Transformers Part 1

Lecture 17

Subir Varma

So Far …

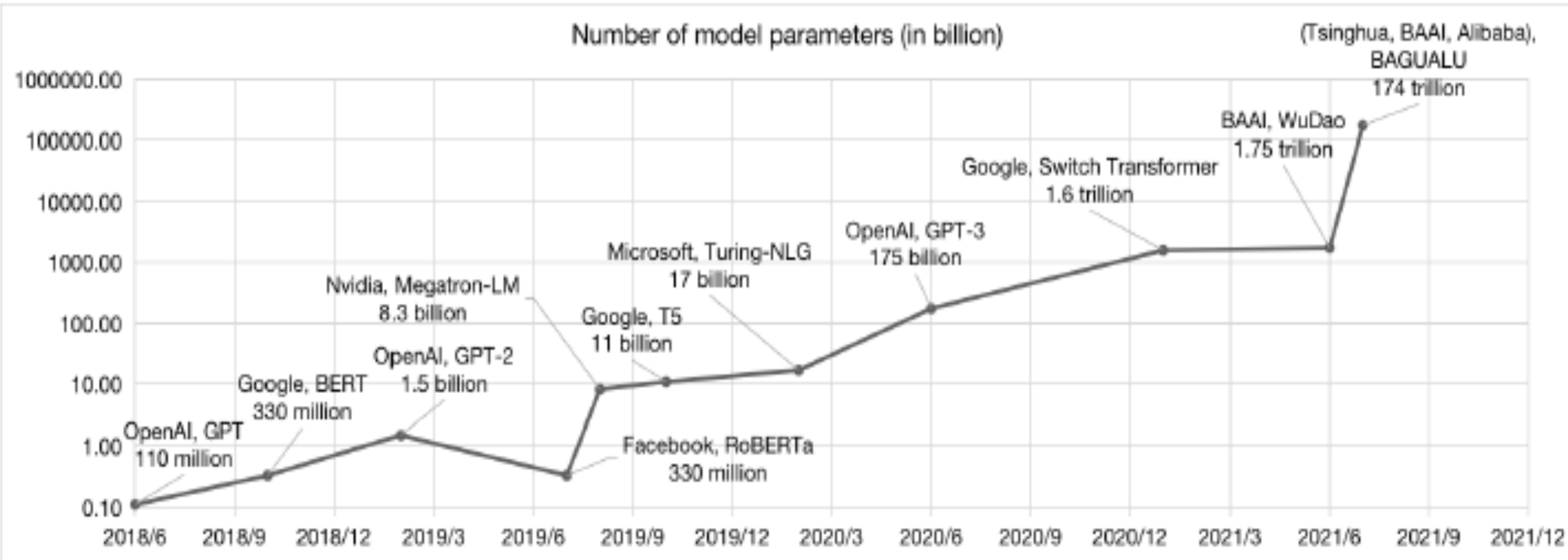
Lecture 3		Lecture 4		Lecture 10-12		Lecture 13		Lecture 14		Lecture 16
Linear	→	Dense Feed Forward	→	ConvNets	→	RNNs	→	LSTMs	→	Attention
		(vectors)		(images)				(sequences)		

▼

Next: Transformers

Also used for Sequences

Transformers

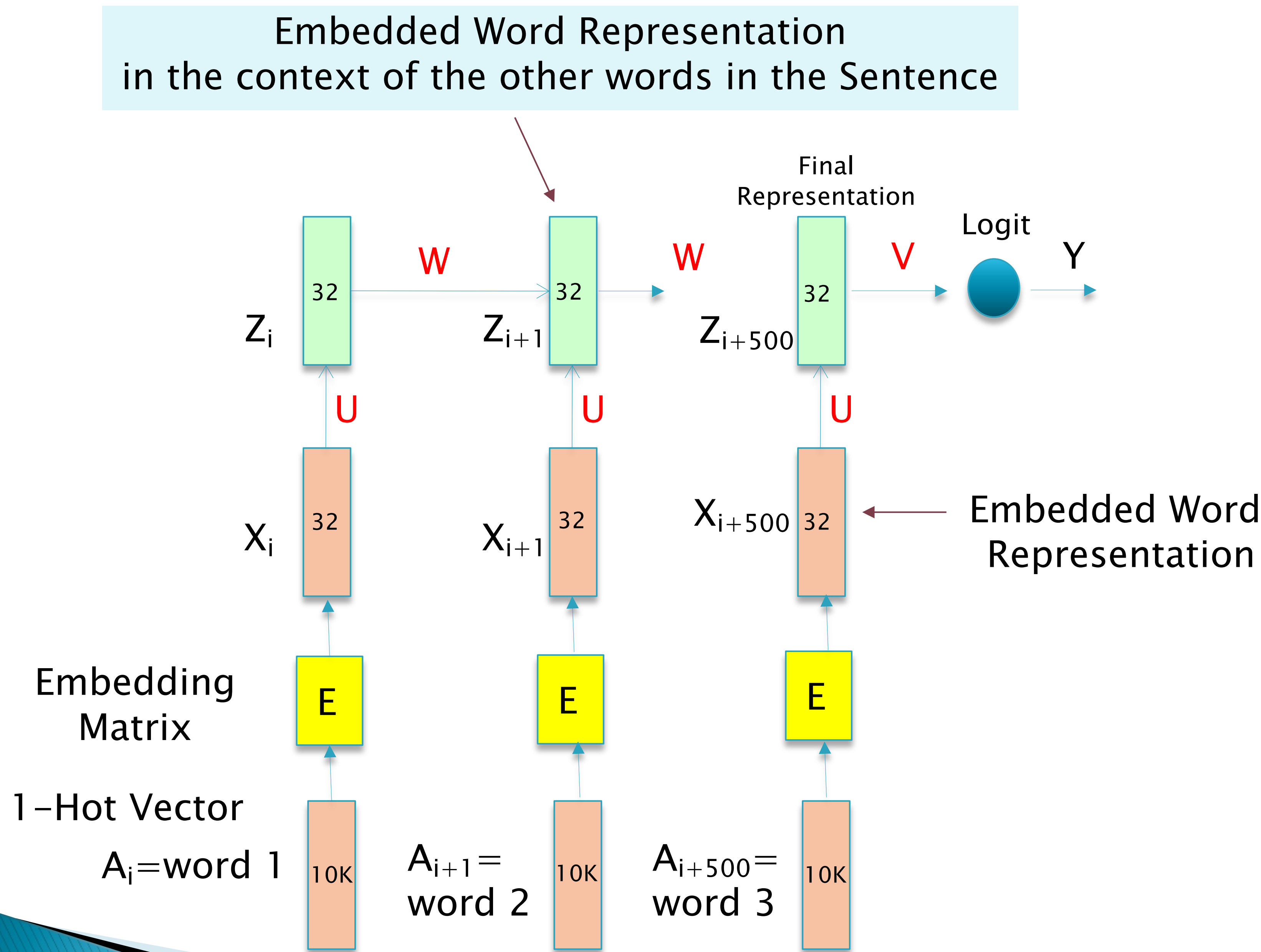


- Has made possible much larger models (higher capacity)
- Can be trained using Self Supervised Learning, so very large datasets are available

