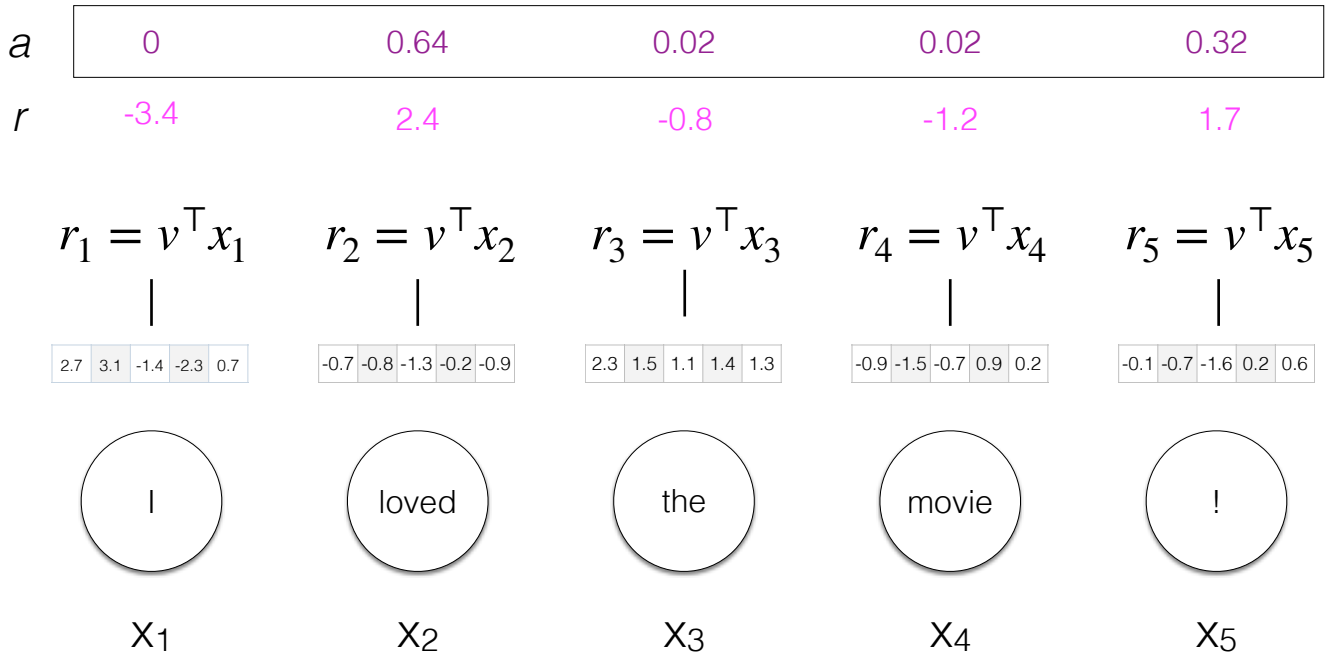
$$v \in \mathcal{R}^H$$

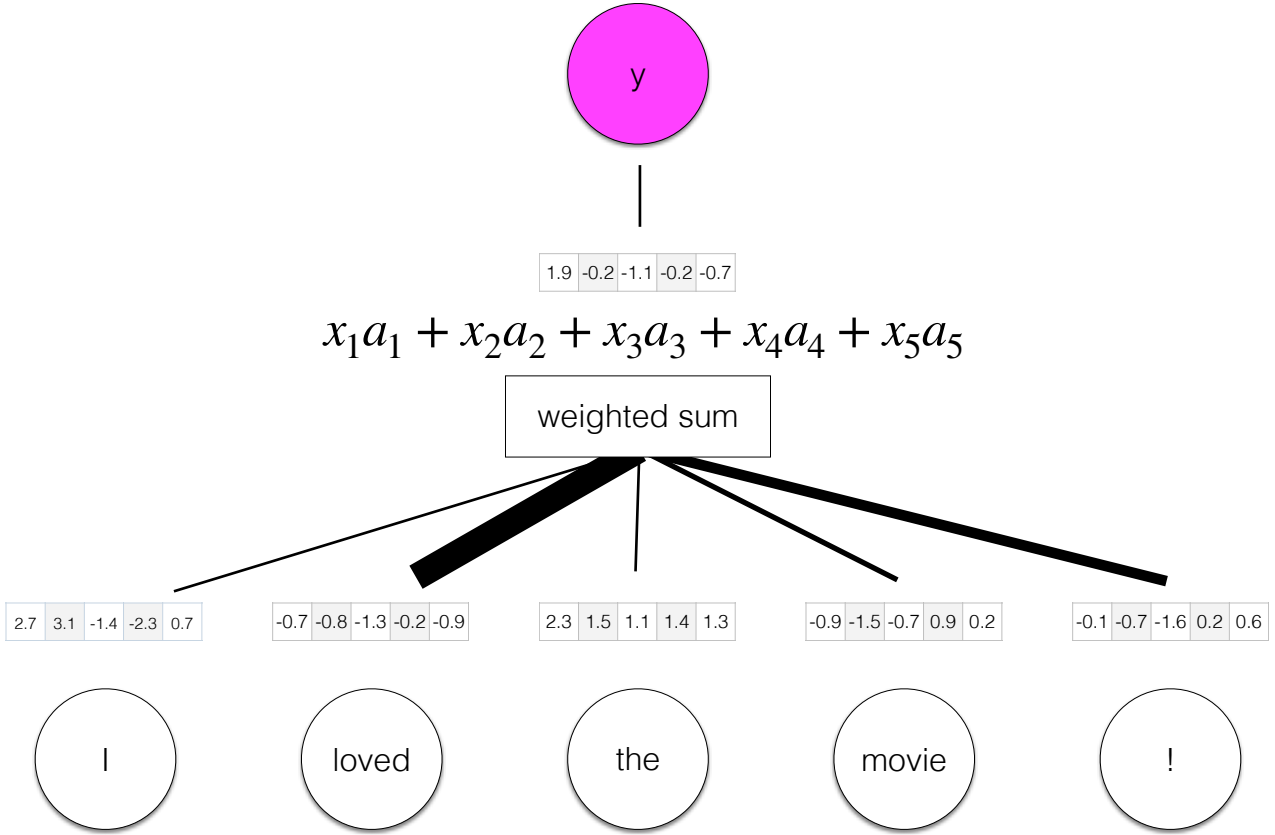
2.7	3.1	-1.4	-2.3	0.7

-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.

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



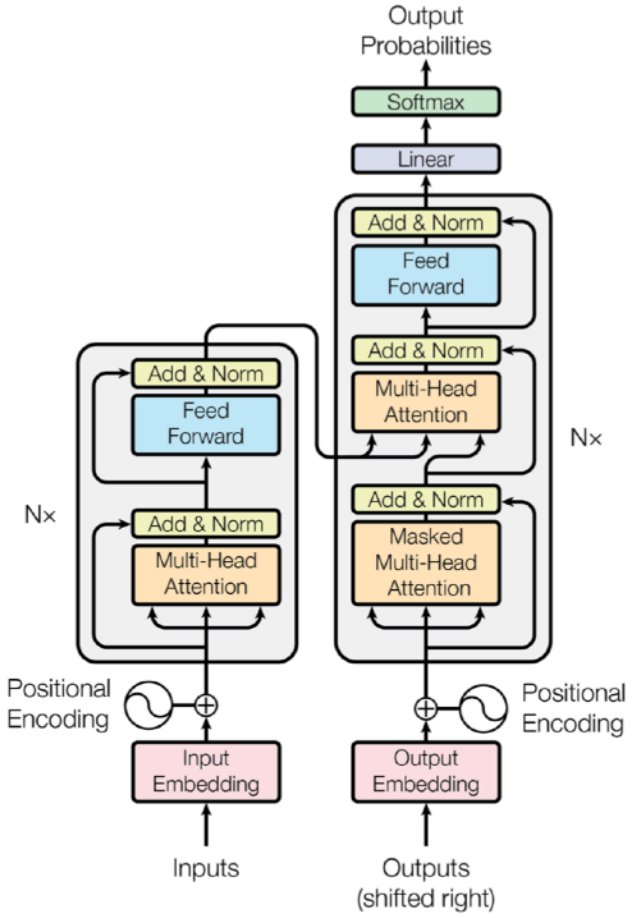

y
1.9 -0.2 -1.1 -0.2 -0.7
$x_1a_1 + x_2a_2 + x_3a_3 + x_4a_4 + x_5a_5$
weighted sum
2.7 3.1 -1.4 -2.3 0.7
-0.7 -0.8 -1.3 -0.2 -0.9
2.3 1.5 1.1 1.4 1.3
-0.9 -1.5 -0.7 0.9 0.2
-0.1 -0.7 -1.6 0.2 0.6
I
loved
the
movie
!

