

# Lecture 05: Self-Attention & Transformer overview

**Anastasia Ianina**

Harbour.Space University  
16.06.2021

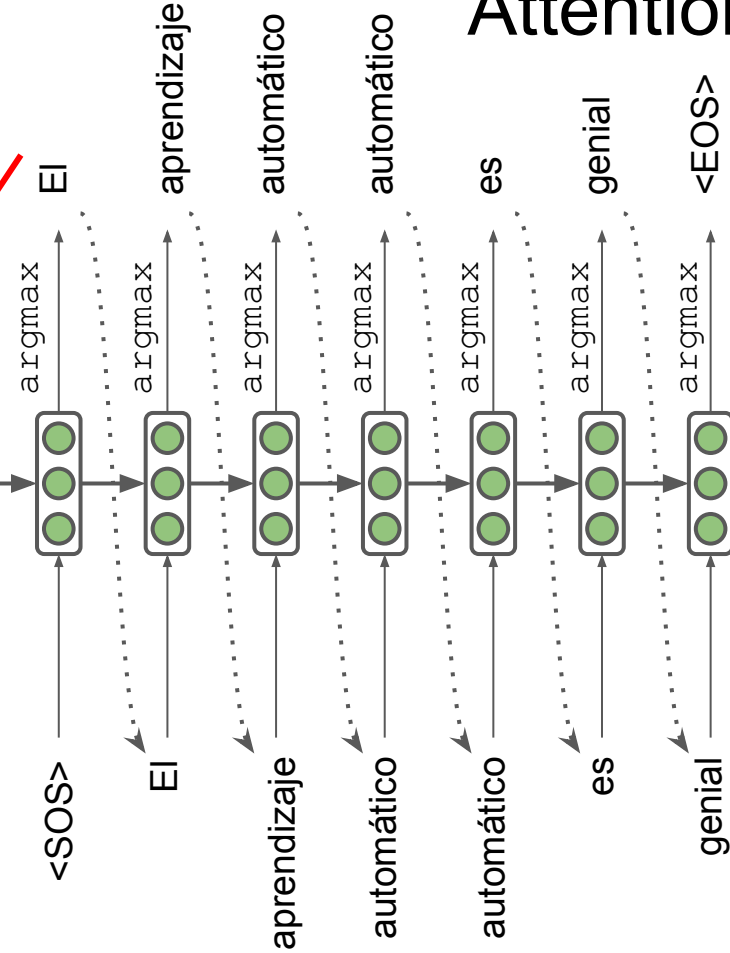
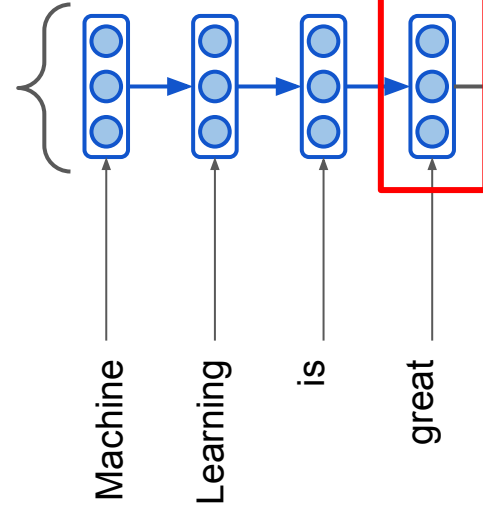
1. Recap: Attention in seq2seq
2. Transformer architecture
3. Self-Attention
4. Positional encoding
5. Layer normalization
6. Decoder in Transformer

# Attention in seq2seq

This state encodes the whole sentence

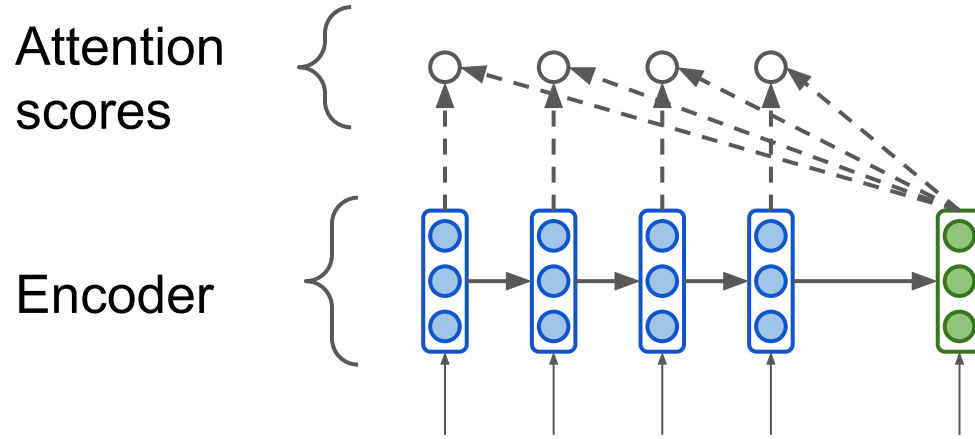
It is a bottleneck!