Huge Models ⟷ Huge Datasets

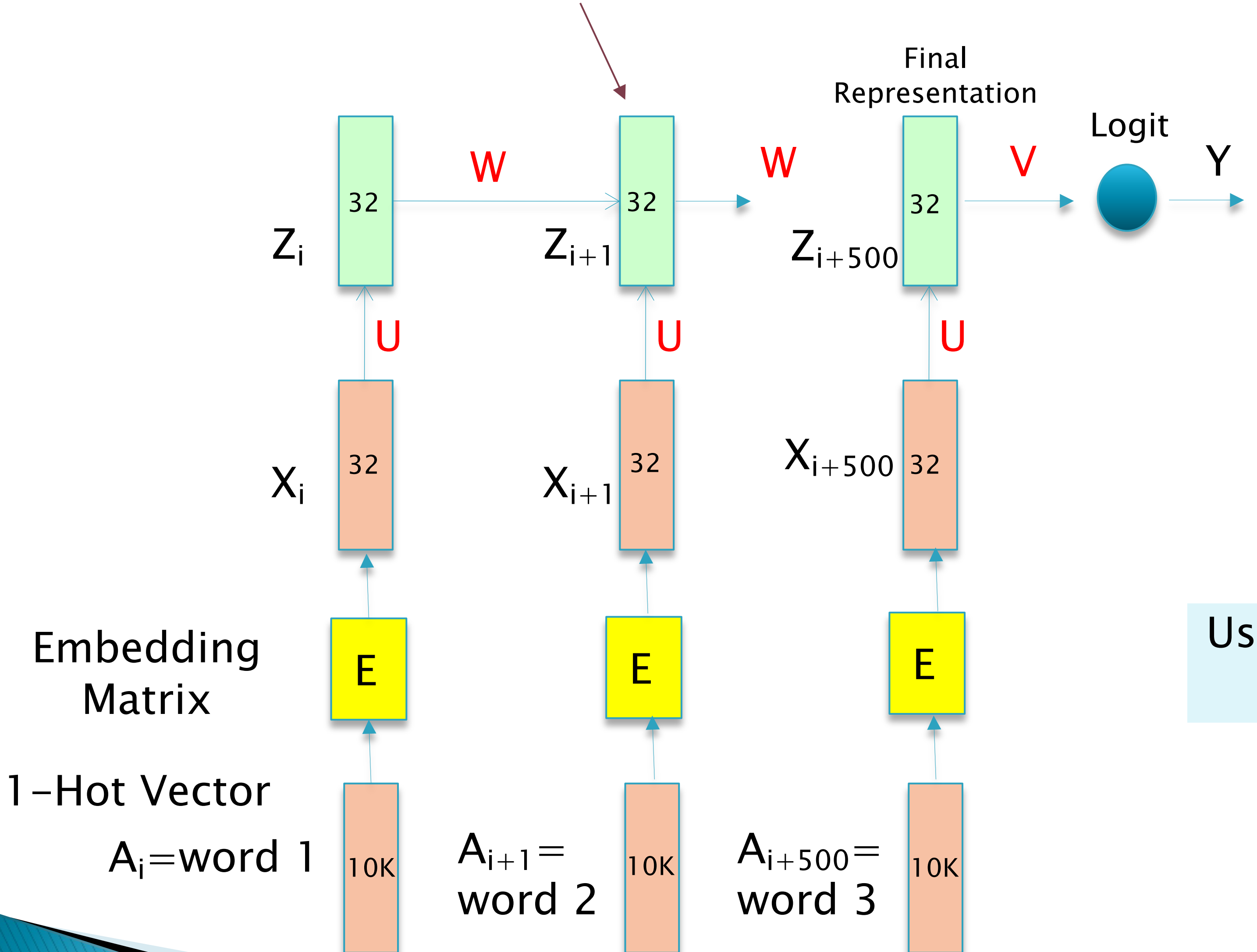
Problems with RNNs/LSTMs

- Sequential computation prevents parallelization
- Despite GRUs and LSTMs, RNNs still need the attention mechanism to deal with long range dependencies – path length for codependent computation between states grows with sequence
- But if attention gives us access to any state… maybe we don't need the RNN?


Embedded Word Representation
in the context of the other words in the Sentence
Final
Representation
Logit
Y
W
W
V
32
32
32
Zi
Zi+1
Zi+500
U
U
U
Xi
Xi+1
Xi+500
32
32
32
Embedded Word
Representation
Embedding
Matrix
E
E
E
1-Hot Vector
Ai=word 1
Ai+1=
word 2
Ai+500=
word 3
10K
10K
10K

But this representation is only a function of the words that came before it

Why not make it a function of all the words in the sentence!

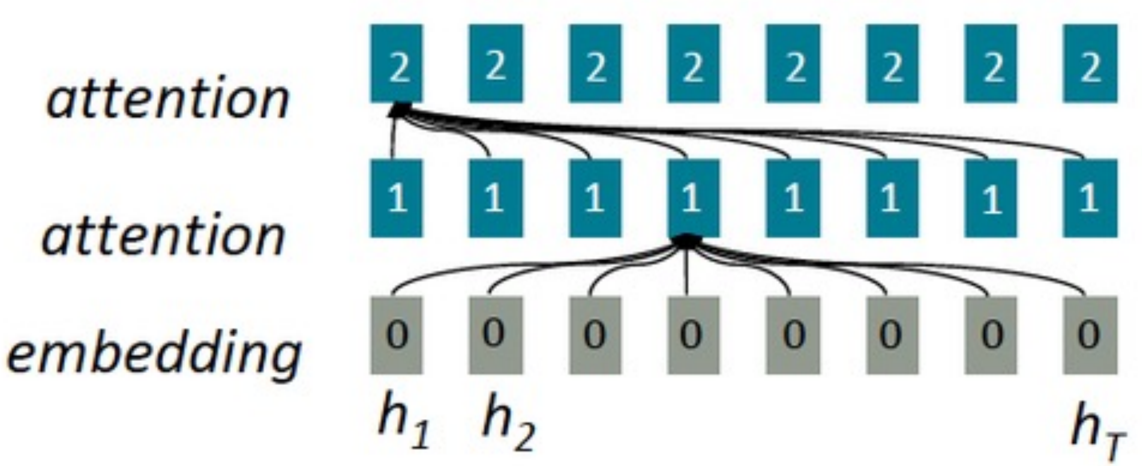


How?

Use the Attention Mechanism

Self Attention

Self Attention



All words attend to all words in previous layer; most arrows here are omitted

- The representation of each of the words is modified by every other word in the sequence by using the Self Attention mechanism
- The idea behind this architecture is that after several layers of Self Attention, each word develops a representation that takes into account all the other words that exist in the sentence.

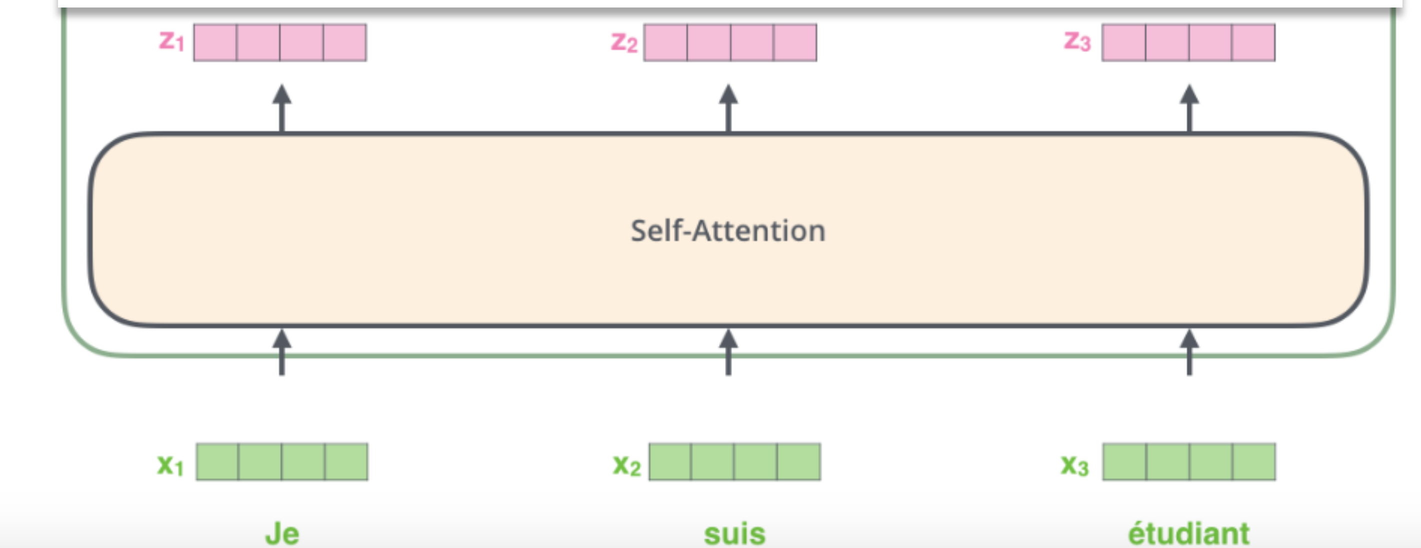
Self Attention

- Meaning of a word is context specific:
  - Example: Mark a Date vs Going on a date vs buying date at the market
- A Smart Embedding technique would provide a different vector representation for a word depending upon the other words surrounding it

- Self Attention: A way to make Word Representations <u>Context Aware</u>
- It does so by modulating the the word representation by using the representations of other words in the sentence.

Self Attention (cont)

How to Compute the Self Attention Scores?



Computing Self Attention Scores

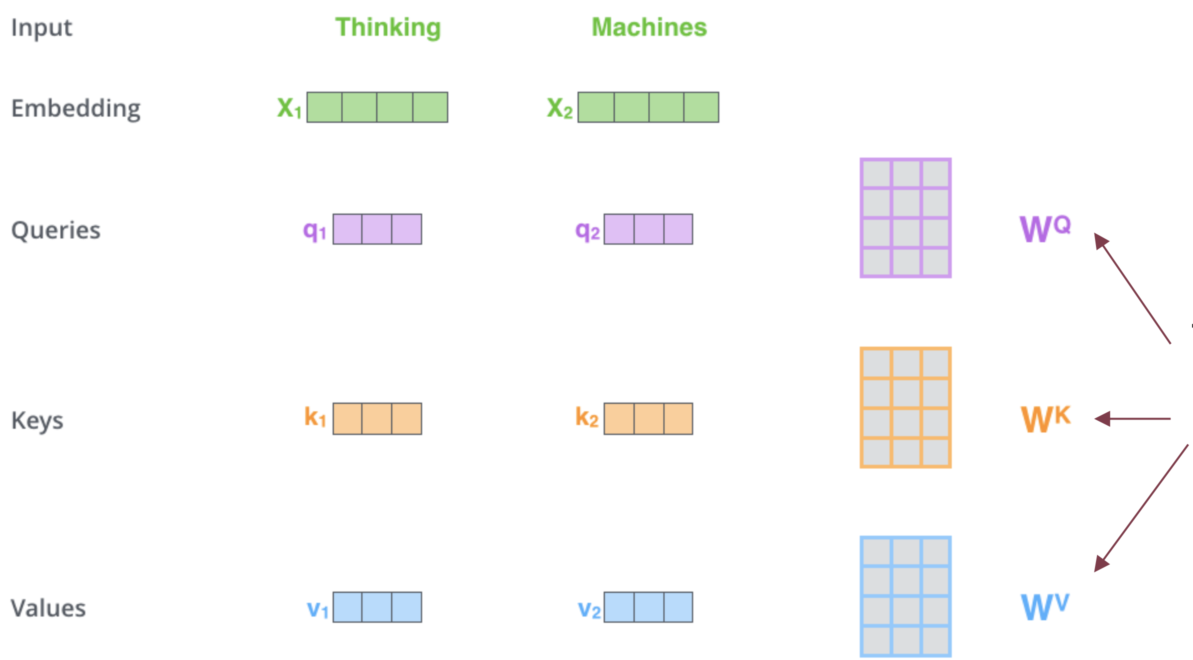
- Simplest way:

$$a_{ik} = x_i \circ x_{k,}\, 1 \leq k \leq n$$
$$w_{ik} = softmax_k(a_{ik})$$

This technique does not involve any learning since there are no parameters

Generalized Self Attention

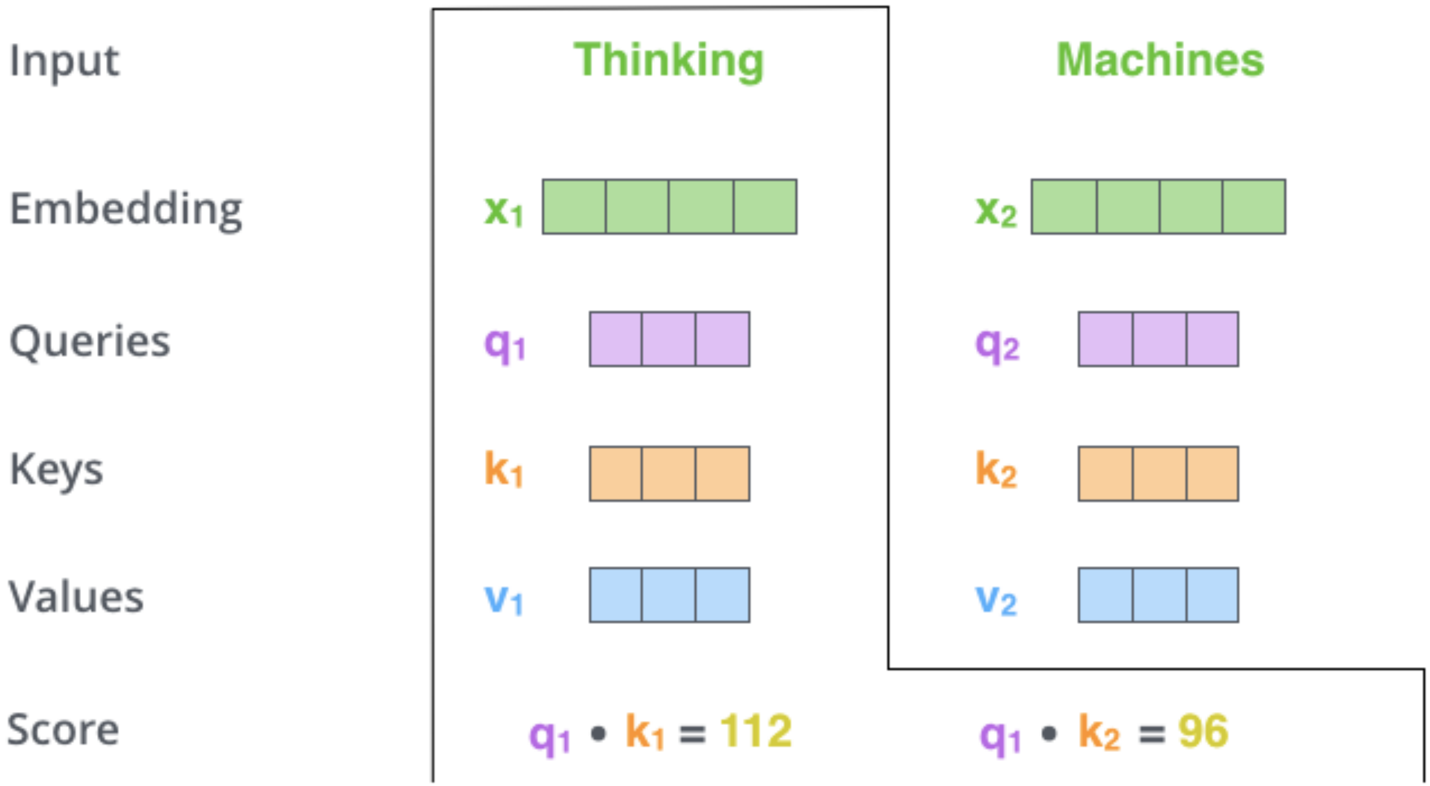
- The initial word embeddings are sent through 3 independent sets of dense projections, resulting in 3 separate vectors (per word)



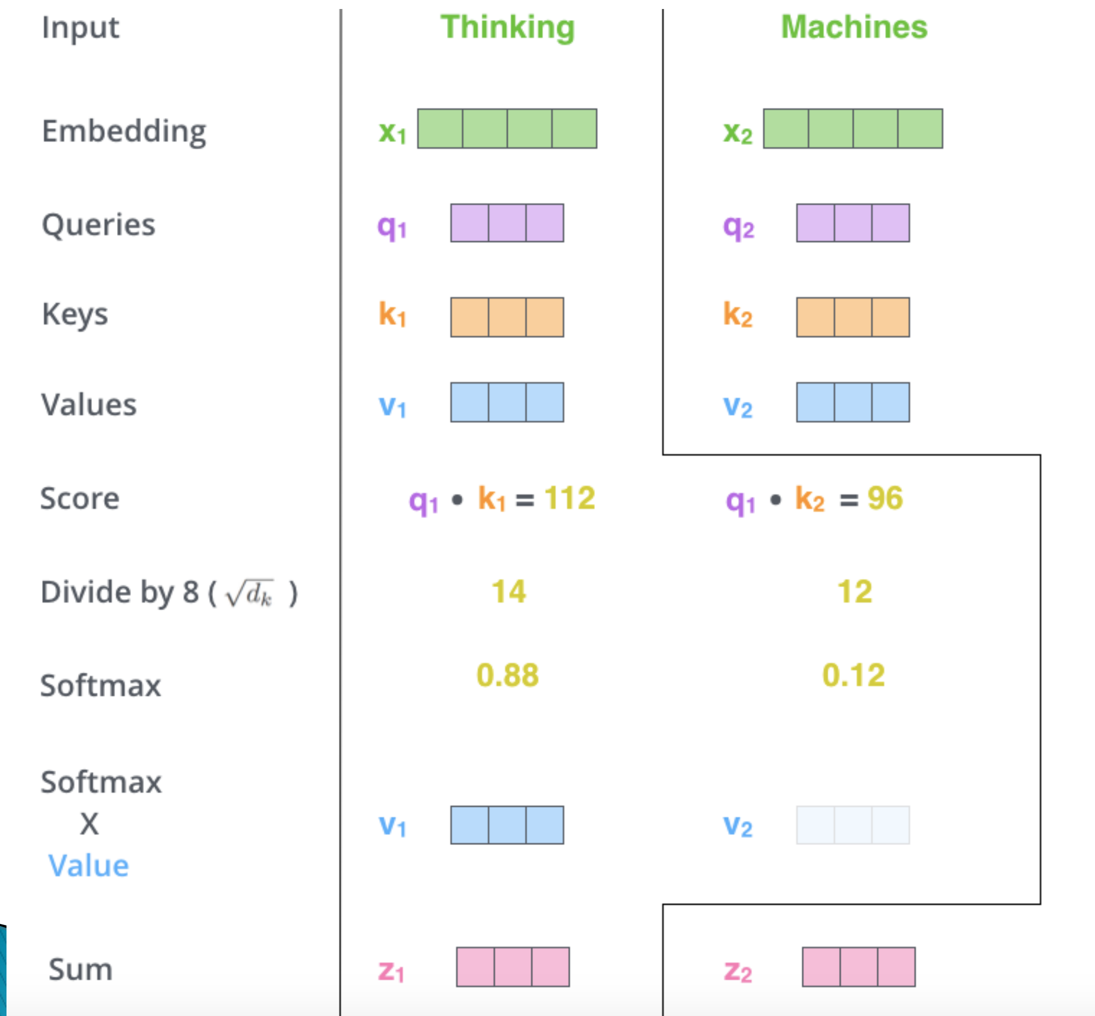
These matrices are learnable!

Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Generalized Self Attention (cont)



Generalized Self Attention (cont)