Attention

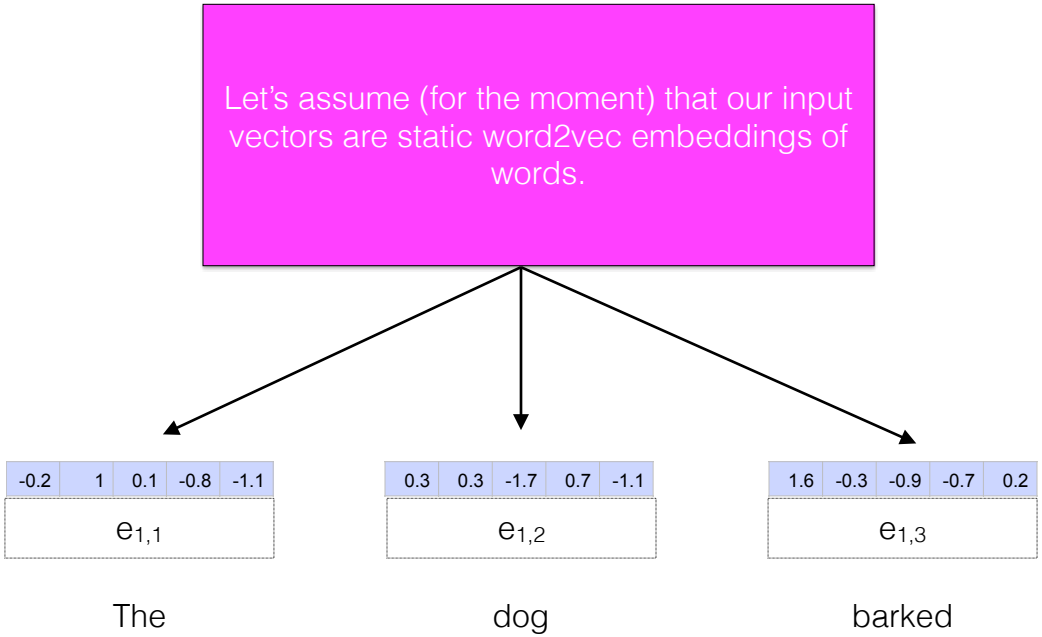
- Lots of variations on attention:
  - Linear transformation of x into before dotting with v
  - Non-linearities after each operation.
  - “Multi-head attention”: multiple v vectors to capture different phenomena that can be attended to in the input.
  - Hierarchical attention (sentence representation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