Encoder

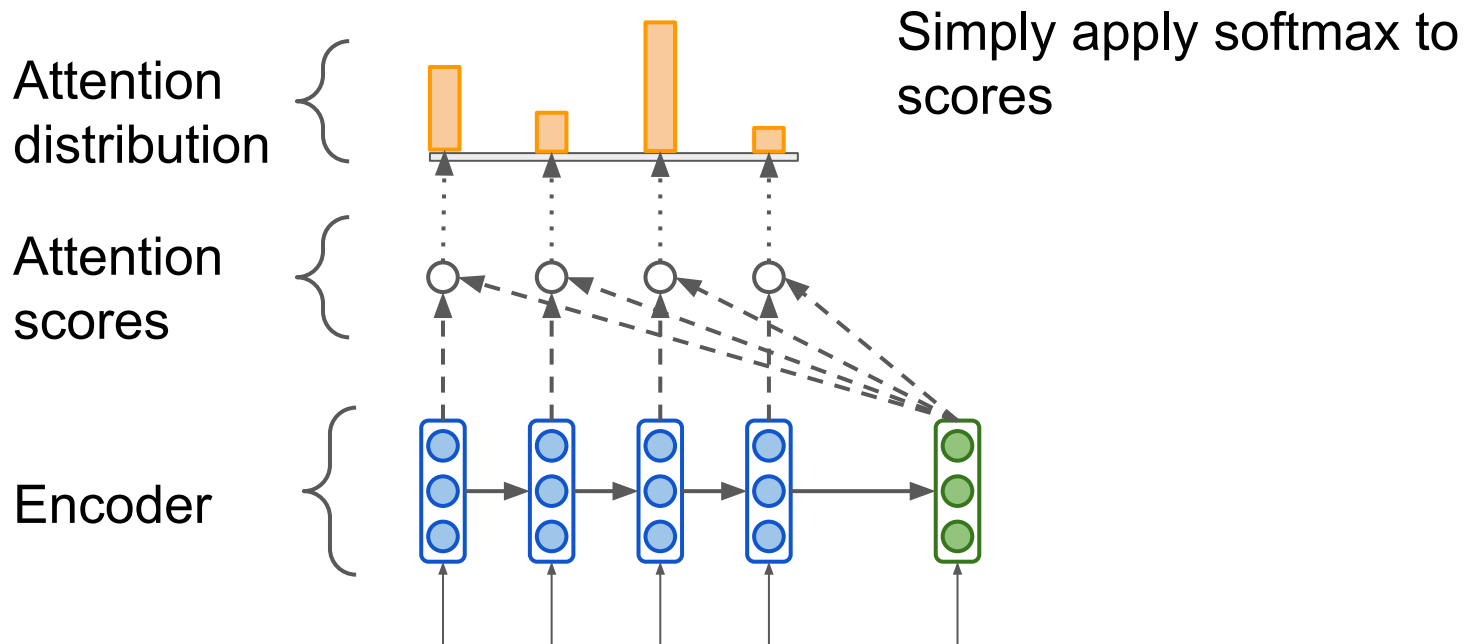


Decoder

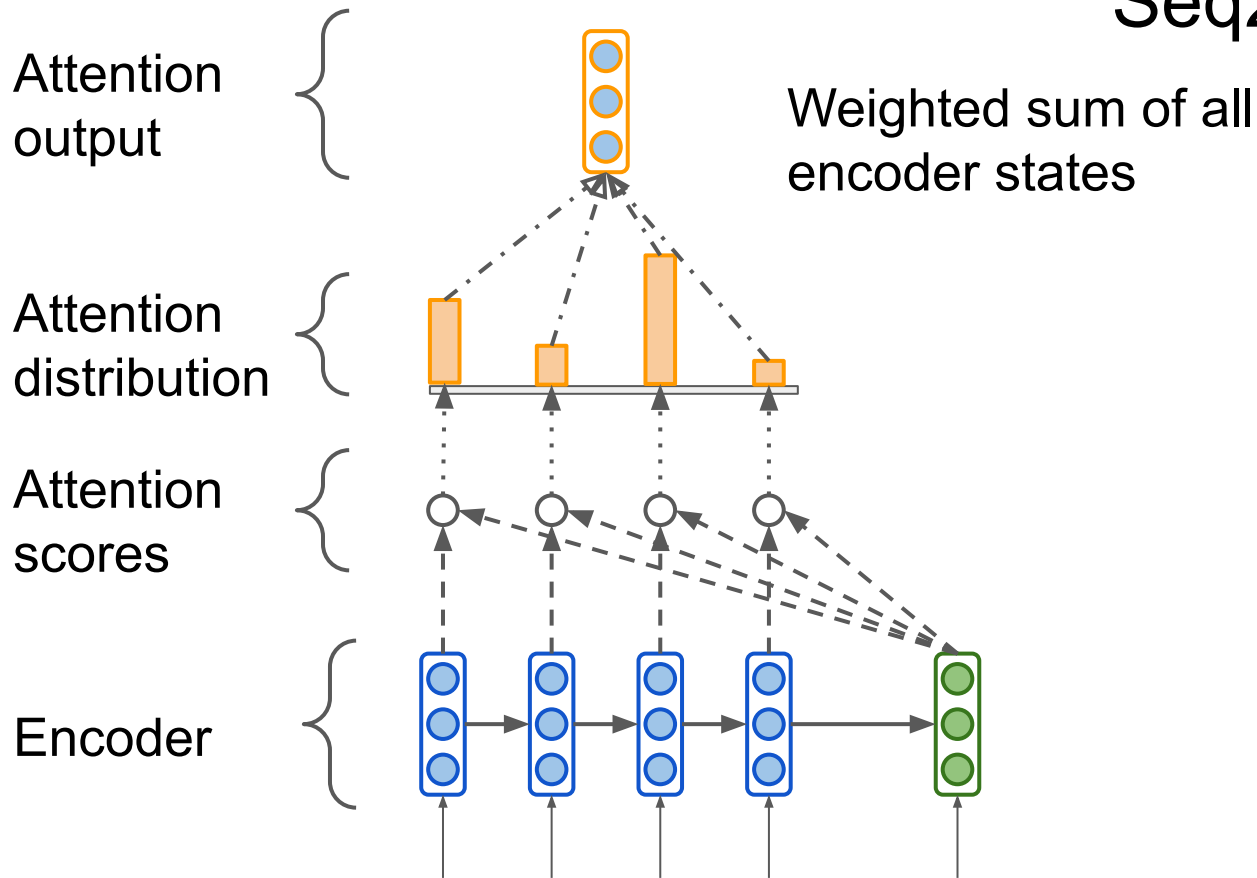
# Seq2seq with attention



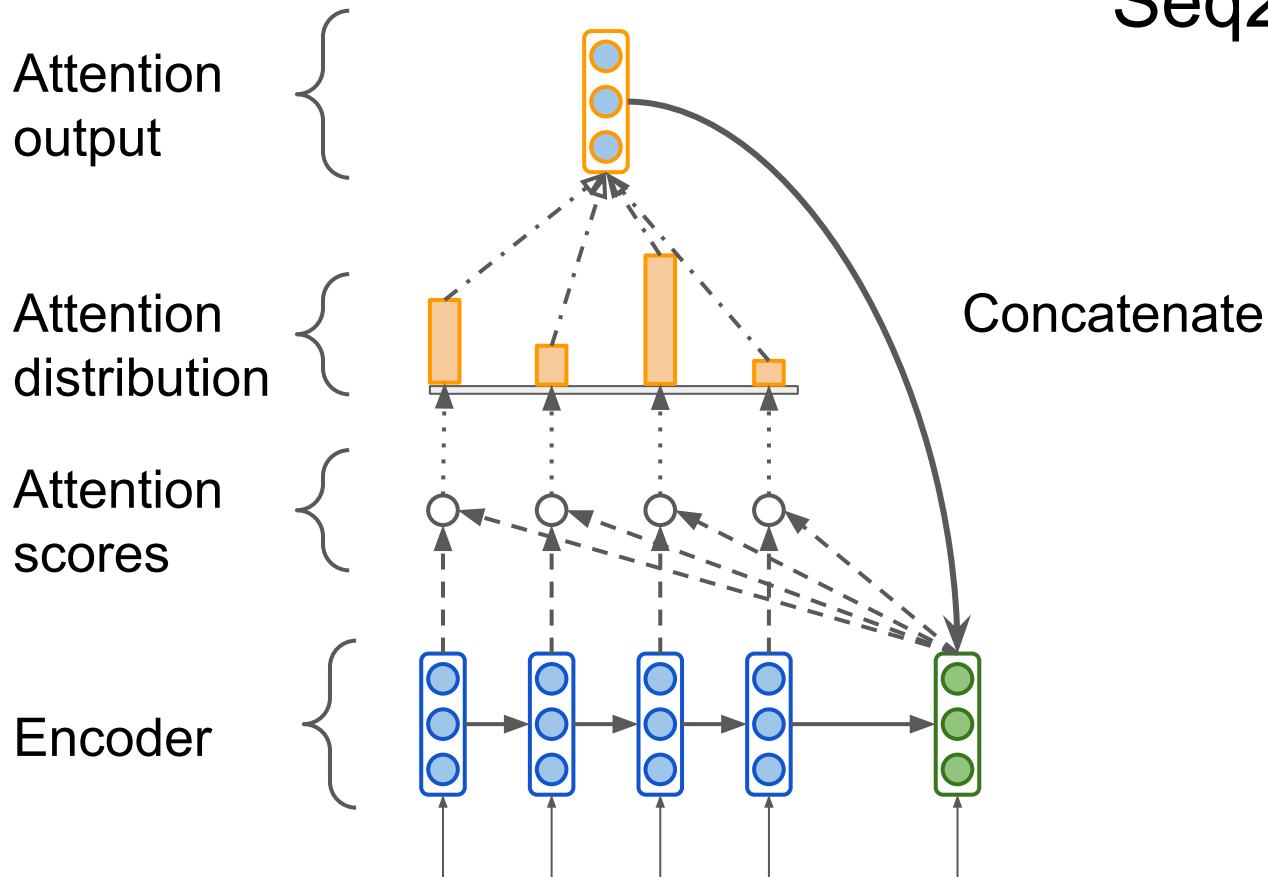
# Seq2seq with attention



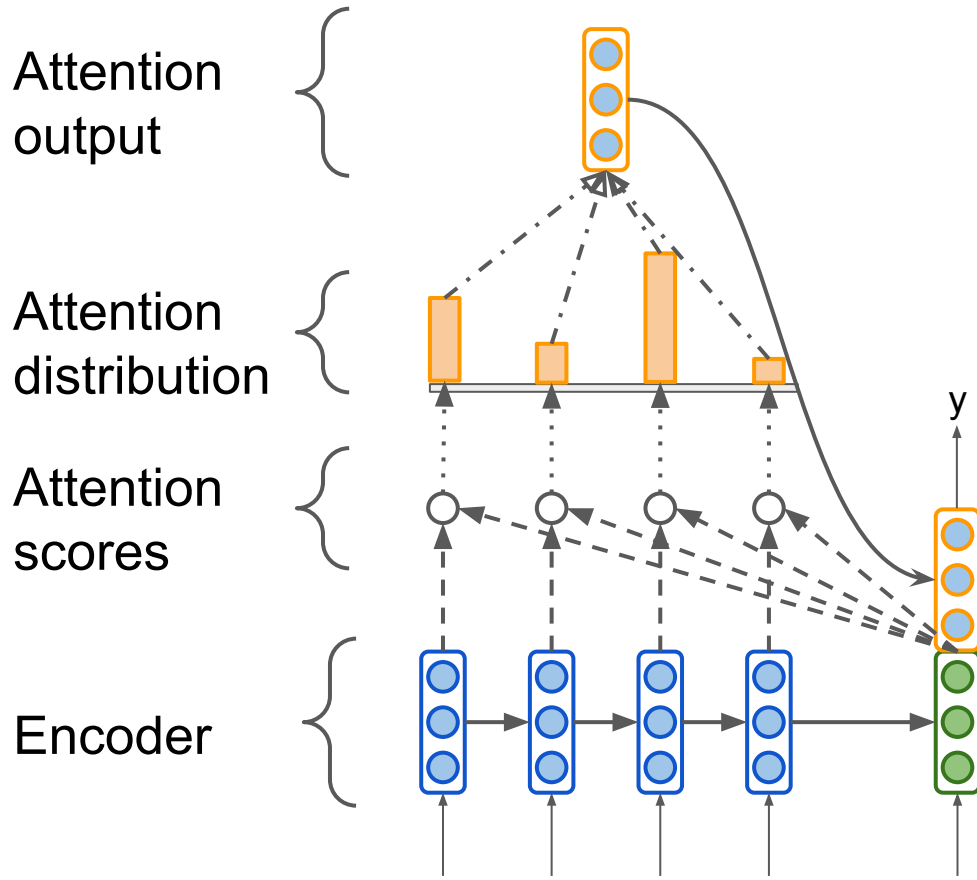
# Seq2seq with attention



# Seq2seq with attention

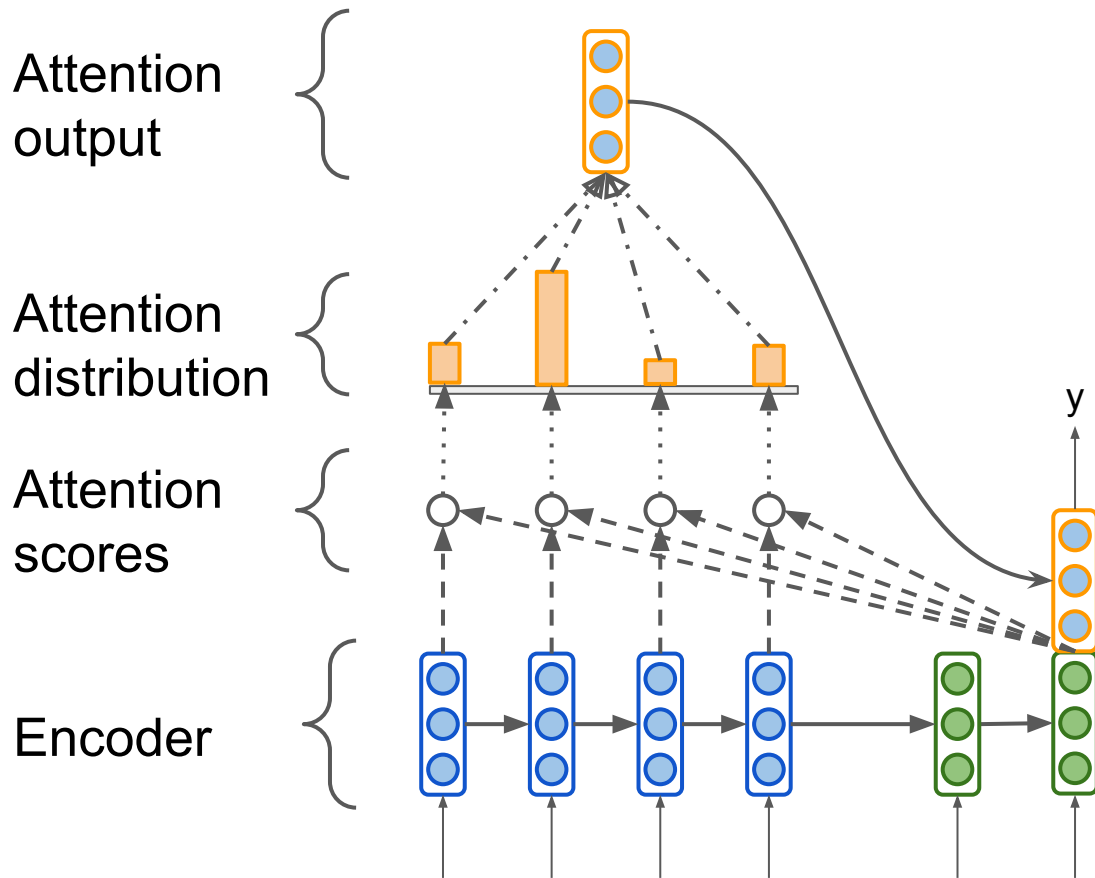


# Seq2seq with attention

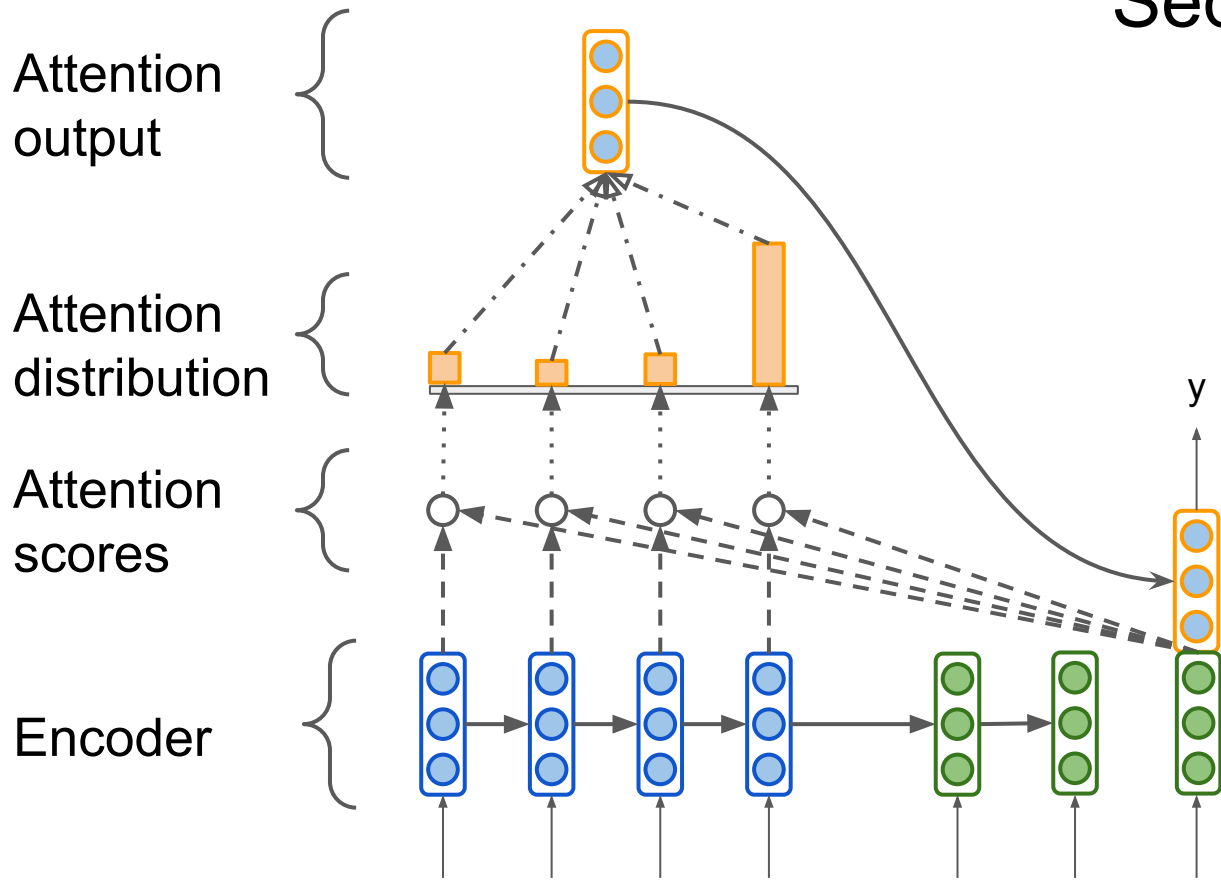




# Seq2seq with attention



# Seq2seq with attention



## Attention in equations

Denote encoder hidden states  $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^k$   
and decoder hidden state at time step  $t$   $\mathbf{s}_t \in \mathbb{R}^k$

The attention scores  $\mathbf{e}^t$  can be computed as dot product

$$\mathbf{e}^t = [\mathbf{s}^T \mathbf{h}_1, \dots, \mathbf{s}^T \mathbf{h}_N]$$

Then the attention vector is a linear combination of encoder states