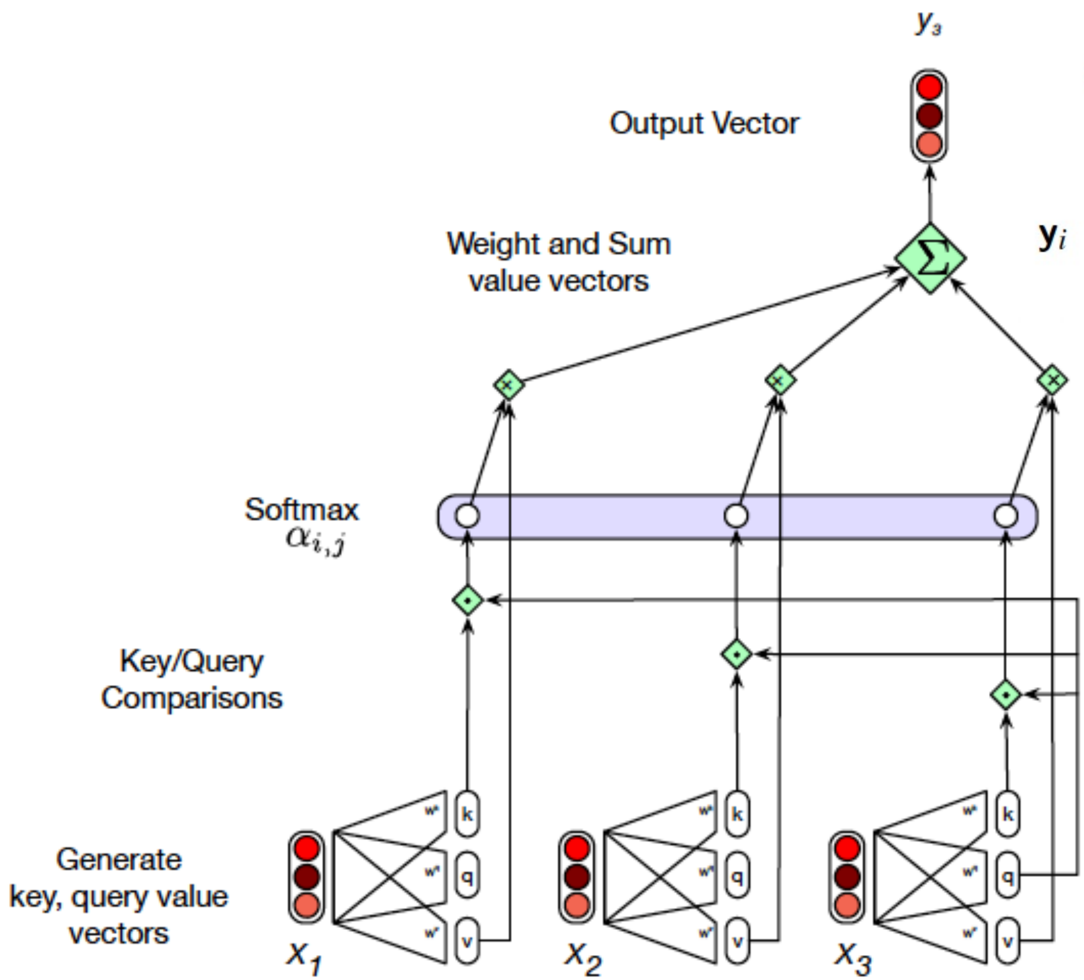


$Q = W^Q X; \ K = W^K X; \ V = W^V X$

Summarized as:

$$Z = softmax(\frac{QK^T}{\sqrt{d}})V$$

Generalized Self Attention (cont)



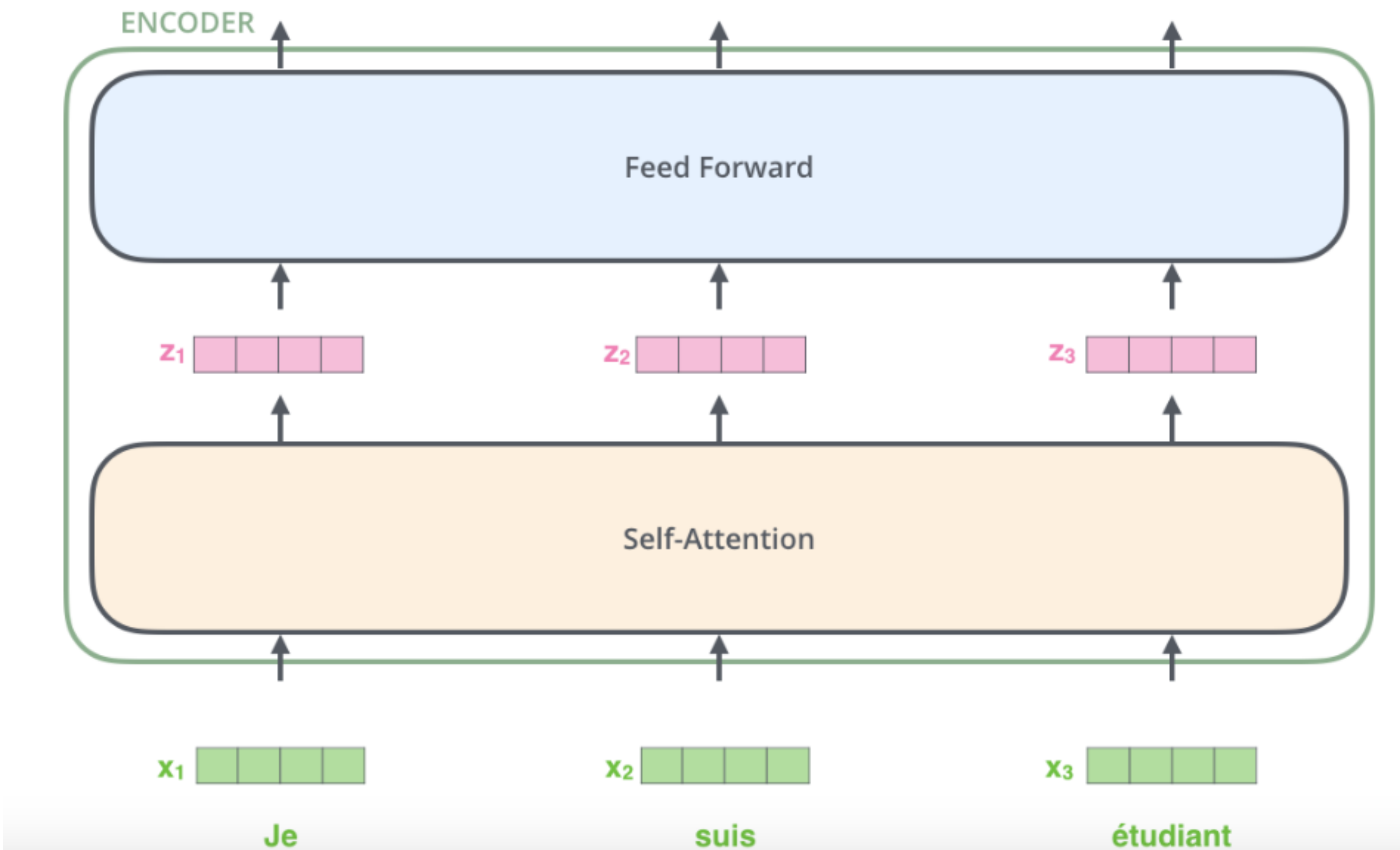
$$\mathbf{q}_i = \mathbf{W}^{\mathsf{Q}}\mathbf{x}_i; \;\; \mathbf{k}_i = \mathbf{W}^{\mathsf{K}}\mathbf{x}_i; \;\; \mathbf{v}_i = \mathbf{W}^{\mathsf{V}}\mathbf{x}_i$$

$$\mathbf{y}_i = \sum_{j \le i} \alpha_{ij}\mathbf{v}_j$$

$$\text{score}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{q}_i \cdot \mathbf{k}_j}{\sqrt{d_k}}$$

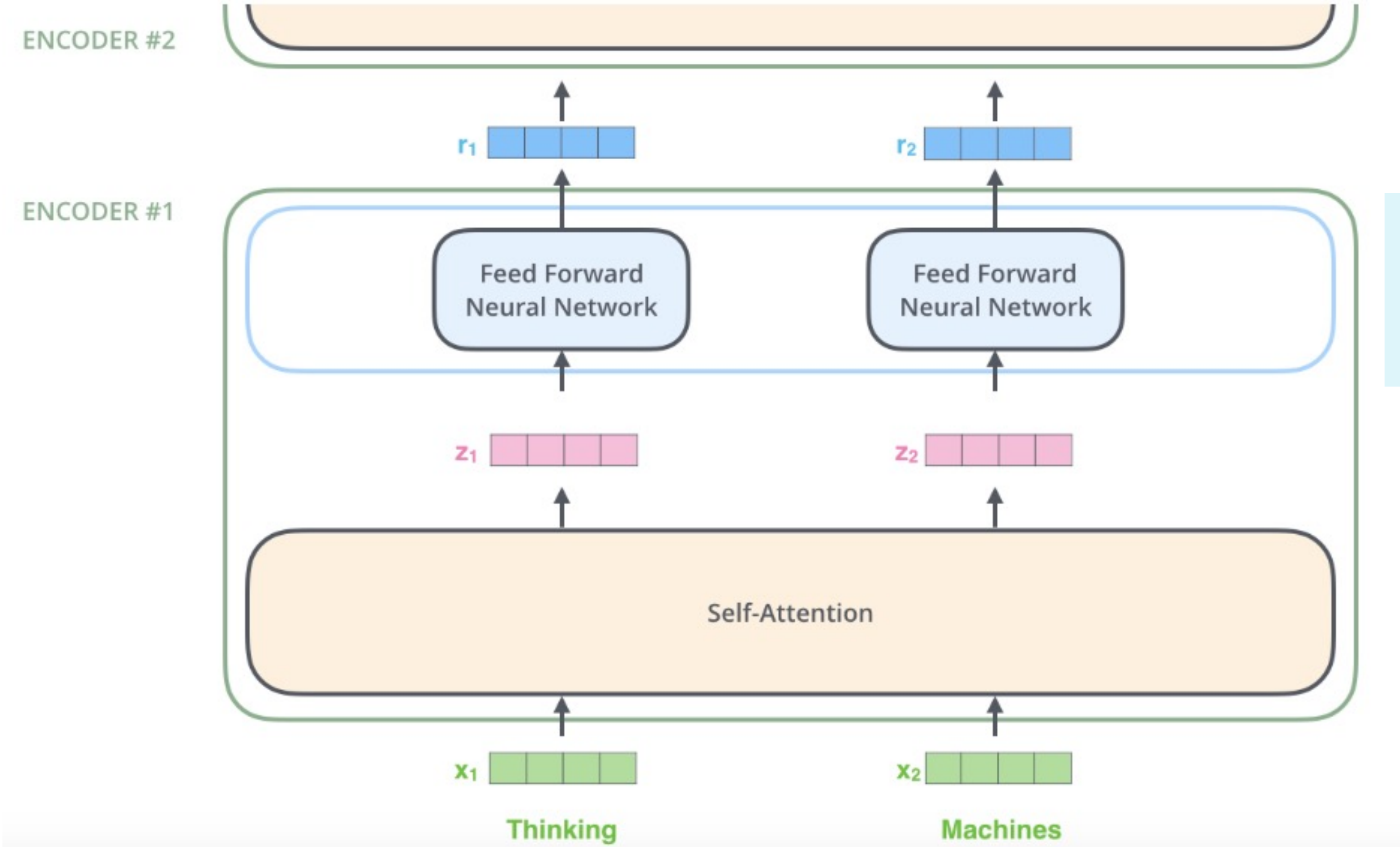
Transformer Encoders

Transformers – Encoders



Here we begin to see one key property of the Transformer, which is that the word in each position flows through its own path in the encoder. There are dependencies between these paths in the self-attention layer. The feed-forward layer does not have those dependencies, however, and thus the various paths can be executed in parallel while flowing through the feed-forward layer.