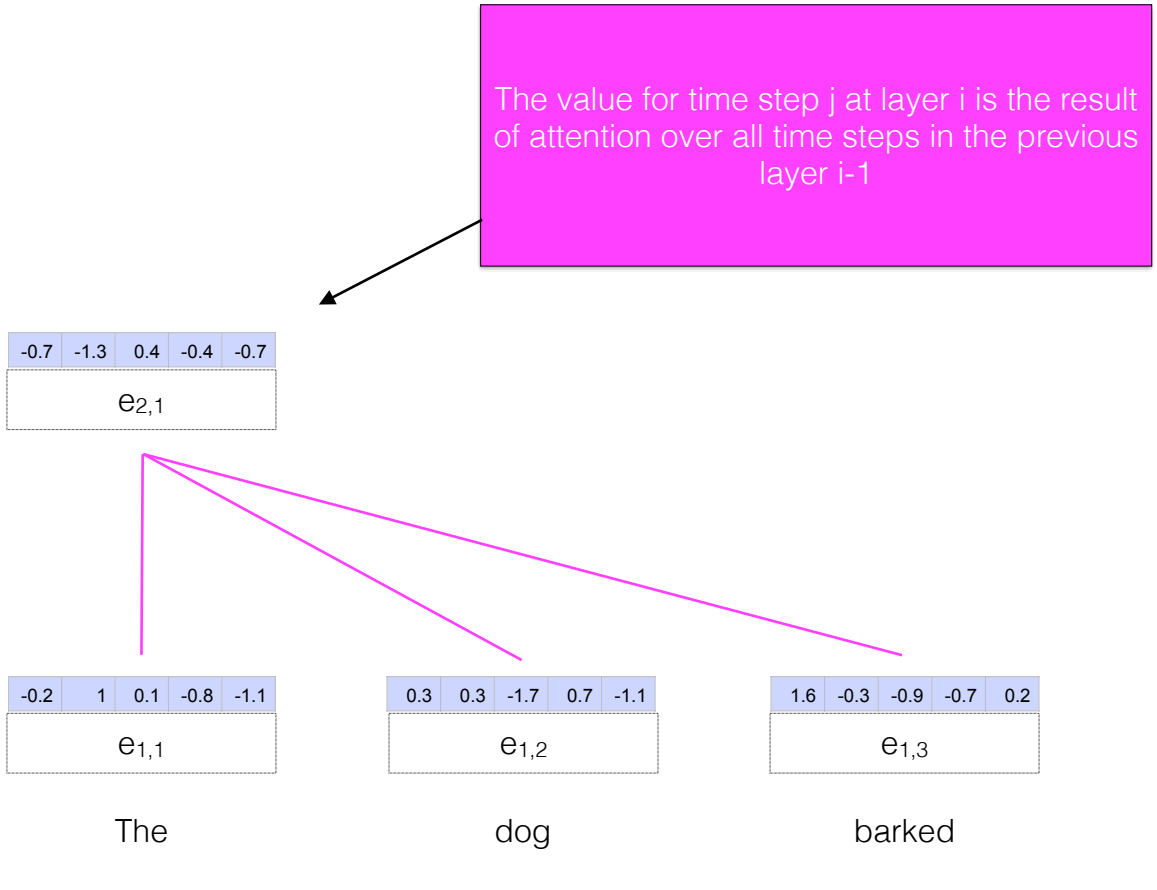
Transformers

- 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



Self-Attention




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

- 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

key

value

original value

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

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

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

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

$e_{1,1}$

The

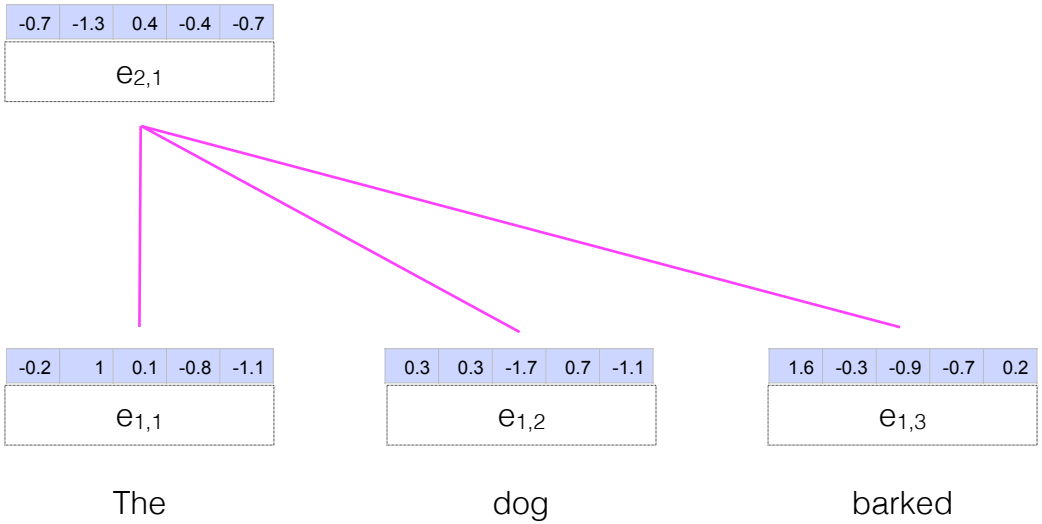
$$W^Q \in \mathbb{R}^{100\times37}$$

$$W^K \in \mathbb{R}^{100\times37}$$

$$W^V \in \mathbb{R}^{100\times100}$$

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

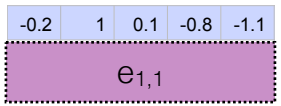
Self attention **from "The"** at position 1 to every token in the sentence



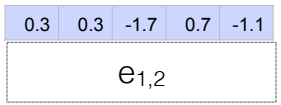
- The compatibility score between two words is the dot product between their respective query and key vectors.

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

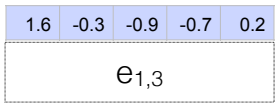
a	0.07	0.58	0.35	a = softmax(scores)
scores	-1.4	0.64	0.14	
	$q_1 \cdot k_1$	$q_1 \cdot k_2$	$q_1 \cdot k_3$	



The



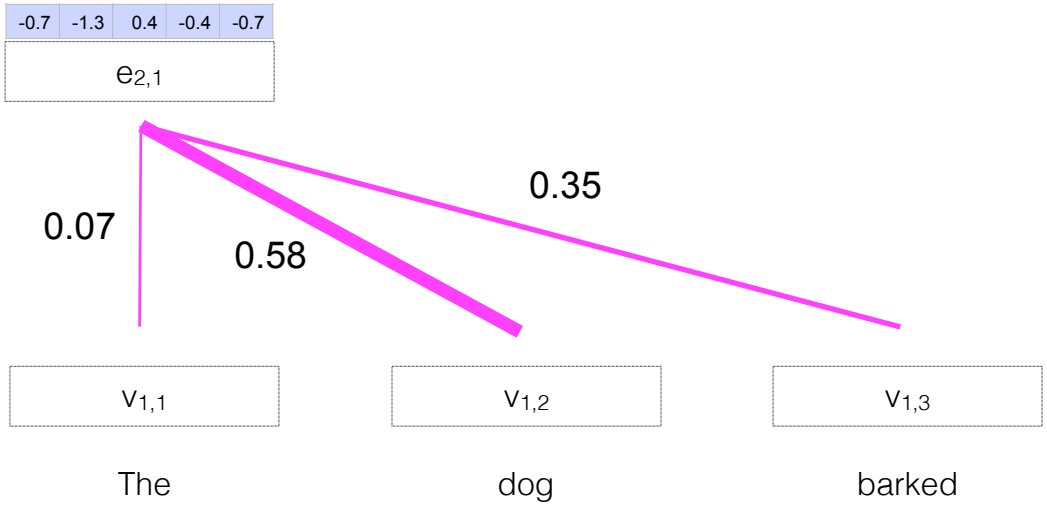
dog



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 ($\sqrt{d_k}$)

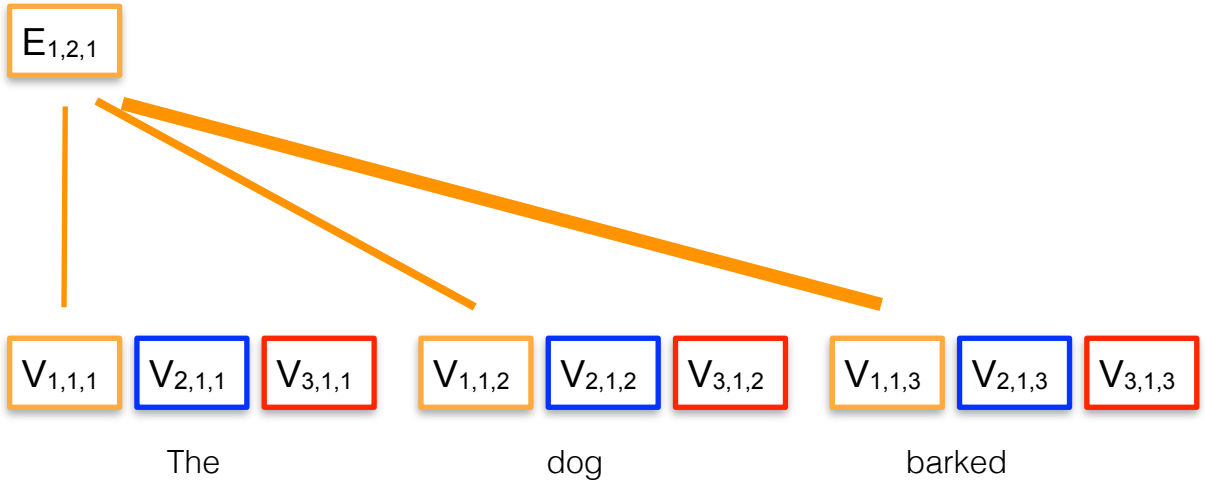
Multihead attention

- Attention provides one view on the data; we can learn multiple perspectives in a transformer by learning multiple (W^Q, W^K, W^V) sets.

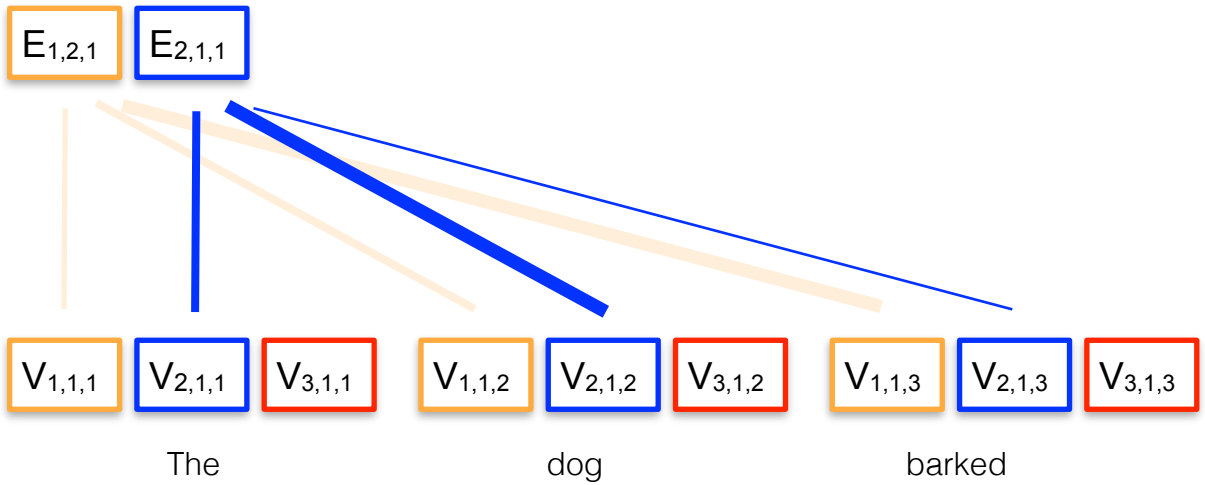
$$Q_i = XW_i^Q; K = XW_i^K; V = XW_i^V$$