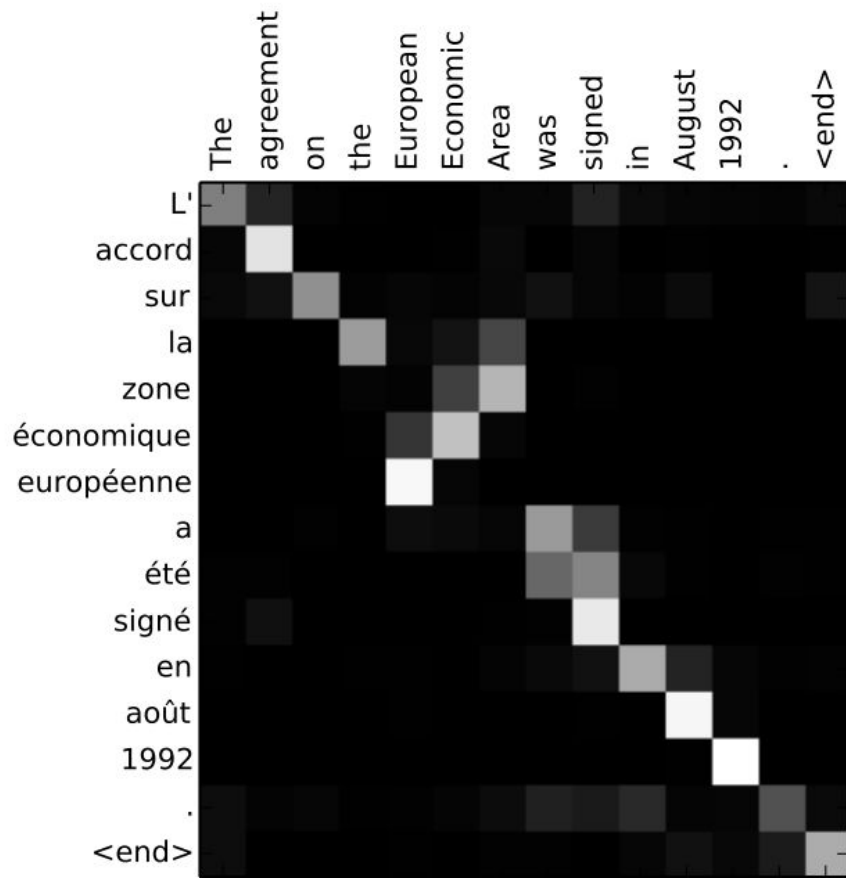
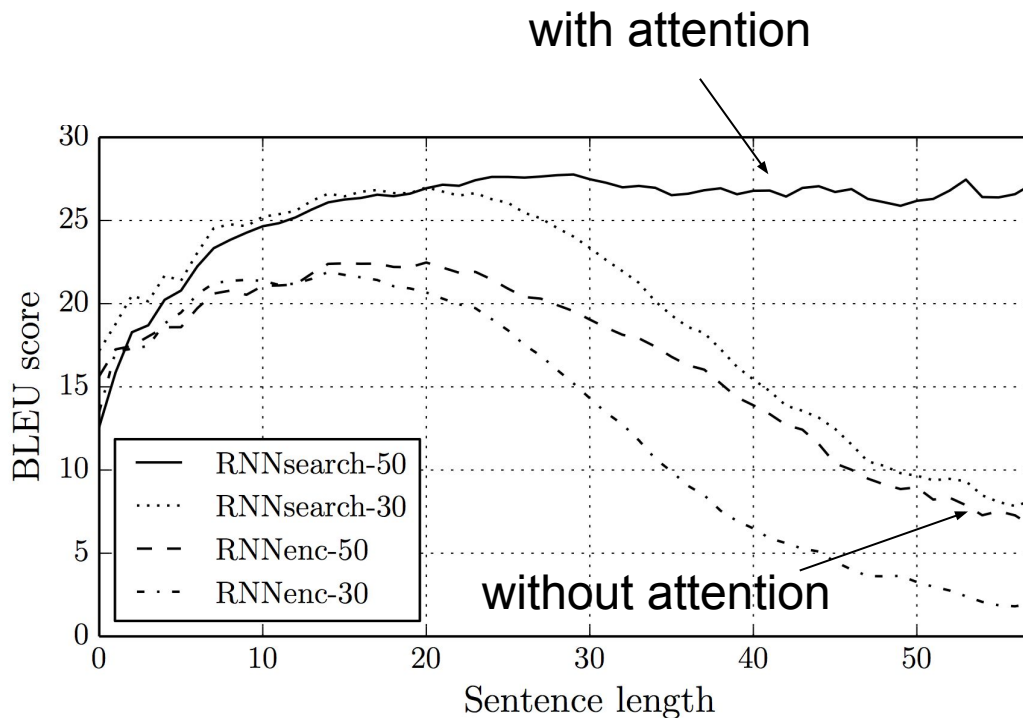
$$\mathbf{a}_t = \sum_{i=1}^N \alpha_i^t \mathbf{h}_i \in \mathbb{R}^k, \text{ where } \boldsymbol{\alpha}_t = \text{softmax}(\mathbf{e}_t)$$

# Attention variants

- Basic dot-product (the one discussed before):  $e_i = \mathbf{s}^T \mathbf{h}_i \in \mathbb{R}$
- Multiplicative attention:  $e_i = \mathbf{s}^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$ 
  - $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$  - weight matrix
- Additive attention:  $e_i = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{s}) \in \mathbb{R}$ 
  - $\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}, \mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}$  - weight matrices
  - $\mathbf{v} \in \mathbb{R}^{d_3}$  - weight vector