Transformers – Encoders

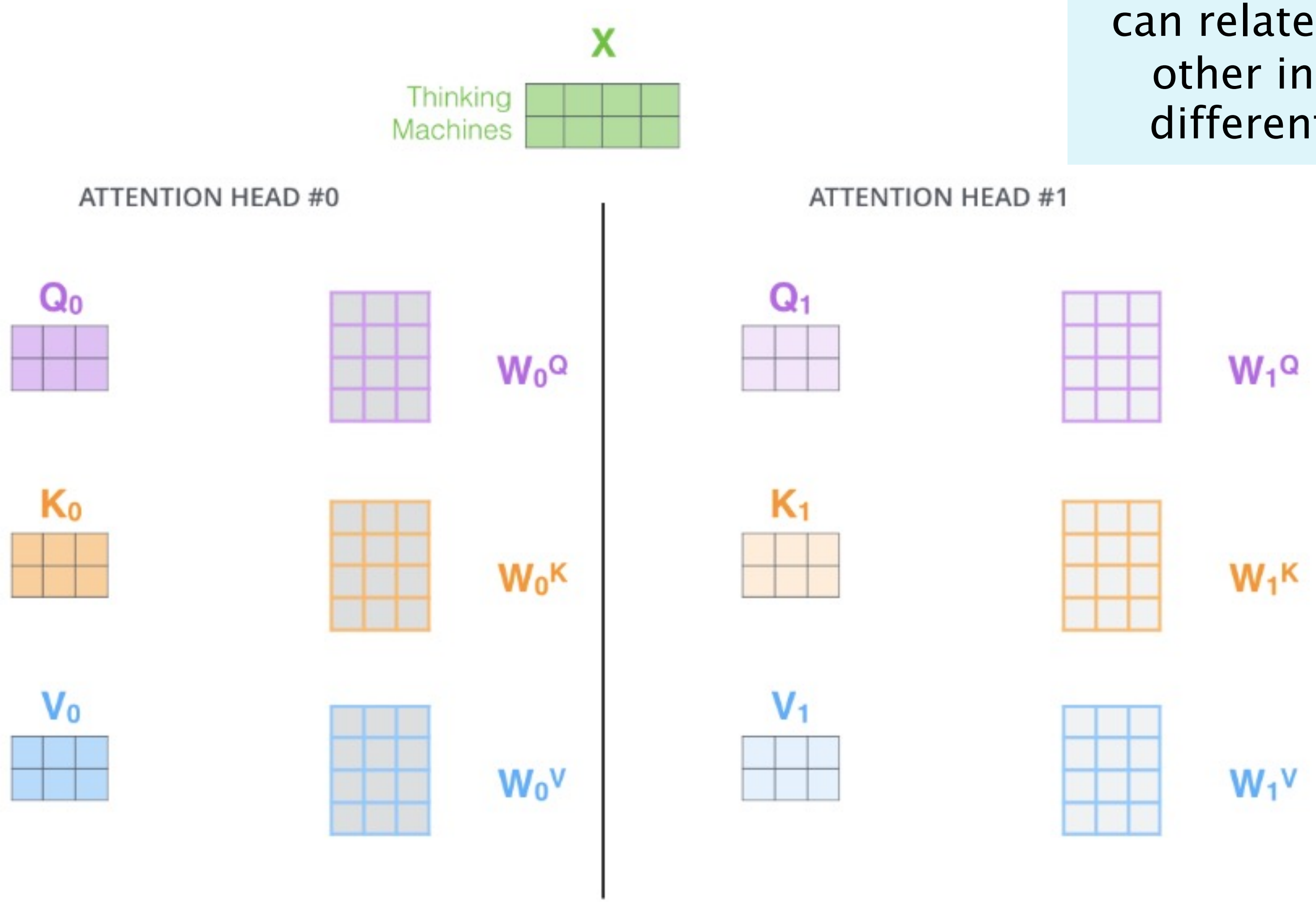


Both these networks have the same parameters

The word at each position passes through a self-attention process. Then, they each pass through a feed-forward neural network -- the exact same network with each vector flowing through it separately.