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

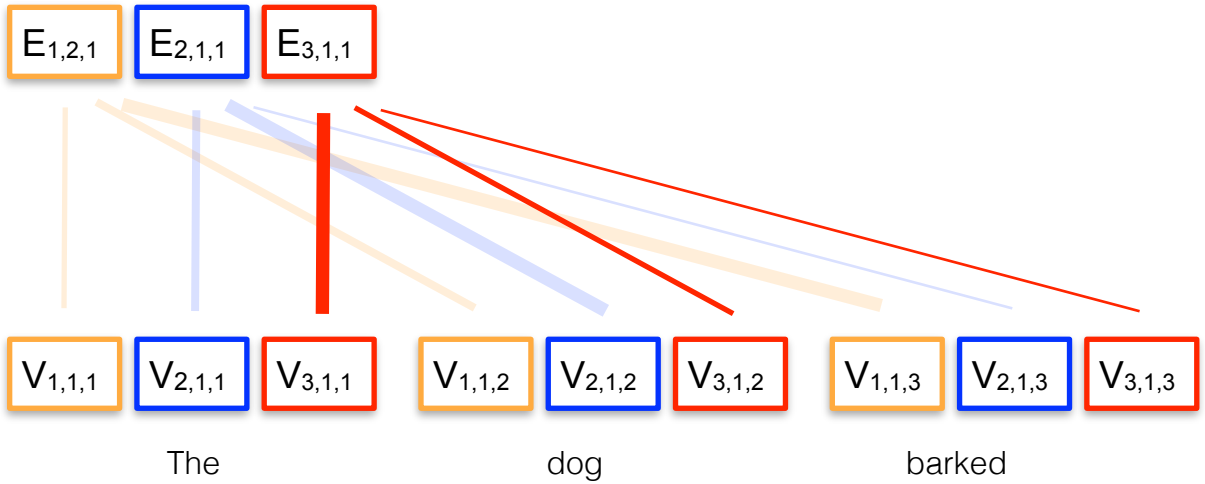
- With multihead attention, each attention head generates its own output vector i based on its own W_i^Q, W_i^K and W_i^V



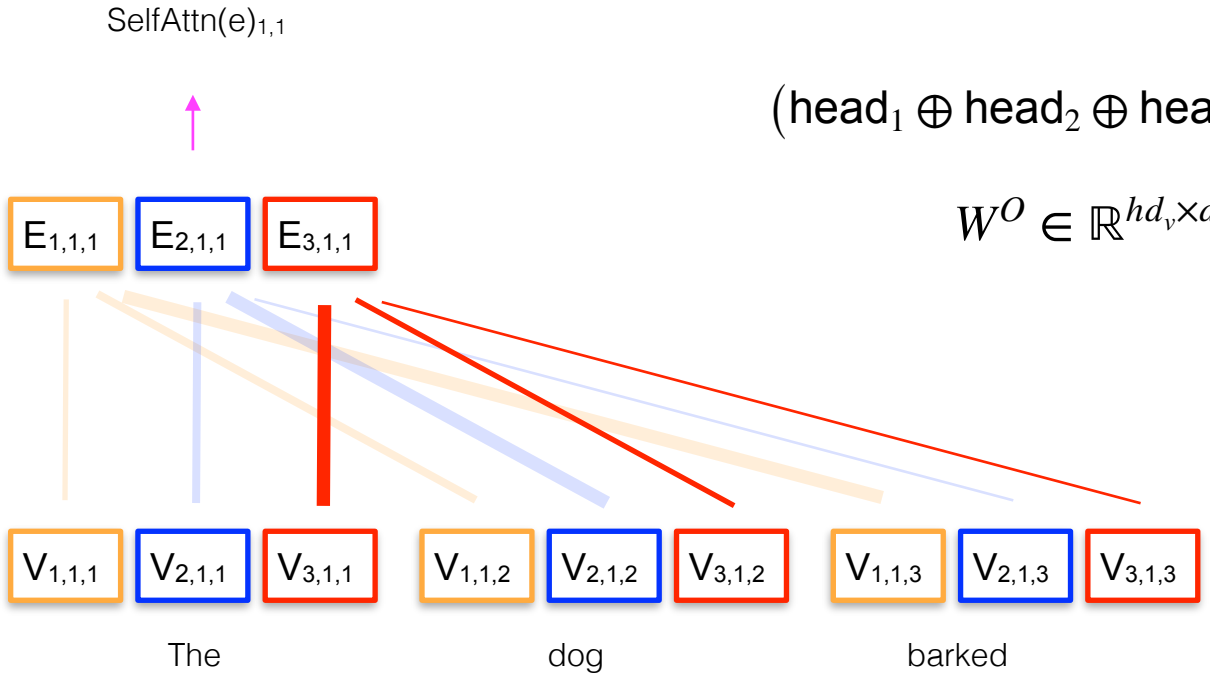
- With multihead attention, each attention head generates its own output vector i based on its own W_i^Q, W_i^K and W_i^V