# Attention advantages

- “Free” word alignment
- Better results on long sequences

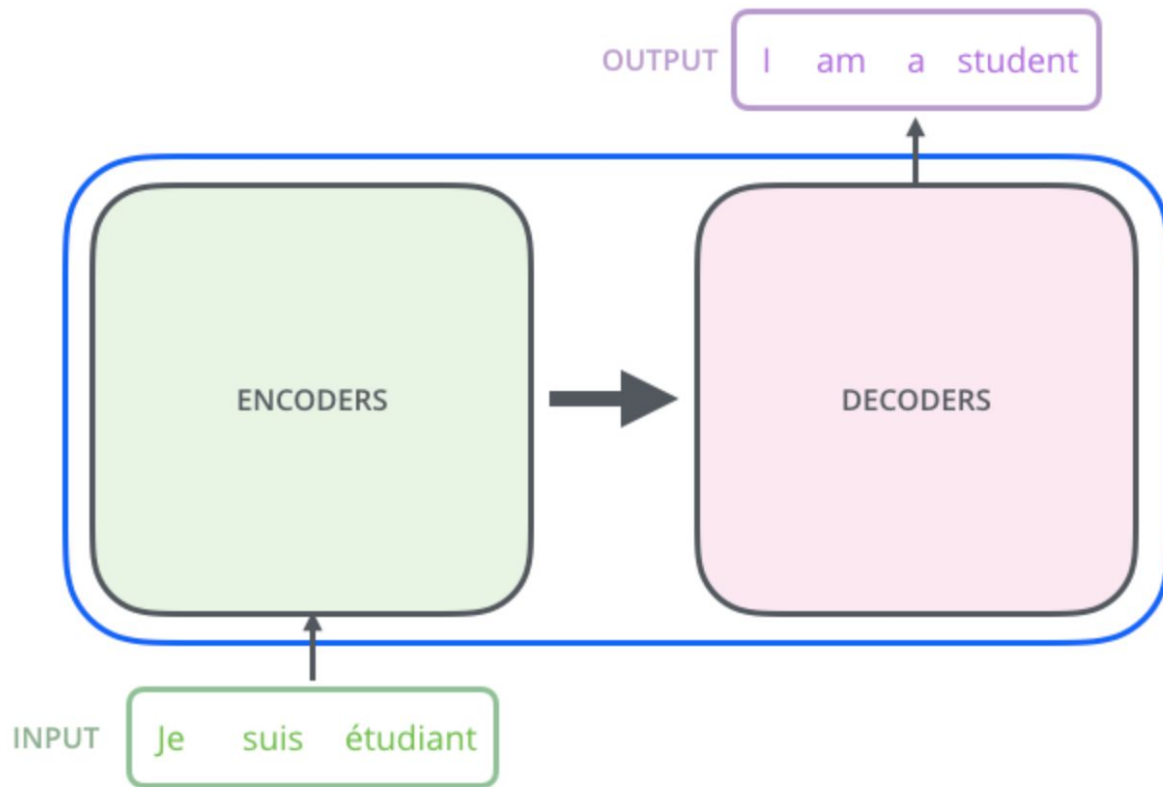


# The Transformer

# The Transformer

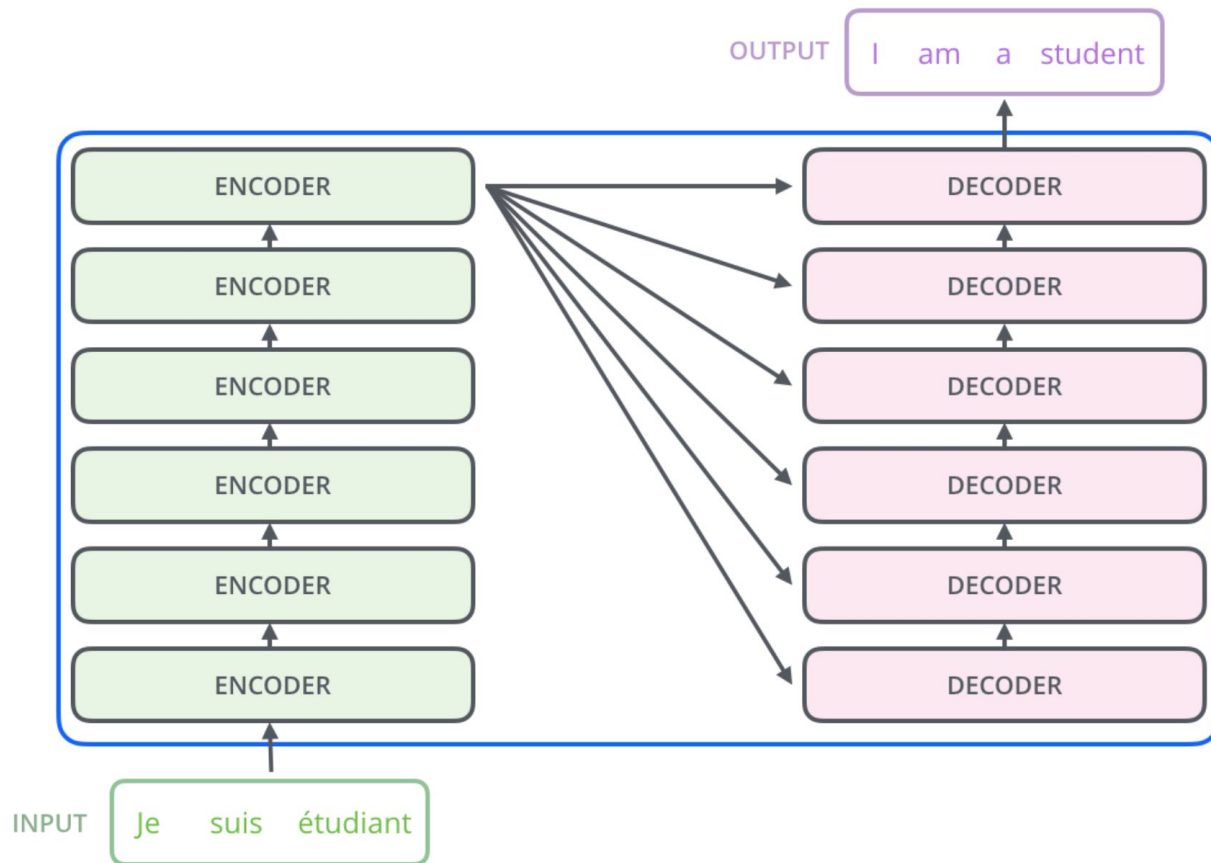


# The Transformer

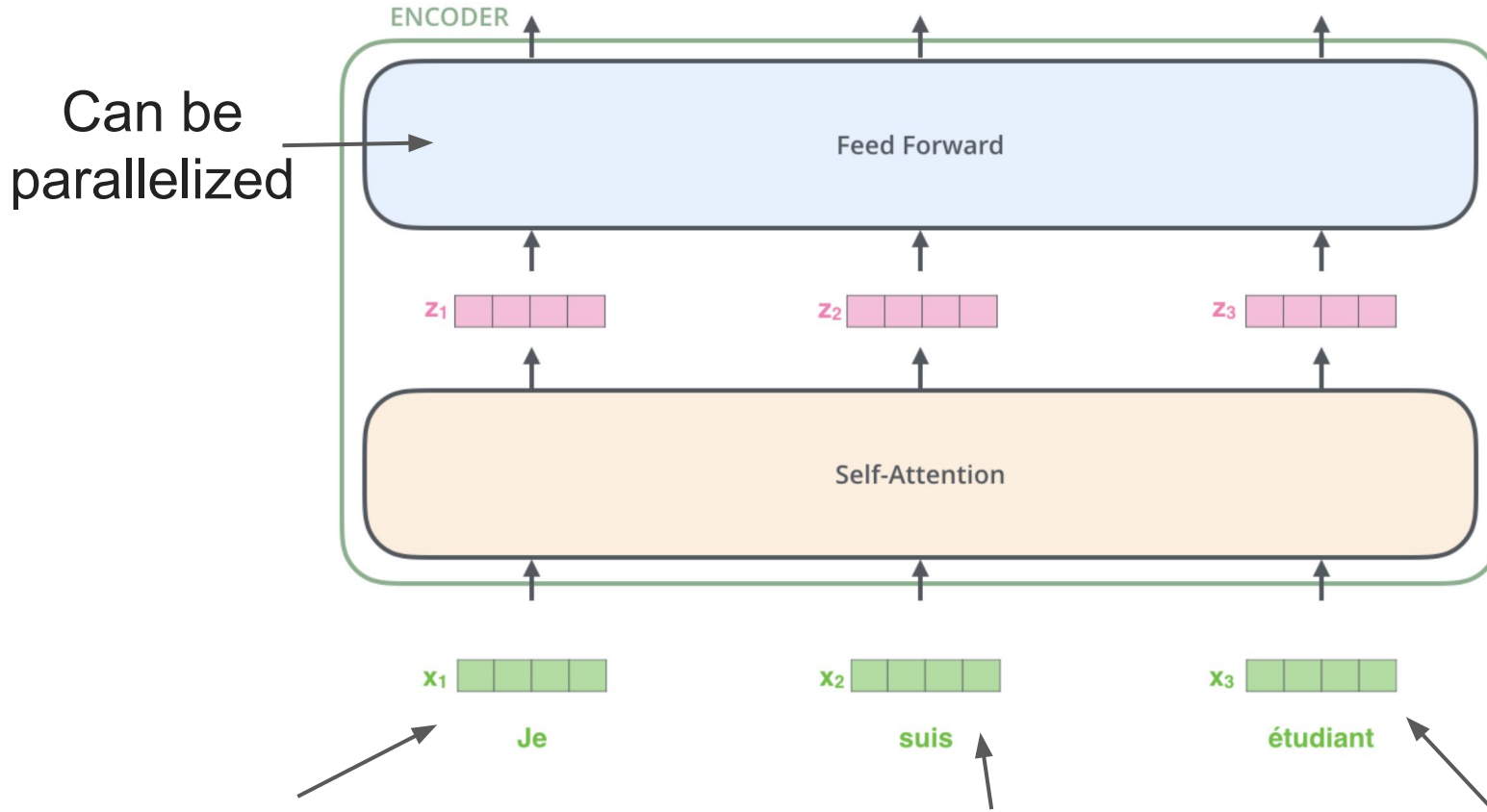




# The Transformer



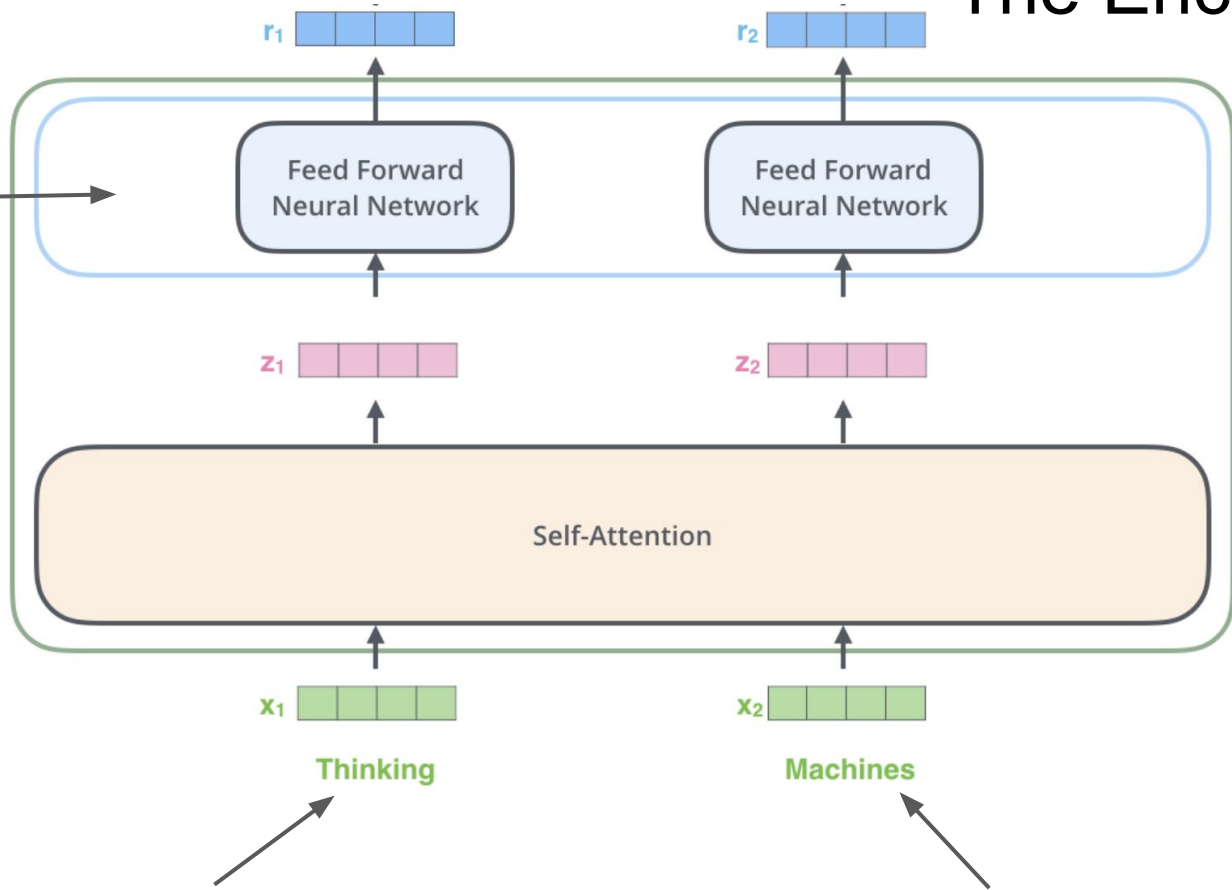
# The Encoder Side



the word in each position flows through its own path in the encoder

# The Encoder Side

Can be  
parallelized



the word in each position flows through its own path in the encoder