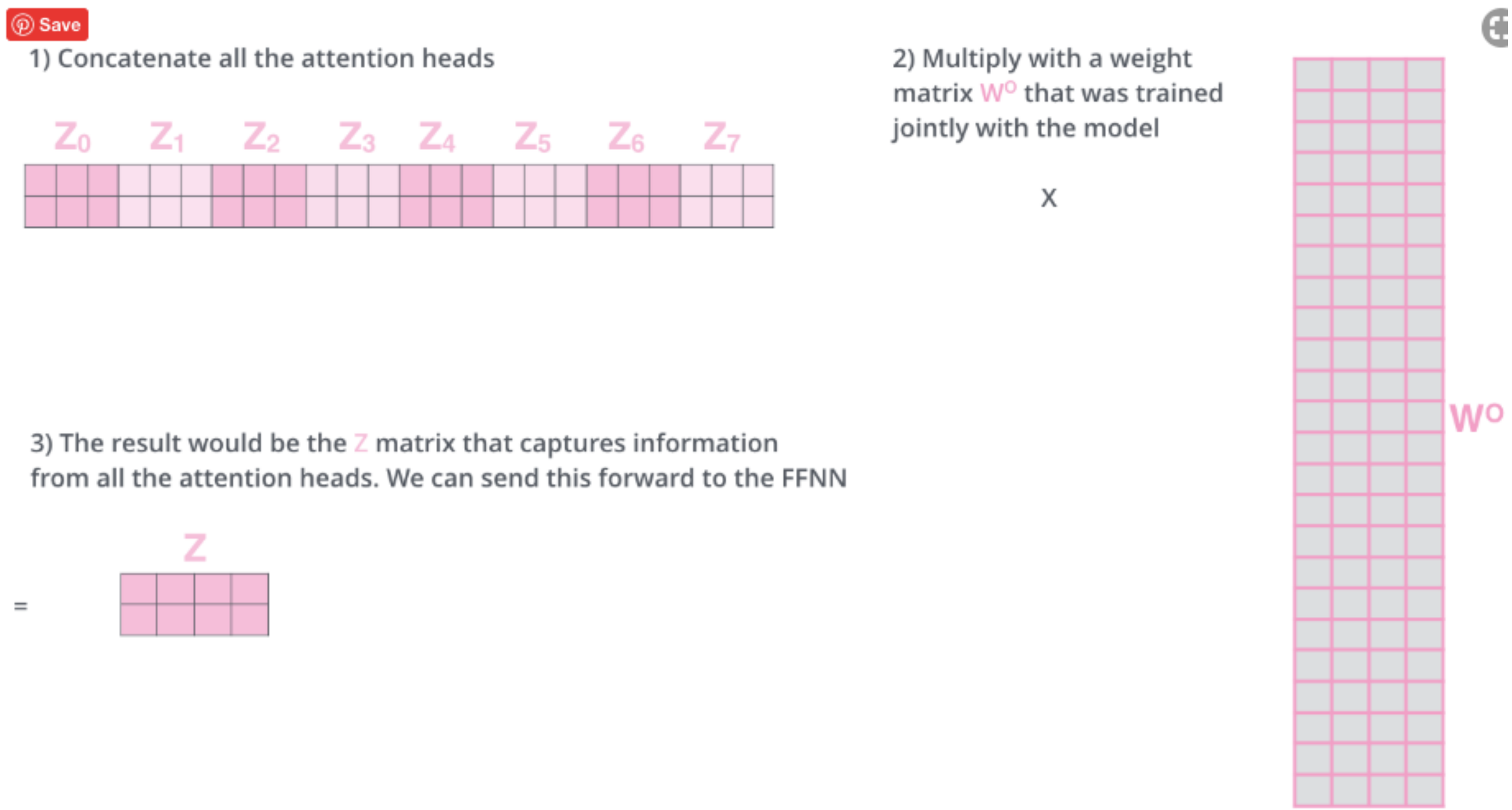
Multi-Headed Attention

Words in a sentence can relate to each other in many different ways

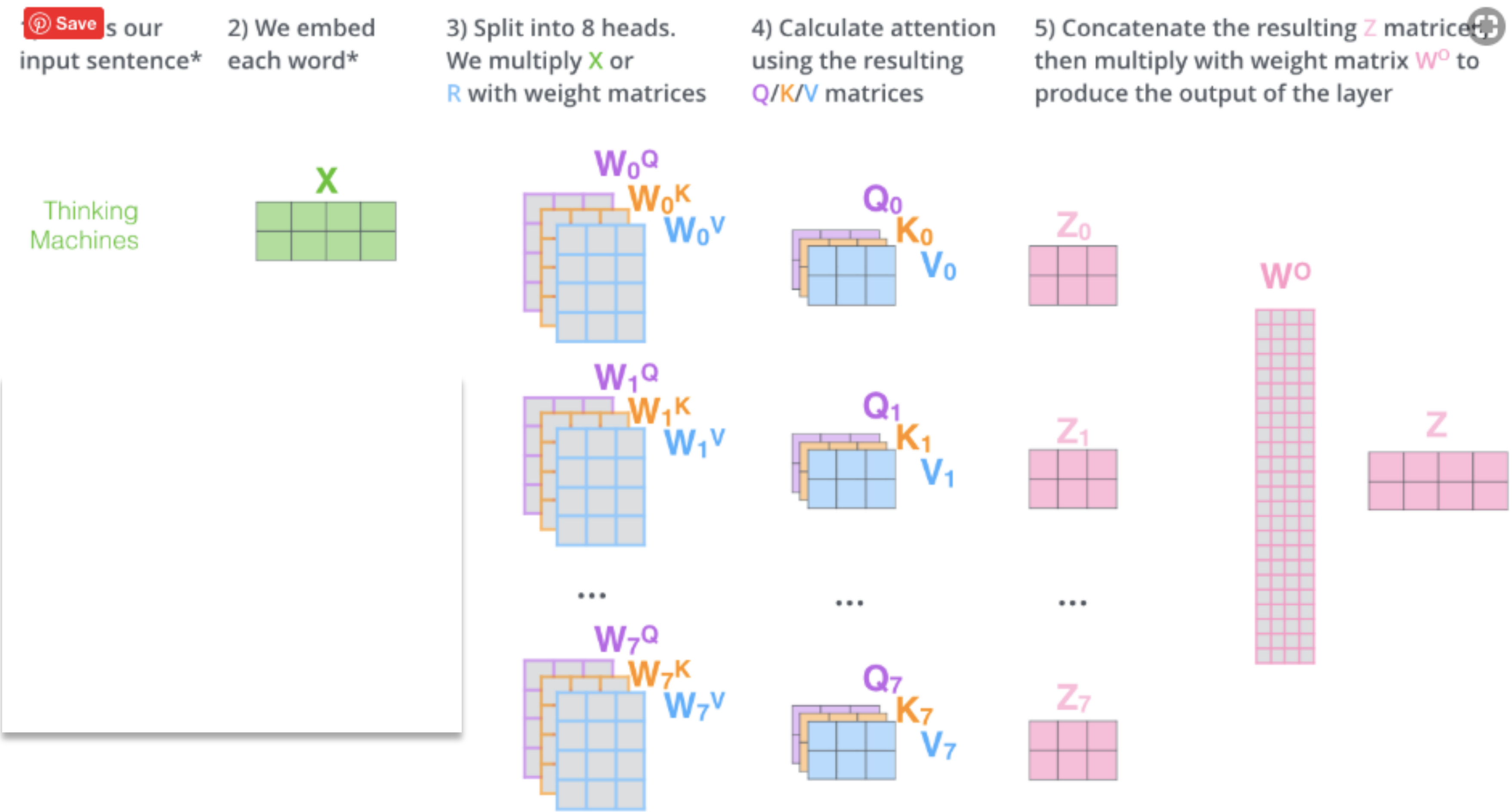


With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

Multi-Headed Attention

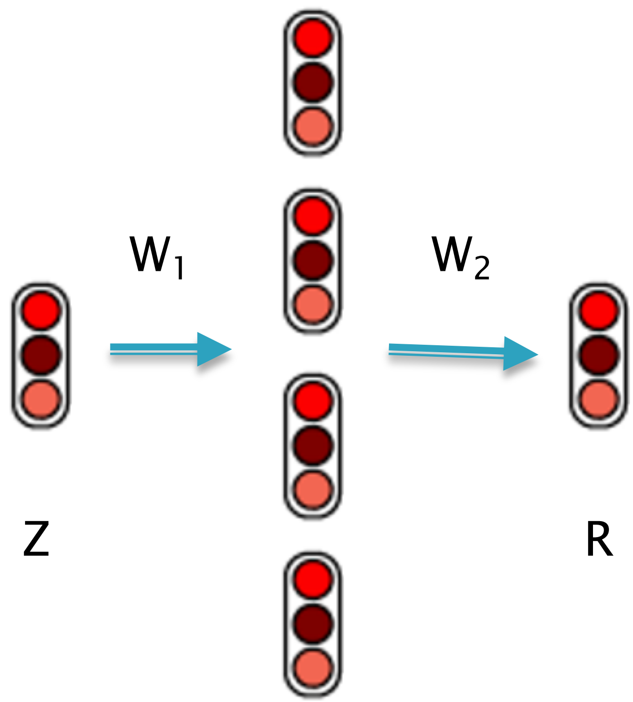


Attention – Summary



Dense Feed Forward Layer

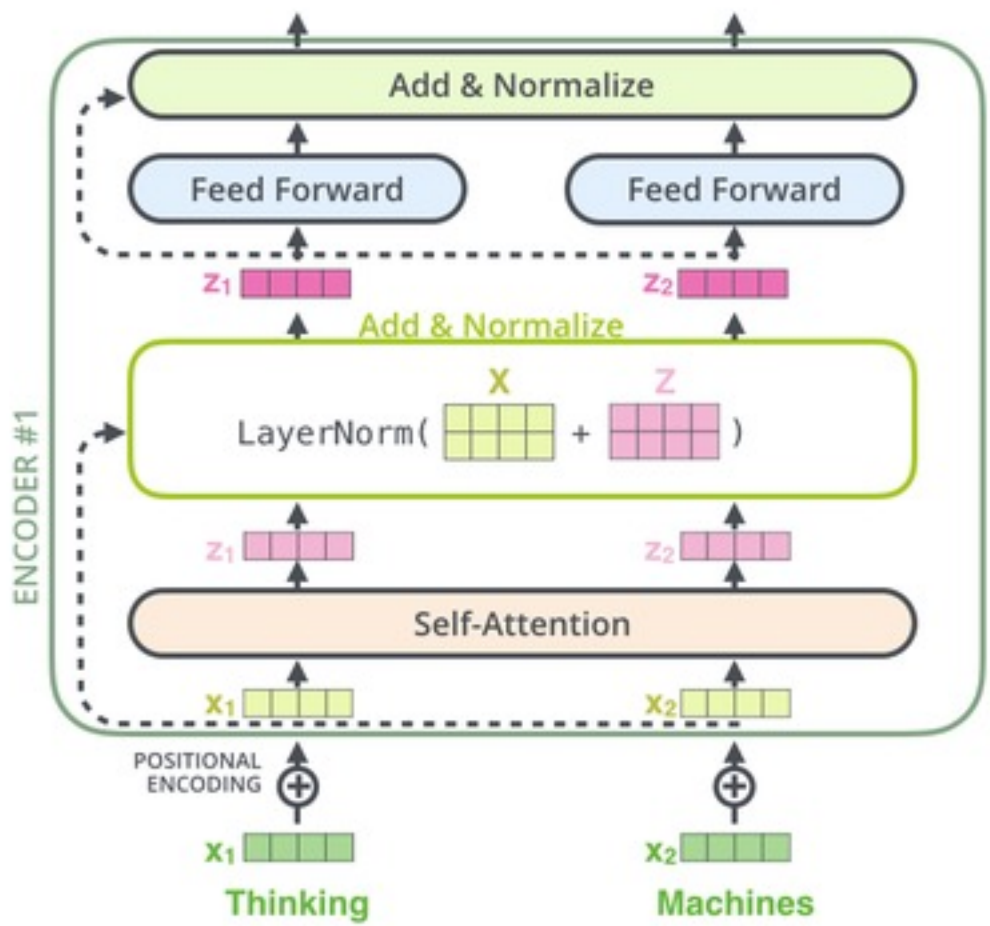
$$R_i = ReLU(Z_i W_1 + b_1) W_2 + b_2, \quad i = 1, \ldots, N$$



Size d Size 4d Size d

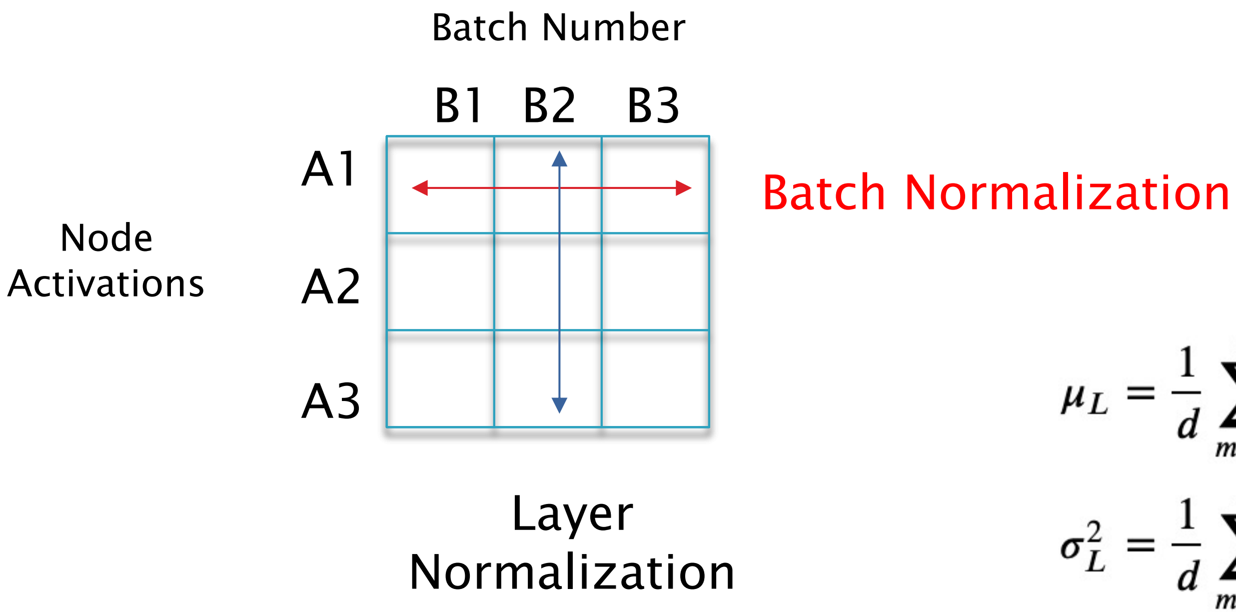
Add Residual Connections and Layer Normalization