- 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

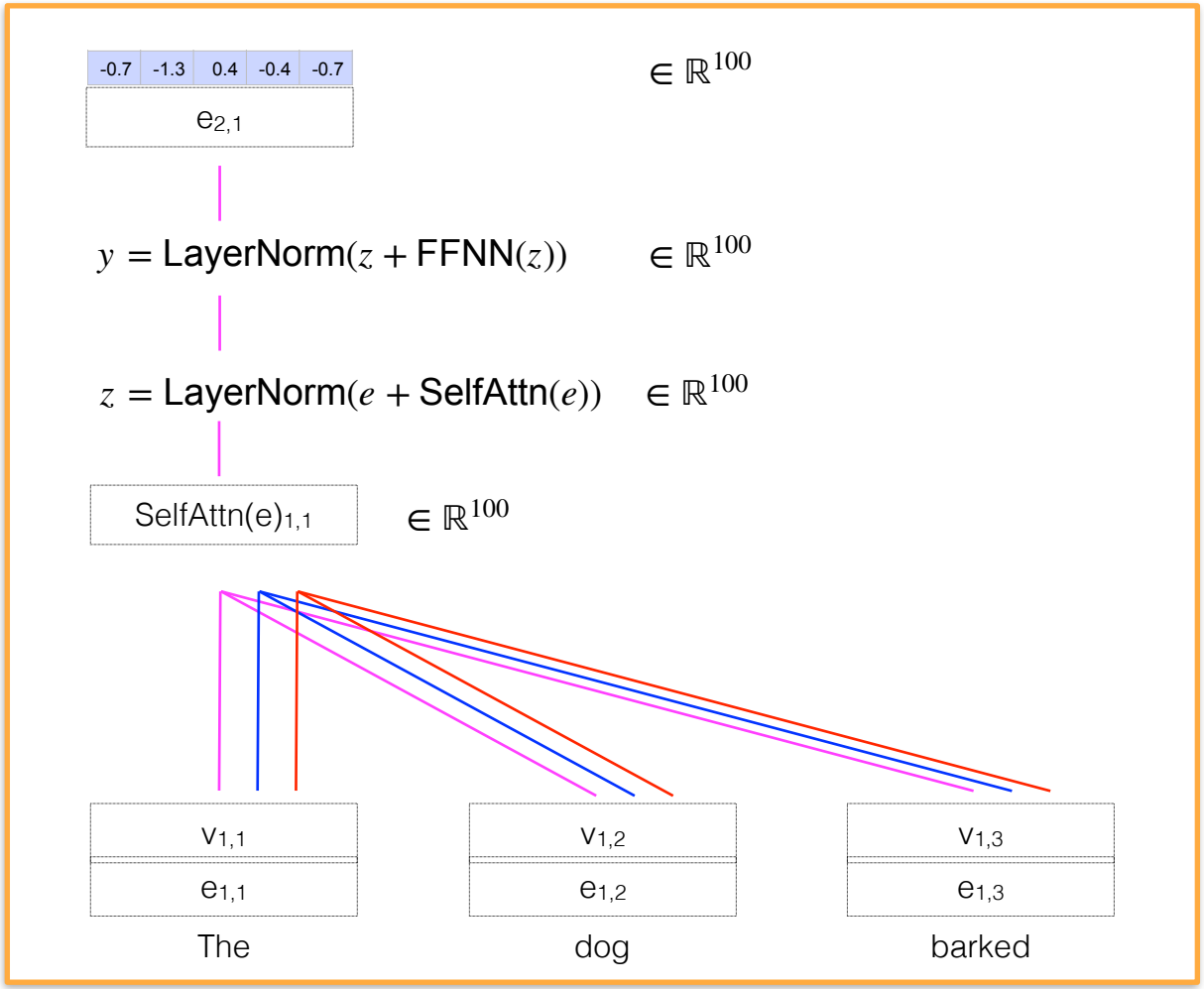


- These *h* separate output heads are then concatenated together and linearly transformed back to the original dimensionality d_v



$$\left(\text{head}_1 \oplus \text{head}_2 \oplus \text{head}_3\right) W^O$$

$$W^O \in \mathbb{R}^{hd_v \times d_v}$$


-0.7 -1.3 0.4 -0.4 -0.7
$e_{2,1}$
$\in \mathbb{R}^{100}$
$y = \text{LayerNorm}(z + \text{FFNN}(z))$
$\in \mathbb{R}^{100}$
$z = \text{LayerNorm}(e + \text{SelfAttn}(e))$
$\in \mathbb{R}^{100}$
$\text{SelfAttn}(e)_{1,1}$
$\in \mathbb{R}^{100}$
$v_{1,1}$
$e_{1,1}$
$v_{1,2}$
$e_{1,2}$
$v_{1,3}$
$e_{1,3}$
The
dog
barked

Layer Normalization

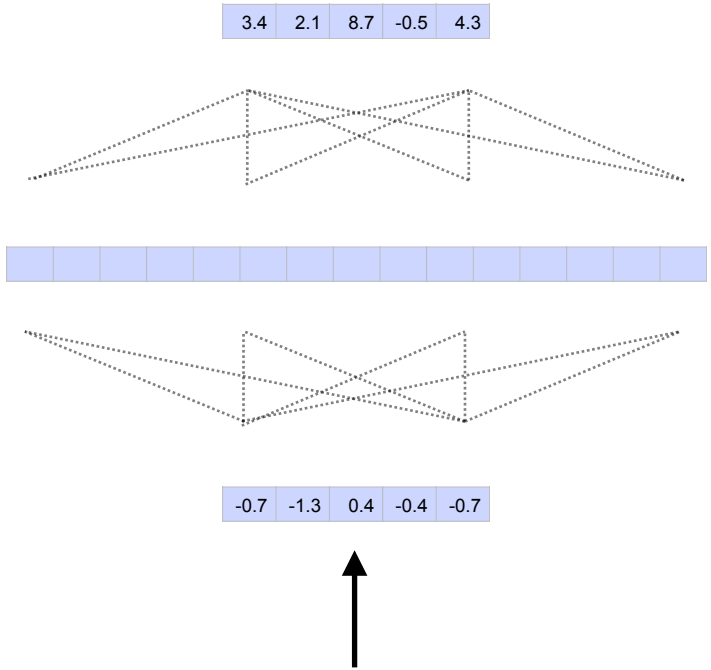
- Transform each output from a layer d_h into its z-score, with two learnable parameters γ and β.

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

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

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

FFNN



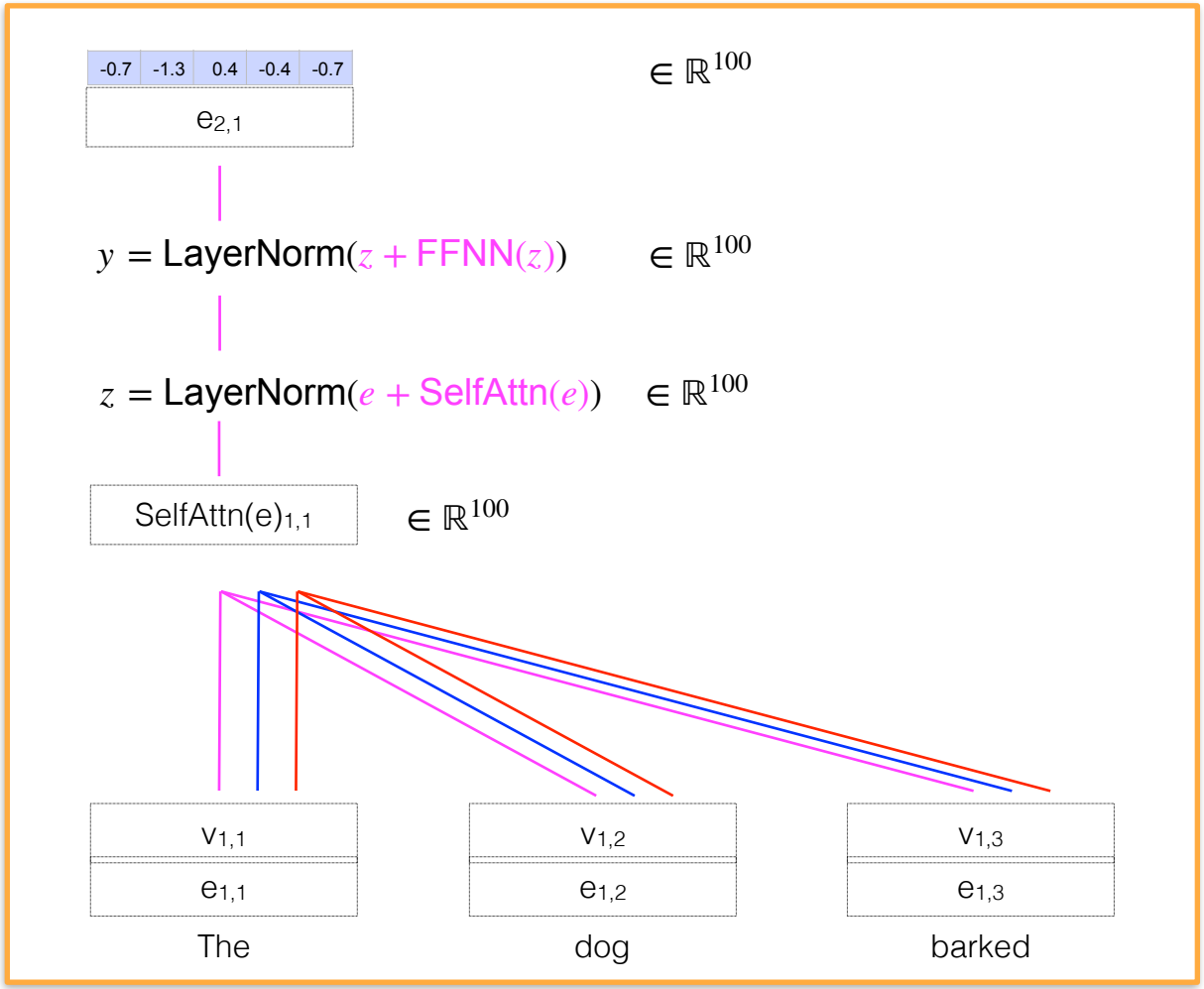
$$W^2 \in \mathbb{R}^{d_{ff} \times d_v}$$