# The Transformer: quick overview

- Proposed in 2017 in paper [Attention is All You Need](#) by Ashish Vaswani et al.
- No recurrent or convolutional layers, only attention
- Beats seq2seq in machine translation task
  - *28.4 BLEU on the WMT 2014 English-to-German translation task*
- Much faster
- Uses **self-attention** concept

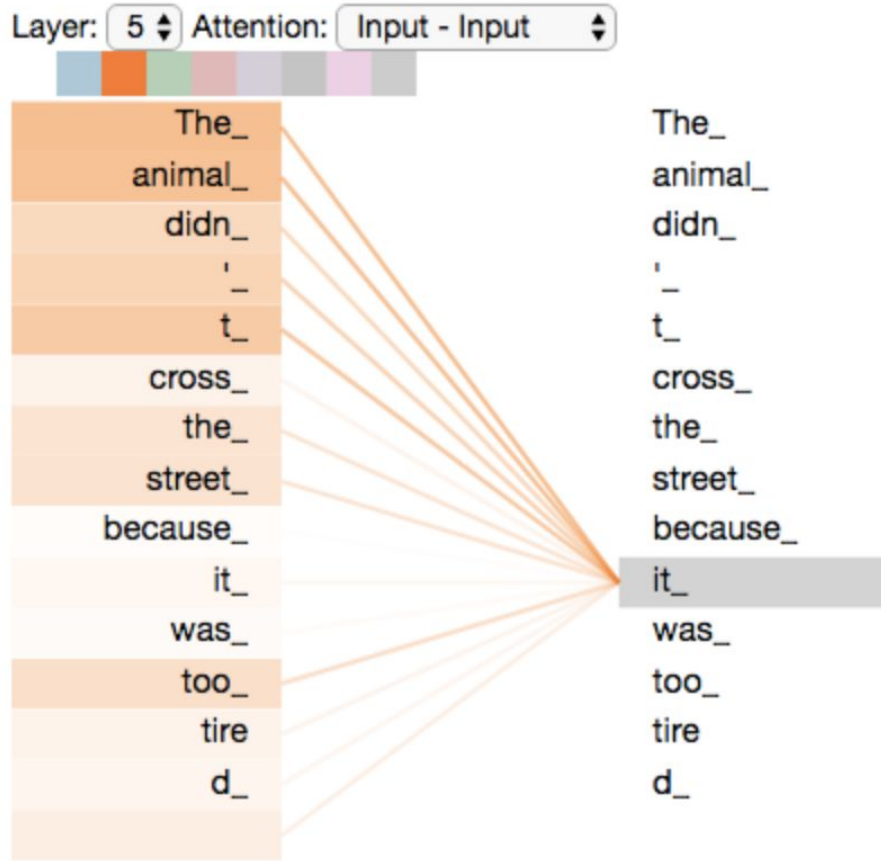
# Self-Attention

# Self-Attention at a High Level

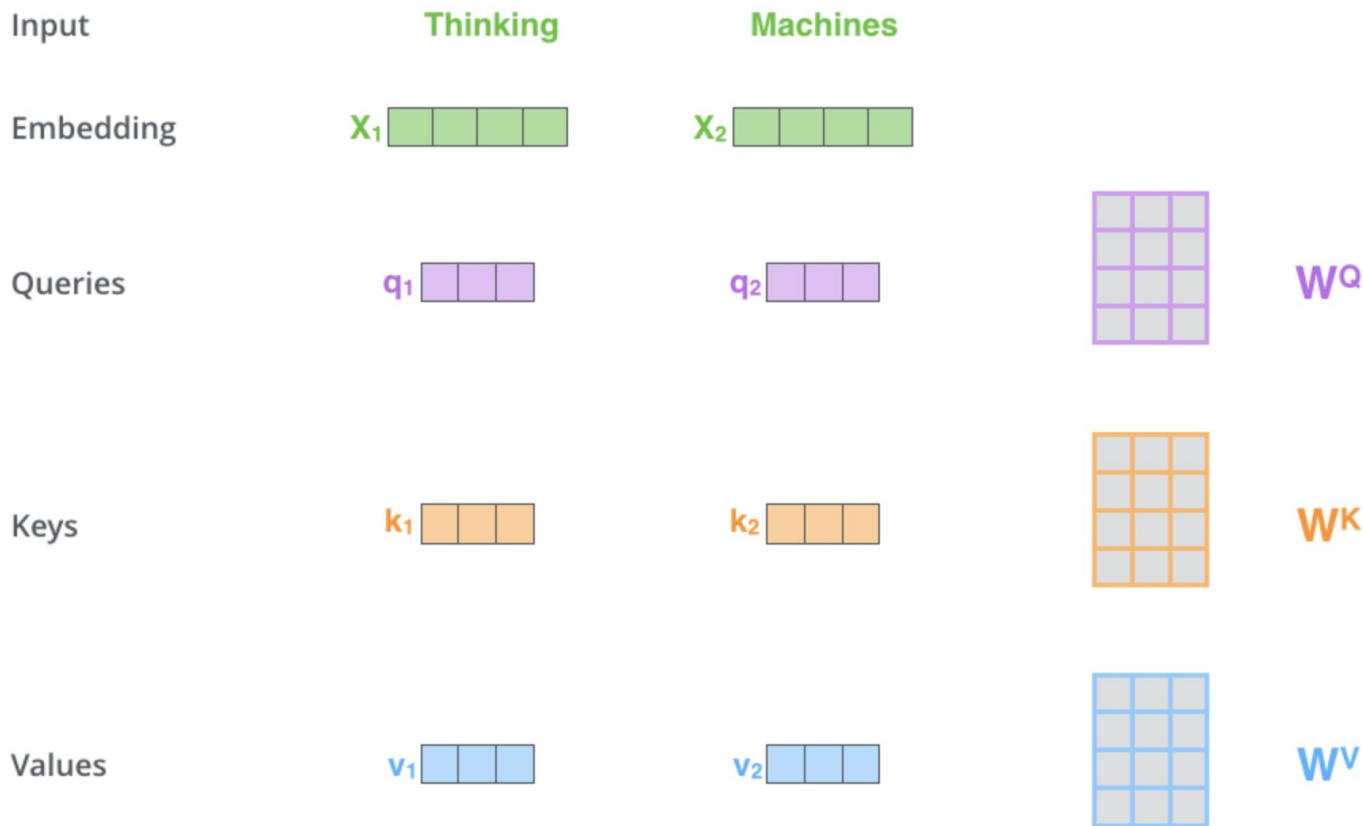
”The animal didn't cross the street because it was too tired”

- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”
- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing

# Self-Attention at a High Level



# Self-Attention: detailed explanation



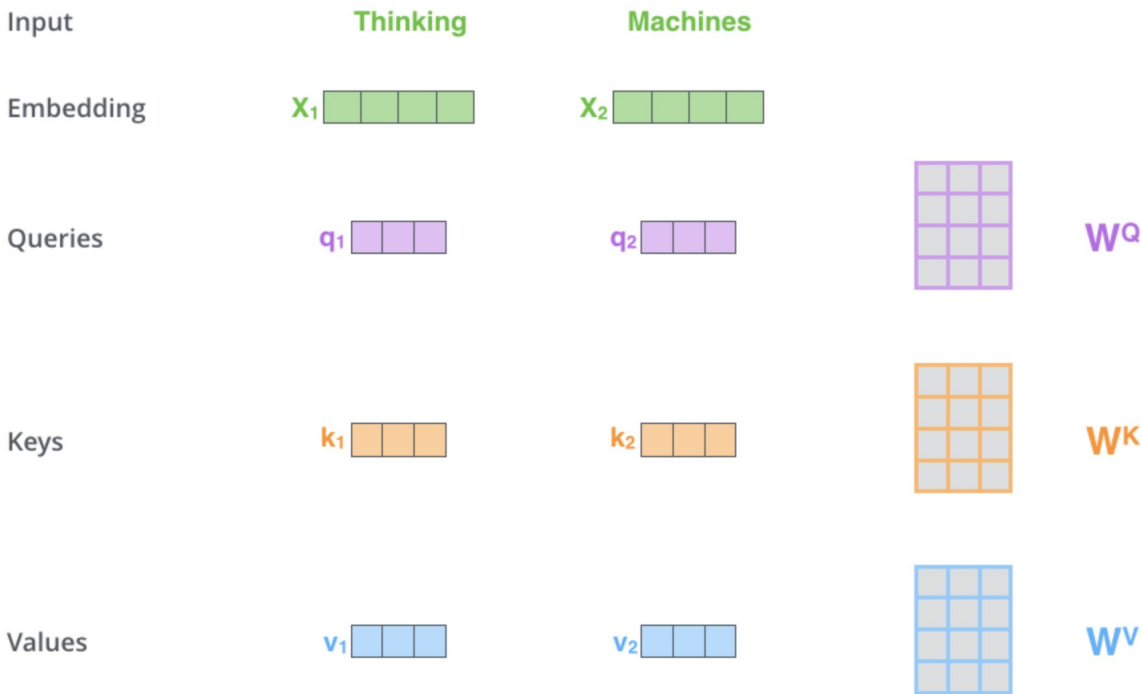


# Self-Attention: detailed explanation

## STEP 1:

create 3 vectors  
(**query**, **key**, **value**)

from each of the encoder's  
input vectors



# Self-Attention: detailed explanation

What are the **query**, **key**, **value** vectors?

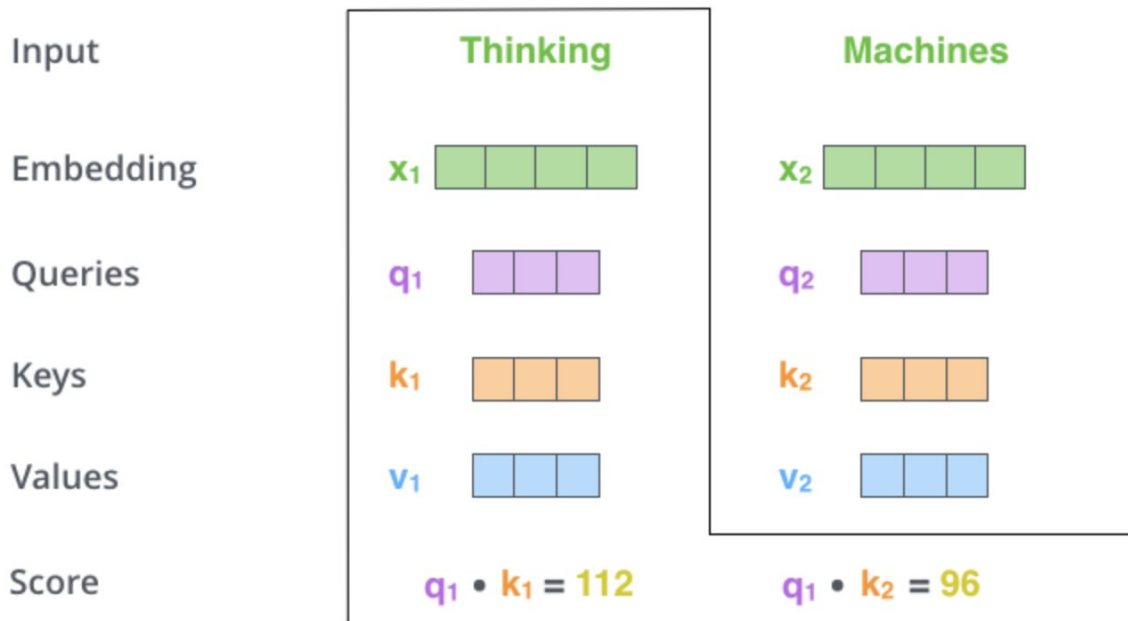
They're abstractions that are useful for calculating and thinking about attention.

# Self-Attention: detailed explanation

## STEP 2:

calculate a score

(score each word of the input sentence against the current word)



# Self-Attention: detailed explanation

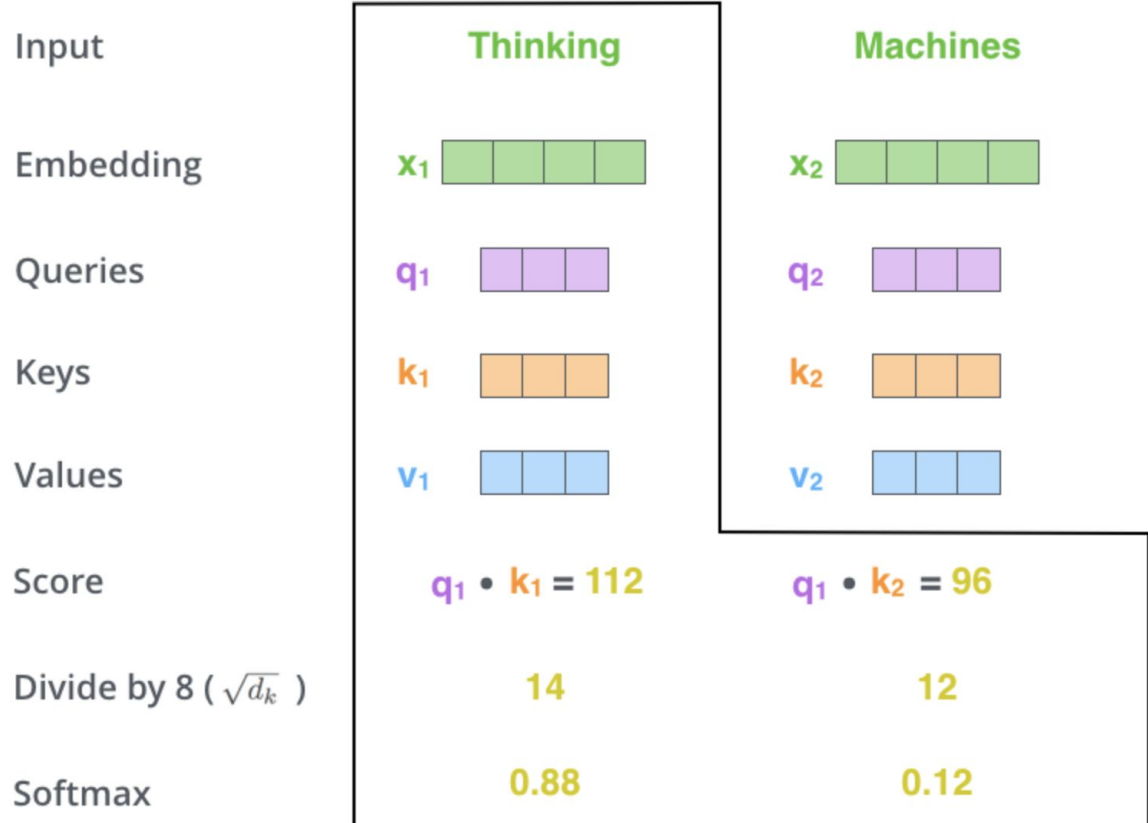
## STEP 3:

divide the scores by 8

(the square root of the dimension of the key vectors)