A Single Layer



Layer Normalization

- Batch Normalization vs Layer Normalization



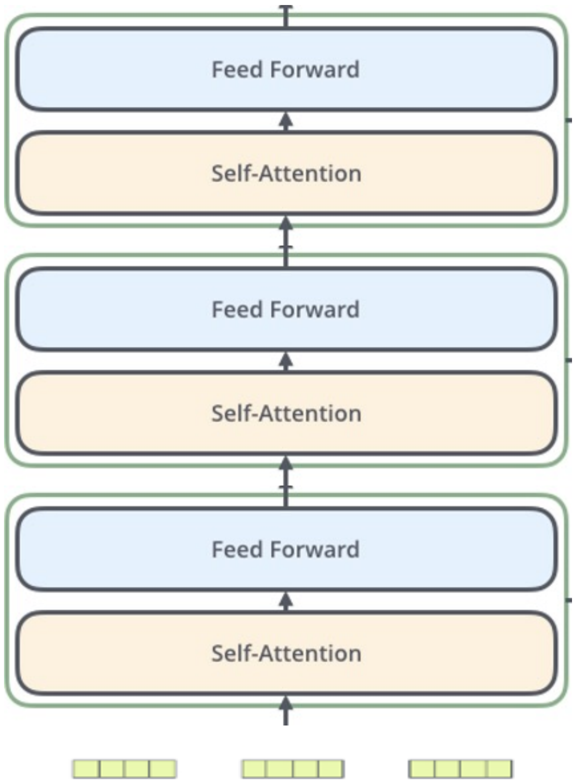
$$\mu_L = \frac{1}{d}\sum_{m=1}^{d} a(m)$$

$$\sigma_L^2 = \frac{1}{d}\sum_{m=1}^{d} (a(m) - \mu_L)^2$$

$$\hat{a}(m) = \frac{a(m) - \mu_L}{\sqrt{\sigma_L^2 + \epsilon}}$$

$$c(m) = \gamma \hat{a}(m) + \beta$$

Multiple Layers



$(W_3^Q, W_3^K, W_3^V), W_3^{FF}$

$(W_2^Q, W_2^K, W_2^V), W_2^{FF}$

$(W_1^Q, W_1^K, W_1^V), W_1^{FF}$

Parameters not shared Between layers

Modularity: All elements of input sequence share the same parameters: RNN like property

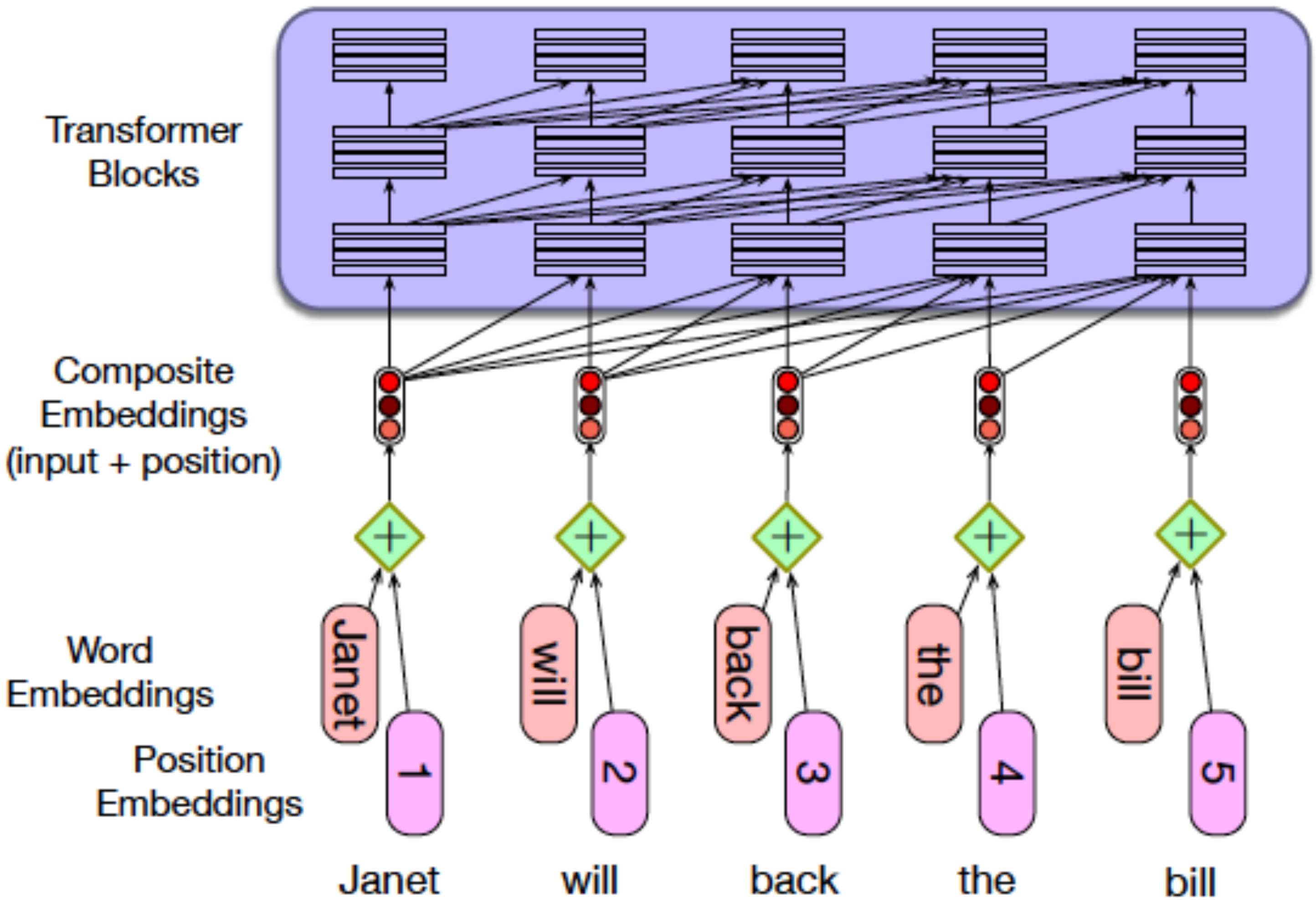
Counting Number of Parameters

	A	B	C	D	E	F	G	H	I	J
1	**Parameters in the Self Attention Block**				# Parms		Embedding Dimension			32
2							Number of Attention Heads			1
3	W^Q			32 x 32 + 32	1,056		Dense Dimension			32
4	W^K			32 x 32 + 32	1,056		Number of Blocks			1
5	W^V			32 x 32 + 32	1,056		Sequence Length			600
6										
7	Sub Total				3,168					
8										
9	**Number of Attention Heads**				2					
10										
11	Sub Total				6,336					
12										
13	**Parameters in the Projection Block**			2 X 32 X 32 + 32	2,080					
14										
15	Sub Total				8,416					
16										
17	**Parameters in Dense Feed Forward Block**									
18										
19	DFN1			32 X 32 + 32	1,056					
20	DFN2			32 X 32 + 32	1,056					
21										
22	Sub Total				10,528					
23										
24	**Layer Normalization 1**			32 X 2	64					
25										
26	**Layer Normalization 2**			32 X 2	64					
27										
28	**Grant Total**				**10,656**					
29										
30										
31	**Self Attention Computations**									
32										
33	For a single element			32 X 32 x 600	614,400					
34	For Entire Sequence			( ) X 600	368,640,000					
35										

Positional Encoding

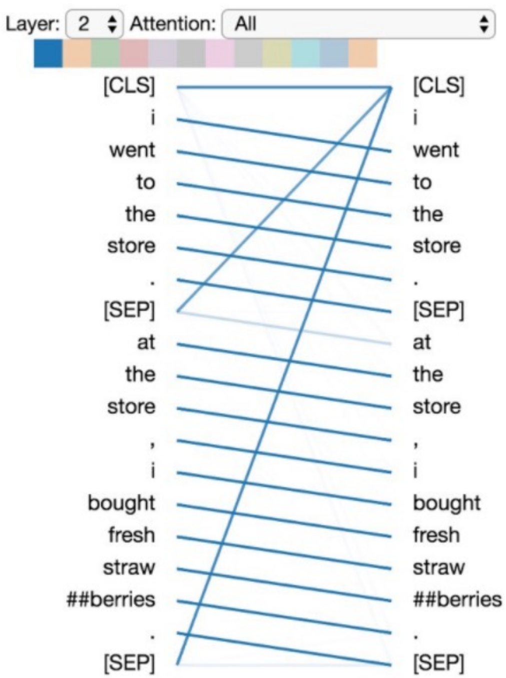
- Positional information arises naturally in RNN/LSTMs
- Transformer Architecture is invariant to permutations of the input sequence
- This is a problem if the position is important
  Example: NLP