ReLU

$$W^1 \in \mathbb{R}^{d_v \times d_{ff}}$$

In Vaswani et al, 2018
$d_v = 512$, $d_{ff} = 2048$

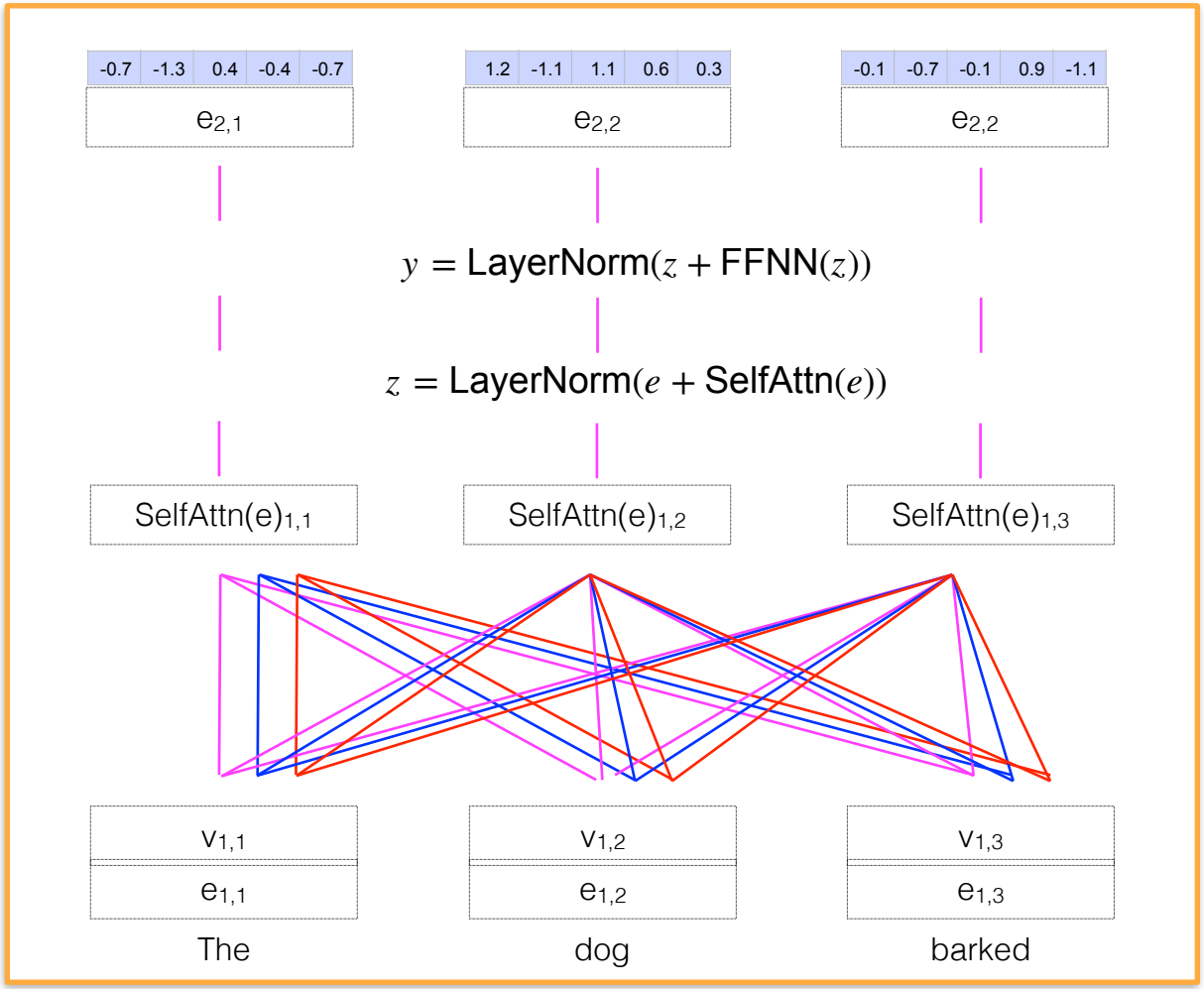
- Residual layers add a layer's input to its output, giving later layers access to unmediated information.



Output

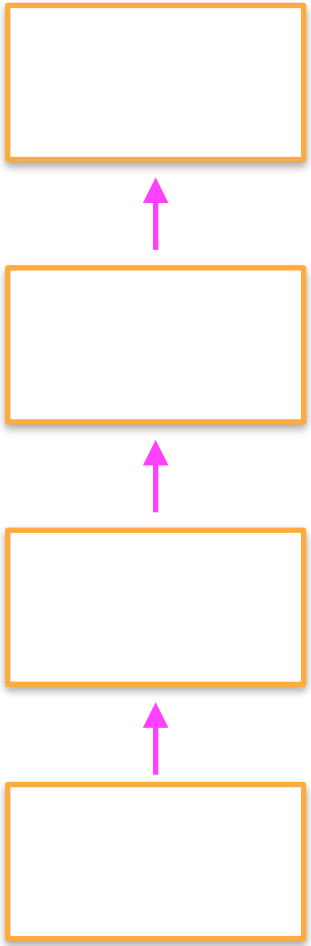
This whole process defines one attention block. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