## STEP 4:

softmax



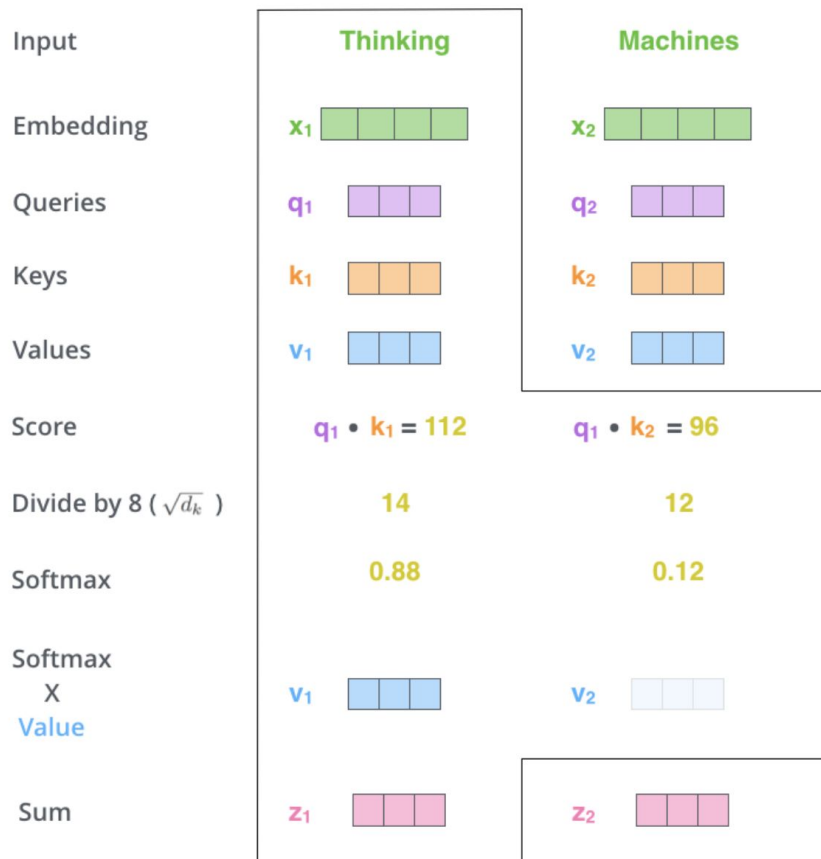
# Self-Attention: detailed explanation

## STEP 5:

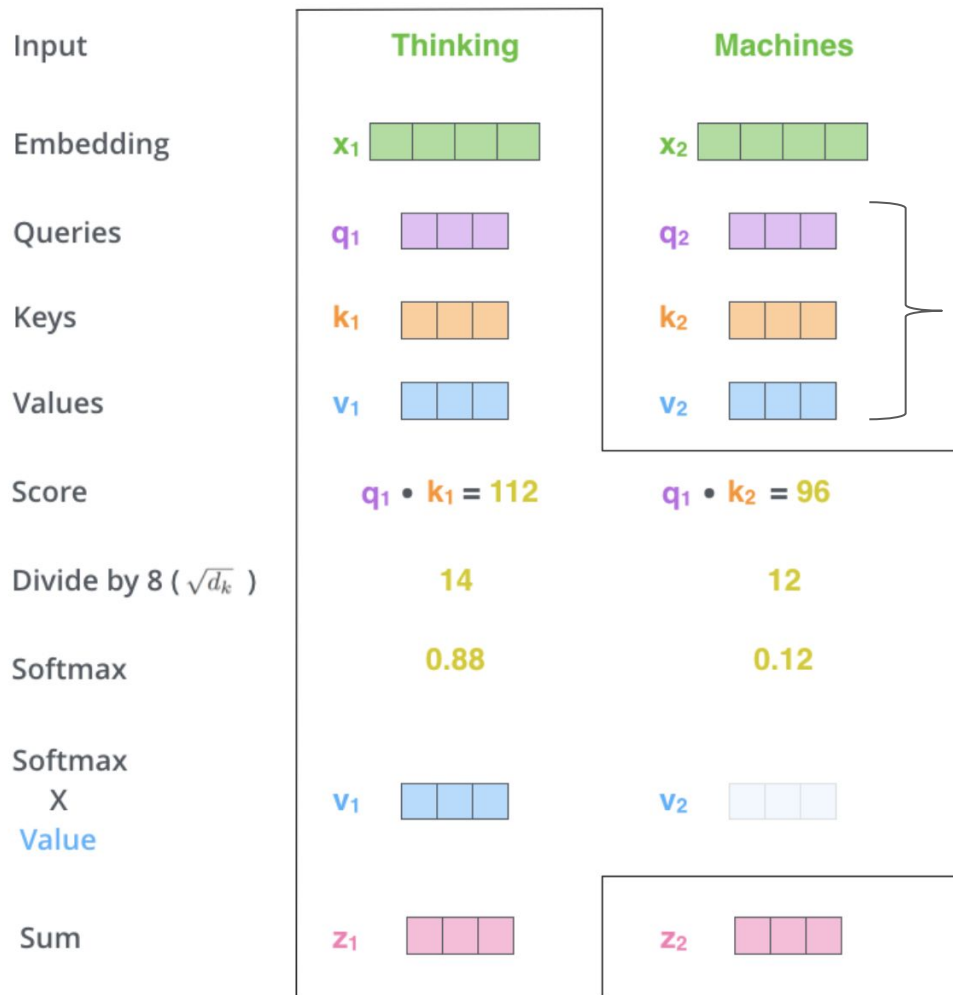
multiply each value vector by the softmax score

## STEP 6:

sum up the weighted value vectors



# Self-Attention



**STEP 1:** create Query, Key, Value

**STEP 2:** calculate scores

**STEP 3:** divide by  $\sqrt{d_k}$

**STEP 4:** softmax

**STEP 5:** multiply each value vector by the softmax score

**STEP 6:** sum up the weighted value vectors

# Self-Attention: Matrix Calculation

Pack embeddings into matrix **X**

Multiply **X** by weight matrices we've trained (**W<sub>k</sub>**, **W<sub>q</sub>**, **W<sub>v</sub>**)



# Self-Attention: Matrix Calculation

$$\text{softmax}\left(\frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline \square & \square \\ \hline \square & \square \\ \hline \square & \square \\ \hline \end{array} \end{matrix}\right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array} \end{matrix}$$

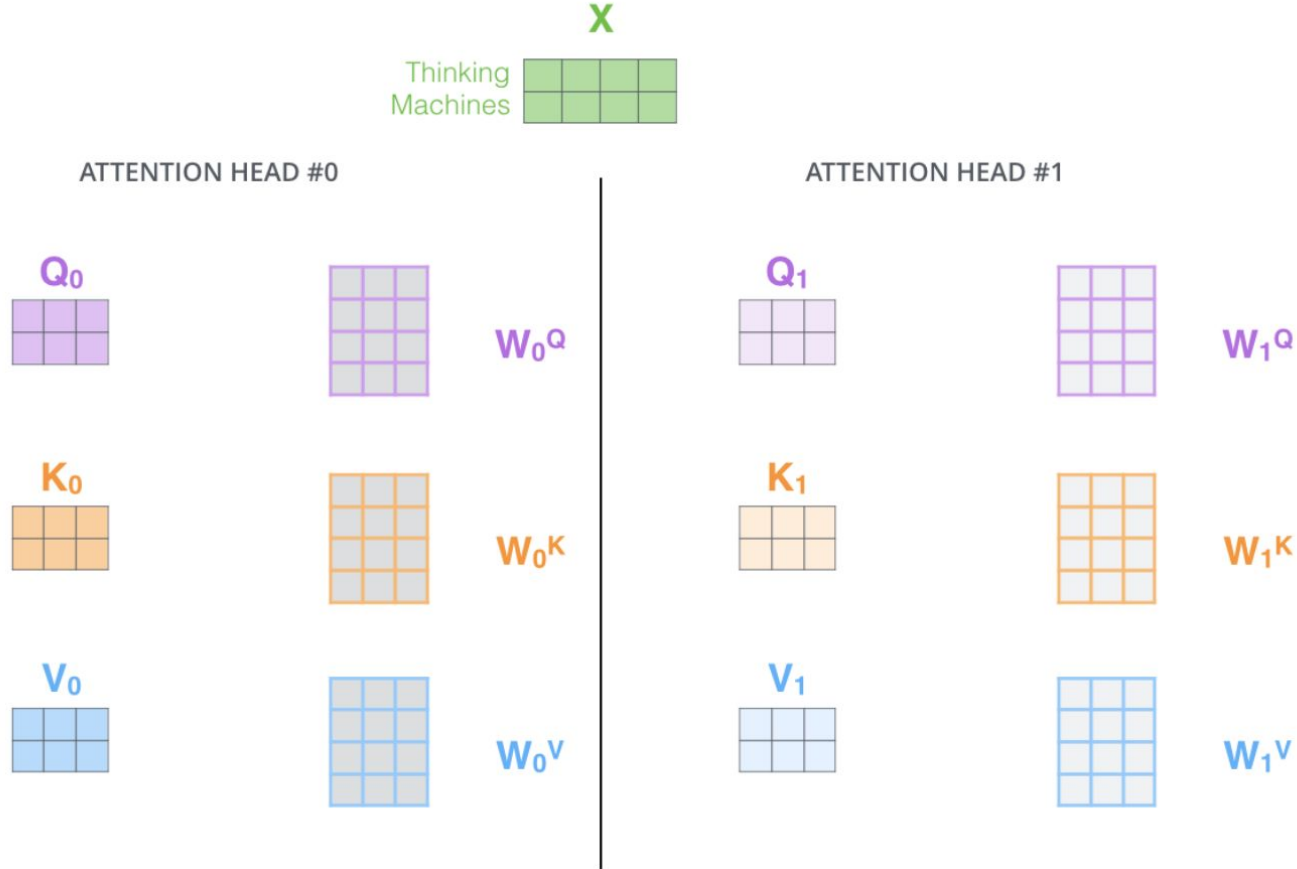
=

$\text{Z}$

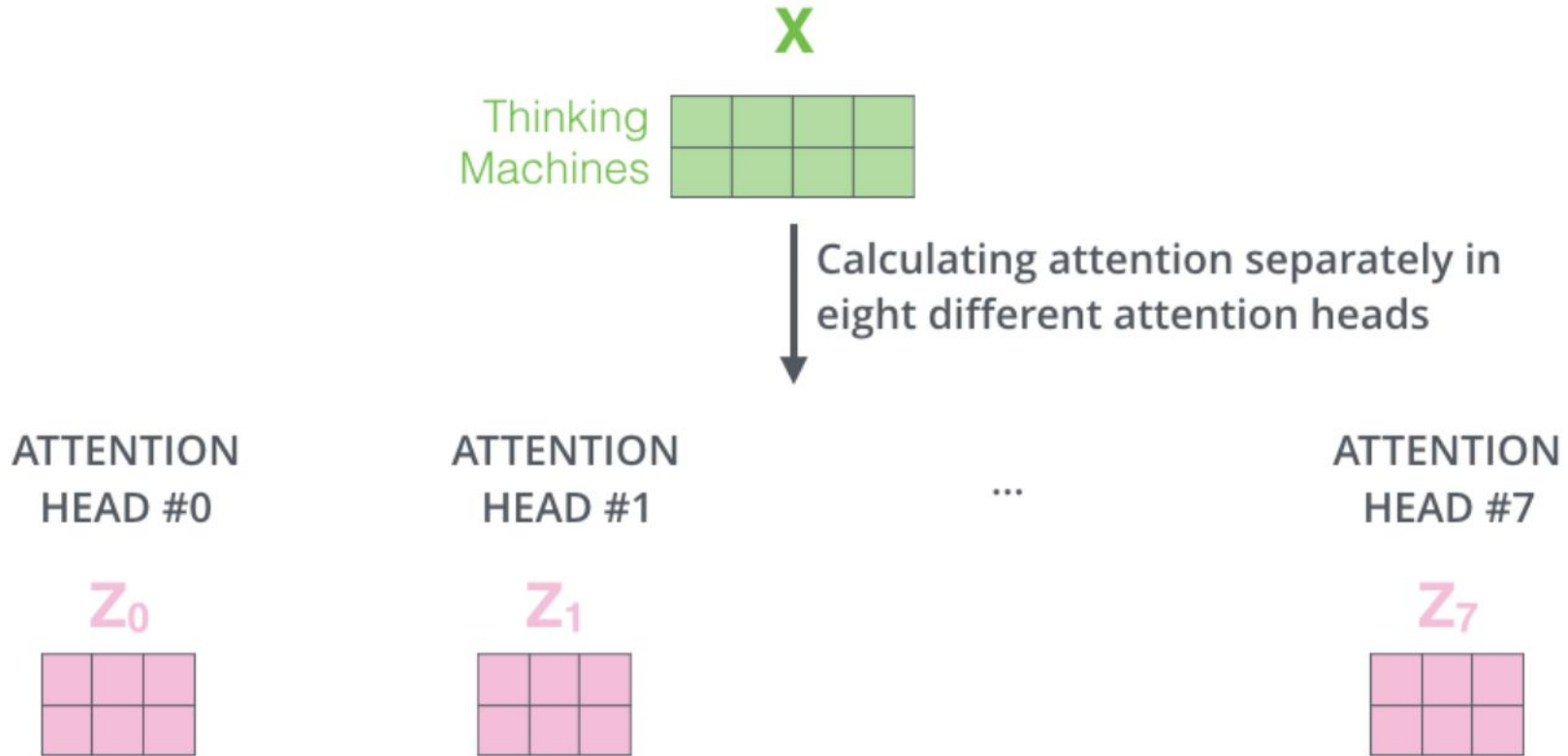
$\begin{array}{|c|c|c|} \hline \square & \square & \square \\ \hline \square & \square & \square \\ \hline \end{array}$