Positional Encoding



Visualizing Attention in Transformers

(a)



Attention to Next Word
Similar to a Backwards RNN

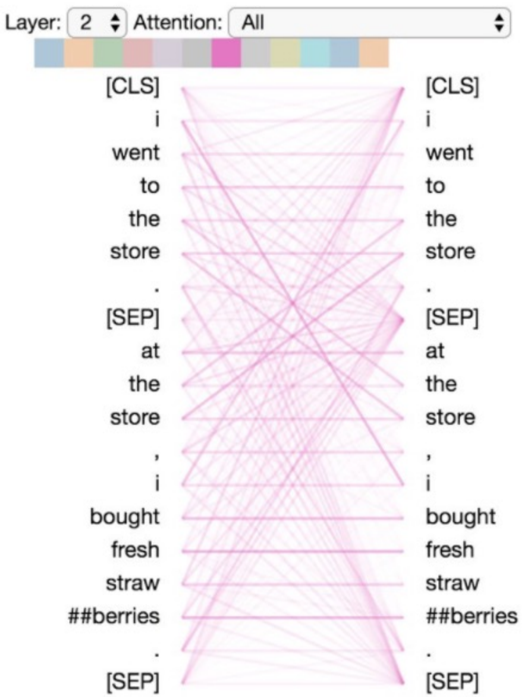
(b)

Attention to Prior Word
Similar to a Forward RNN

https://towardsdatascience.com/deconstructing-bert-distilling-6-patterns-from-100-million-parameters-b49113672f77

Visualizing Attention

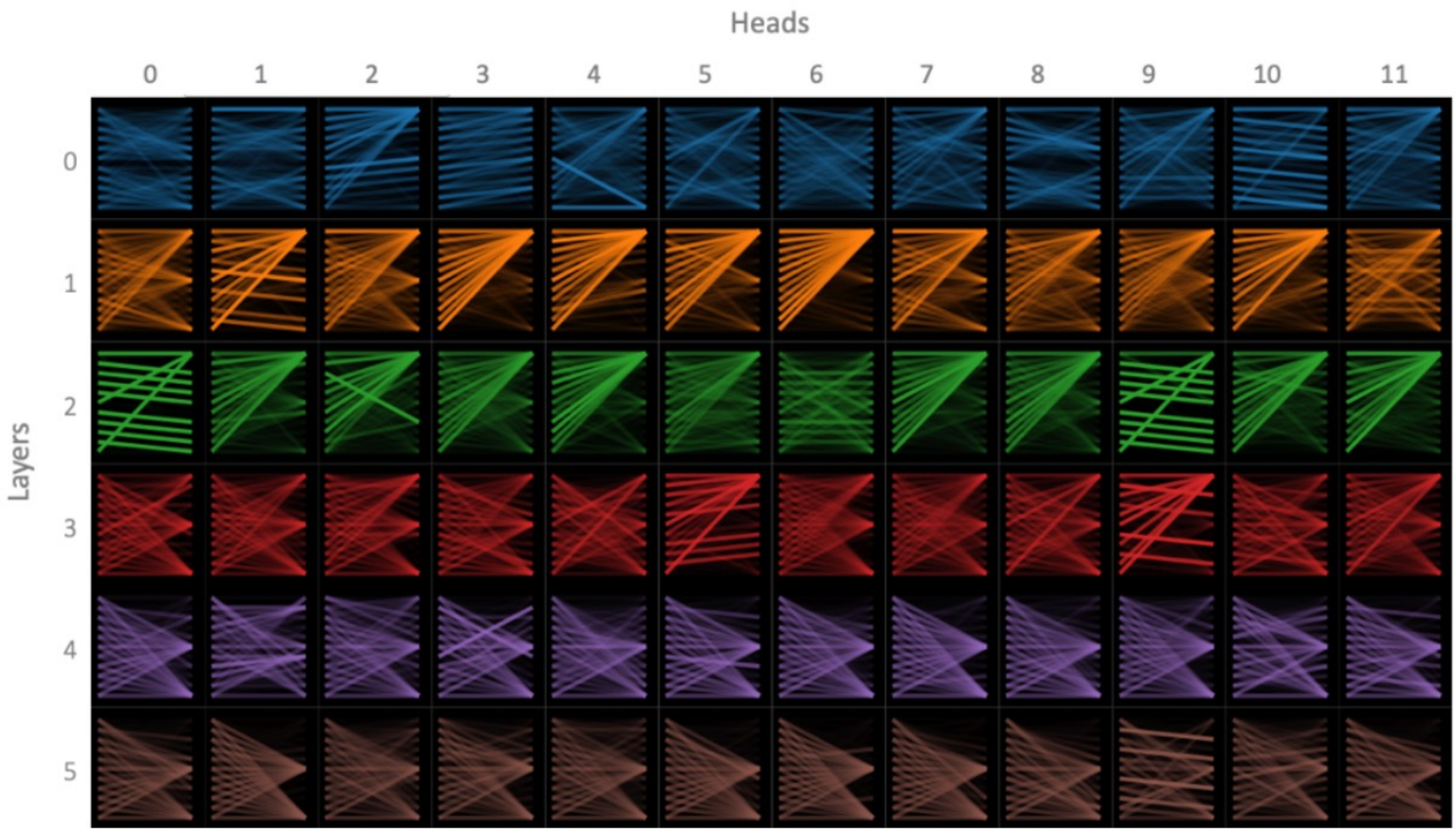
(c)



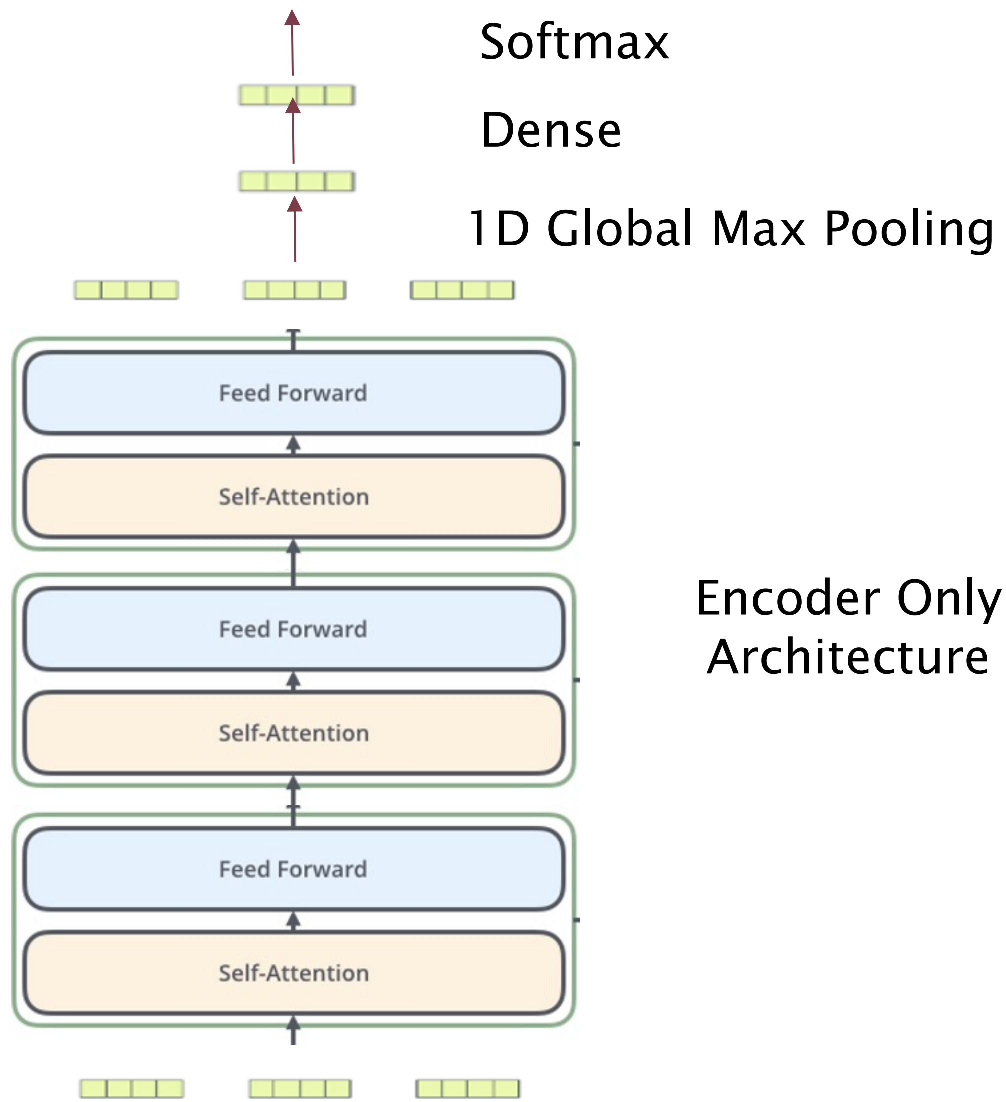
(d)

Attention to Similar Words

Visualizing Attention



Classification using Transformers



Further Reading

- Das and Varma: Chapter Transformers
- Chollet (2nd Edition): Chapter 11, Section11.4
- http://jalammar.github.io/illustrated-transformer/