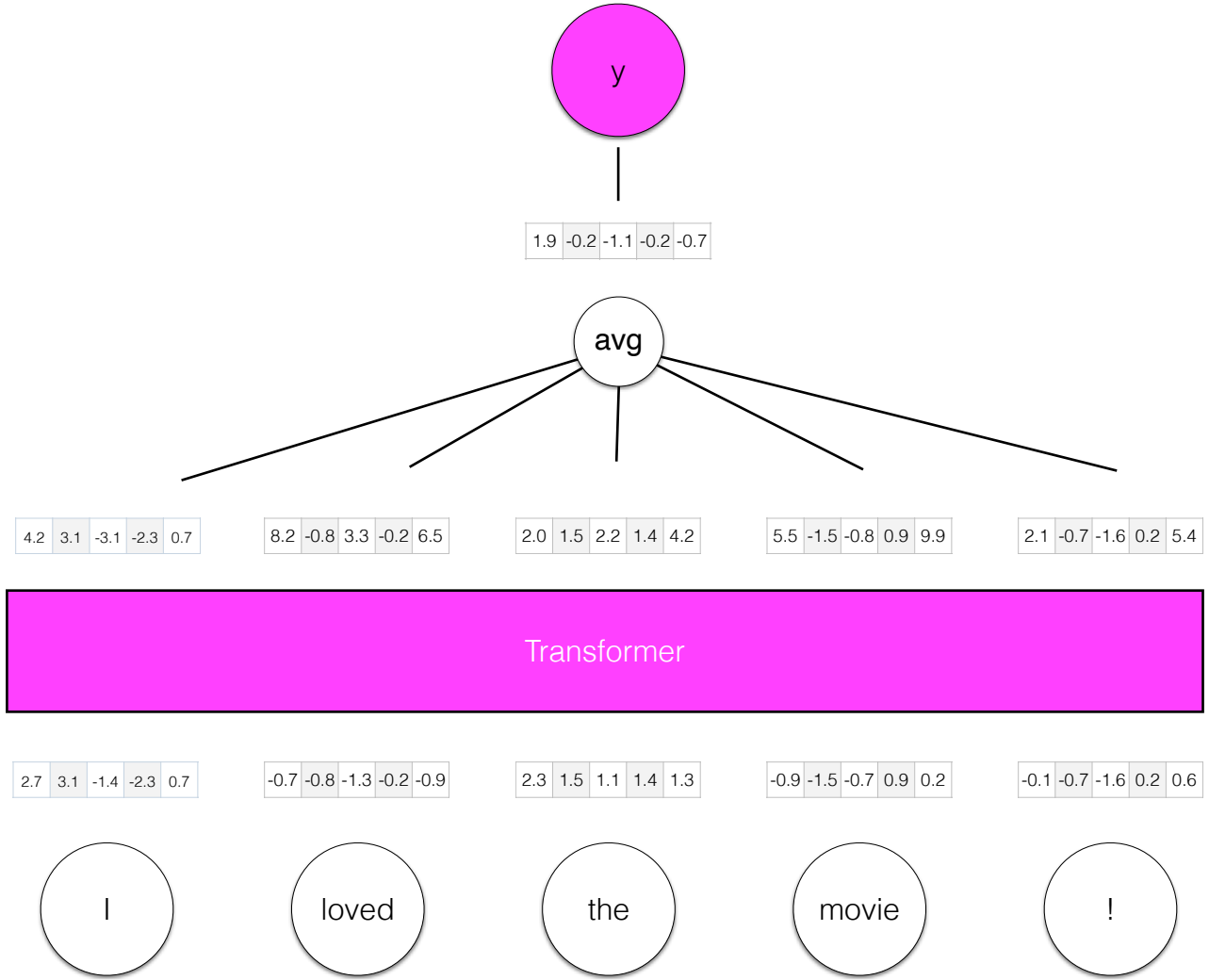
Input



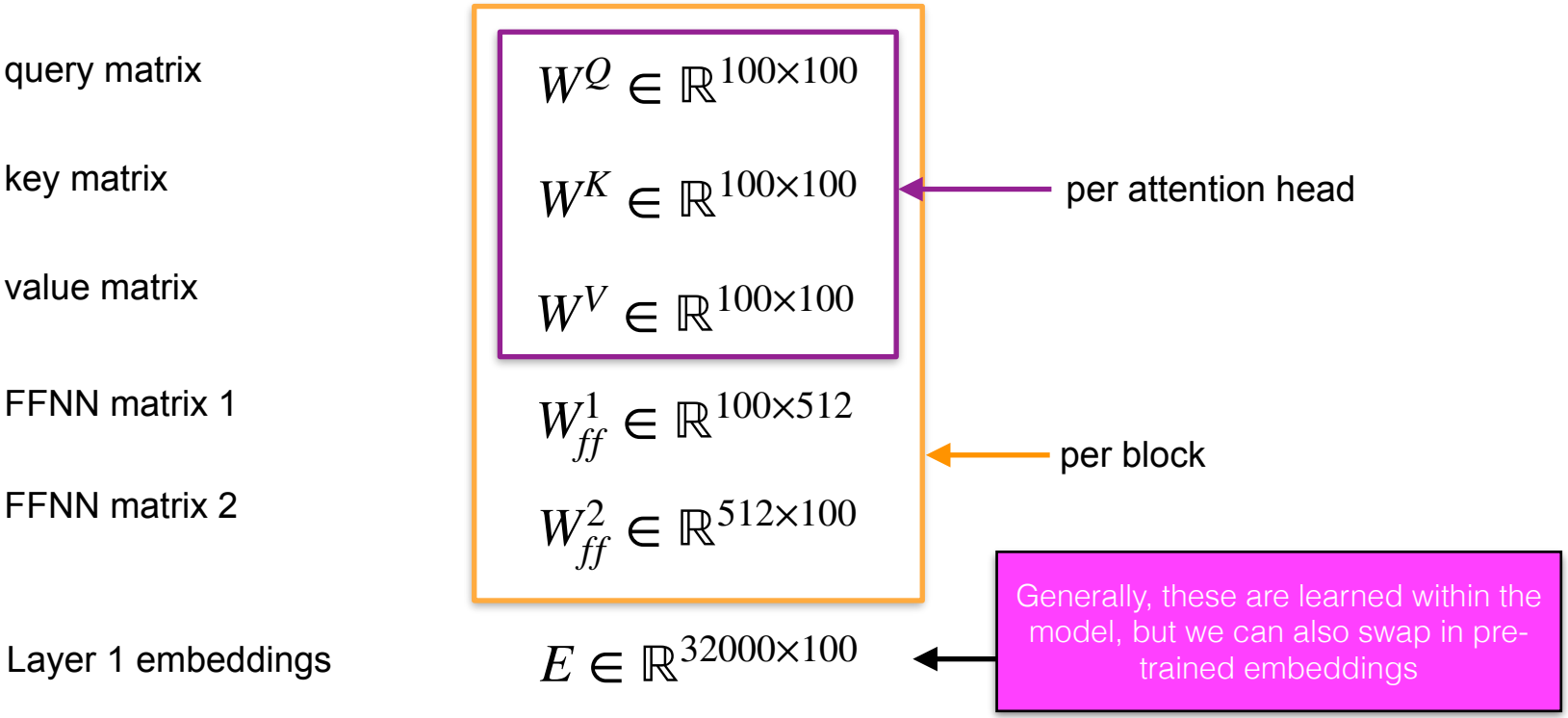
This whole process defines one attention block. The input is a sequence of (e.g. 100-dimensional) vectors; the output of each block is a sequence of (100-dimensional) vectors.

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

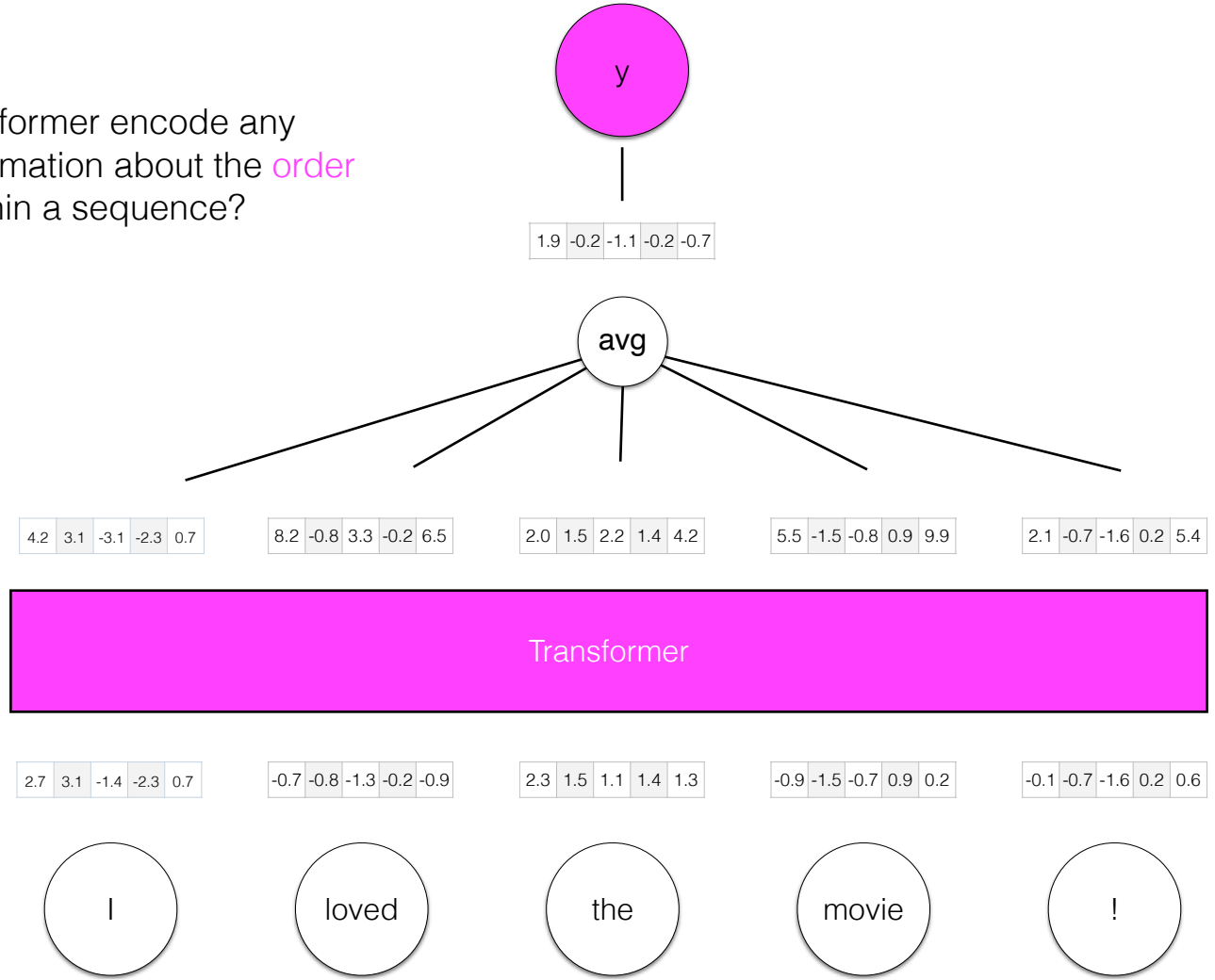



y
1.9 -0.2 -1.1 -0.2 -0.7
avg
4.2 3.1 -3.1 -2.3 0.7
8.2 -0.8 3.3 -0.2 6.5
2.0 1.5 2.2 1.4 4.2
5.5 -1.5 -0.8 0.9 9.9
2.1 -0.7 -1.6 0.2 5.4
Transformer
2.7 3.1 -1.4 -2.3 0.7
-0.7 -0.8 -1.3 -0.2 -0.9
2.3 1.5 1.1 1.4 1.3
-0.9 -1.5 -0.7 0.9 0.2
-0.1 -0.7 -1.6 0.2 0.6
I
loved
the
movie
!

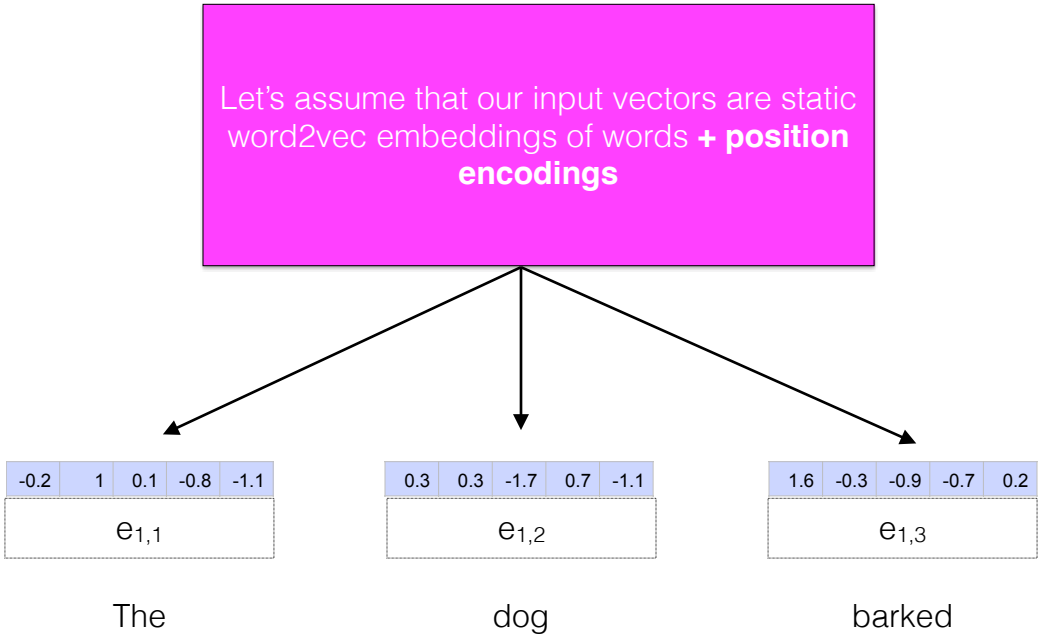
Parameters


query matrix
key matrix
value matrix
FFNN matrix 1
FFNN matrix 2
Layer 1 embeddings
$W^Q \in \mathbb{R}^{100\times100}$
$W^K \in \mathbb{R}^{100\times100}$
$W^V \in \mathbb{R}^{100\times100}$
$W^1_{ff} \in \mathbb{R}^{100\times512}$
$W^2_{ff} \in \mathbb{R}^{512\times100}$
$E \in \mathbb{R}^{32000\times100}$
per attention head
per block
Generally, these are learned within the model, but we can also swap in pre-trained embeddings

Does a transformer encode any intrinsic information about the order of words within a sequence?



Position encoding



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

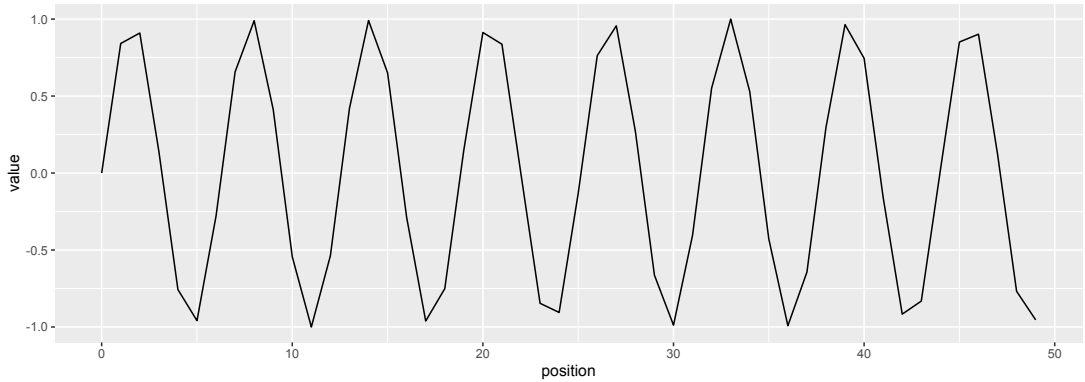
pos = position in sequence

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

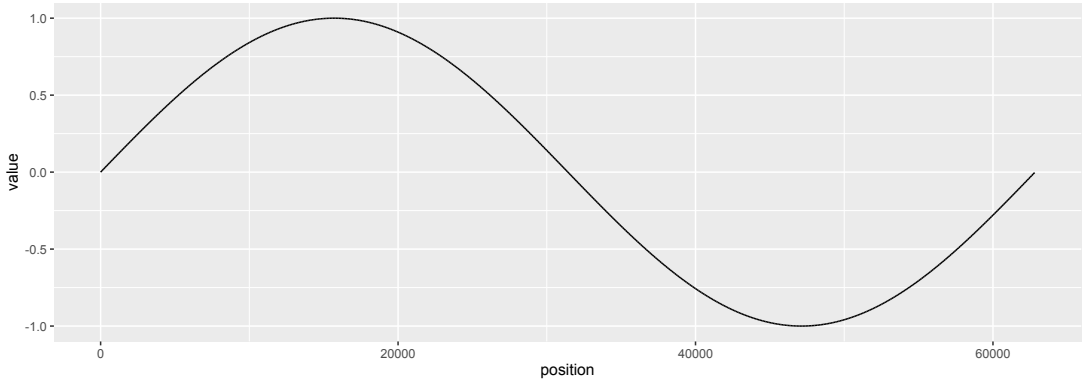
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Position encodings

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



Sine(x) values for integers x from positions 0 to 62,800, embedding index = 49 (d_{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).

position in sequence

dimension	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

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).

Activity

`0.setup/Colab_Intro.ipynb`

`https://pytorch.org/tutorials/beginner/basics/intro.html`