# Multi-Head Attention



# Multi-Head Attention

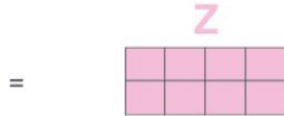


# Multi-Head Attention

1) Concatenate all the attention heads

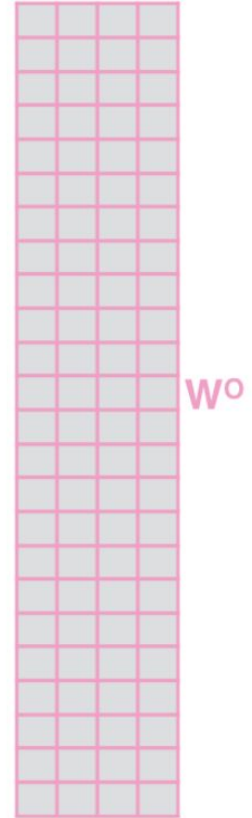


3) The result would be the  $Z$  matrix that captures information from all the attention heads. We can send this forward to the FFNN

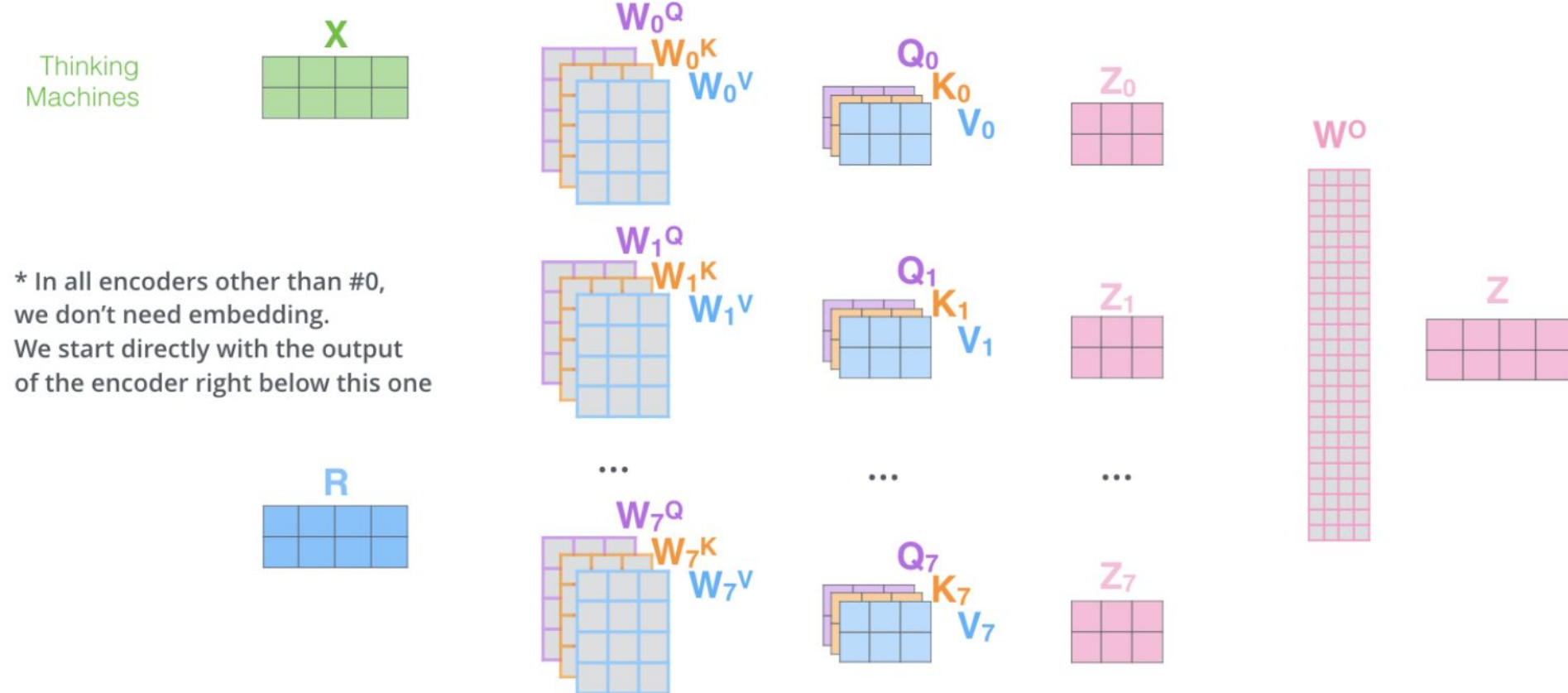


2) Multiply with a weight matrix  $W^O$  that was trained jointly with the model

$\times$



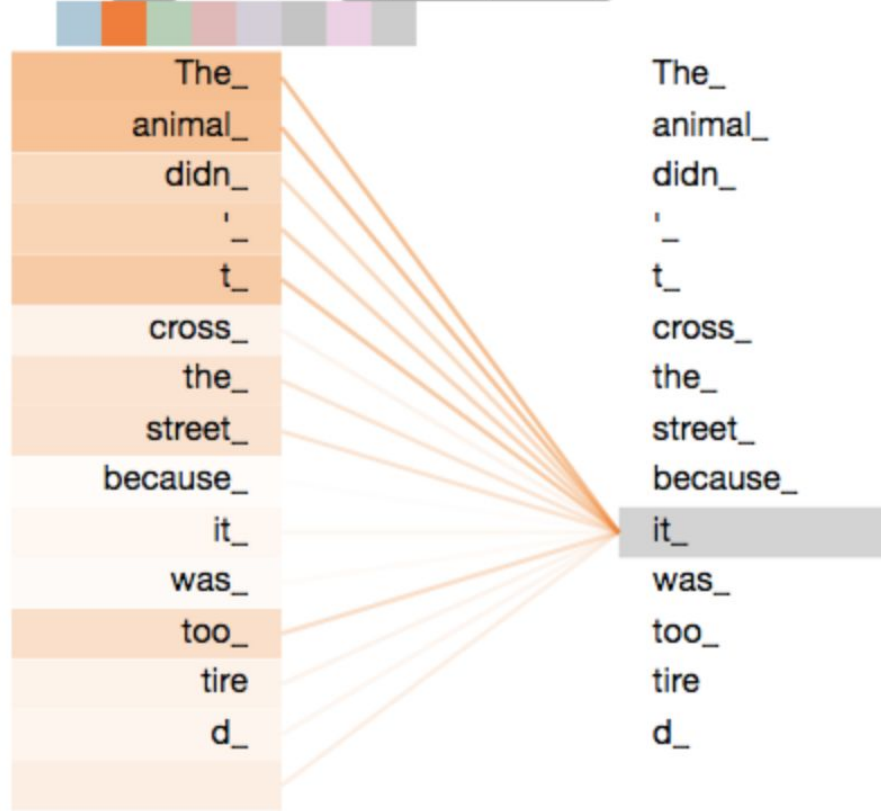
- 1) This is our input sentence\*
- 2) We embed each word\*
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer



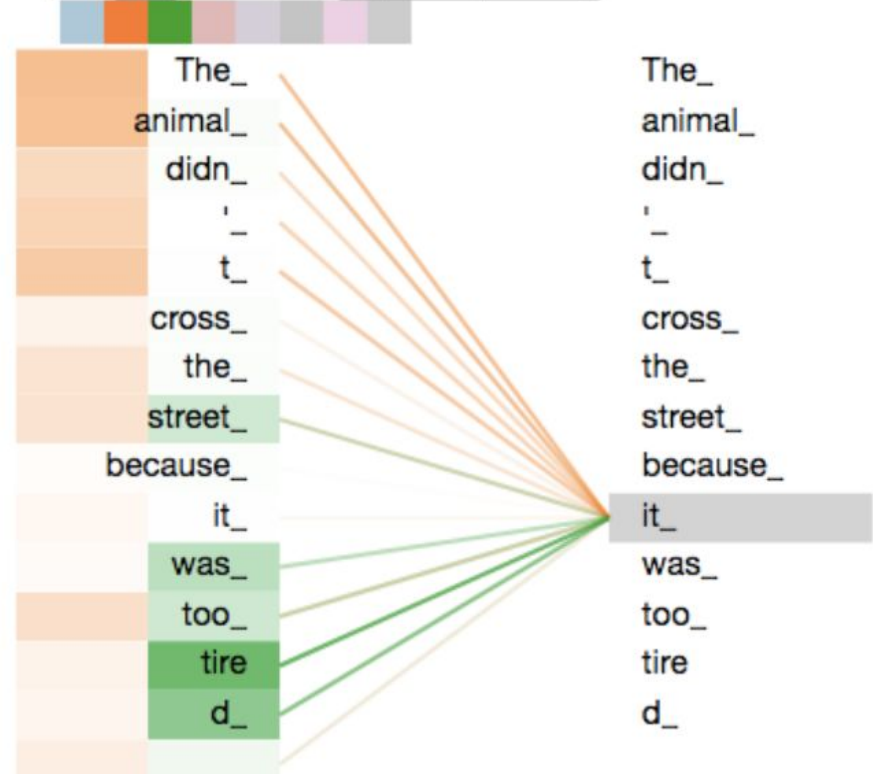
\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

# Multi-Head Attention

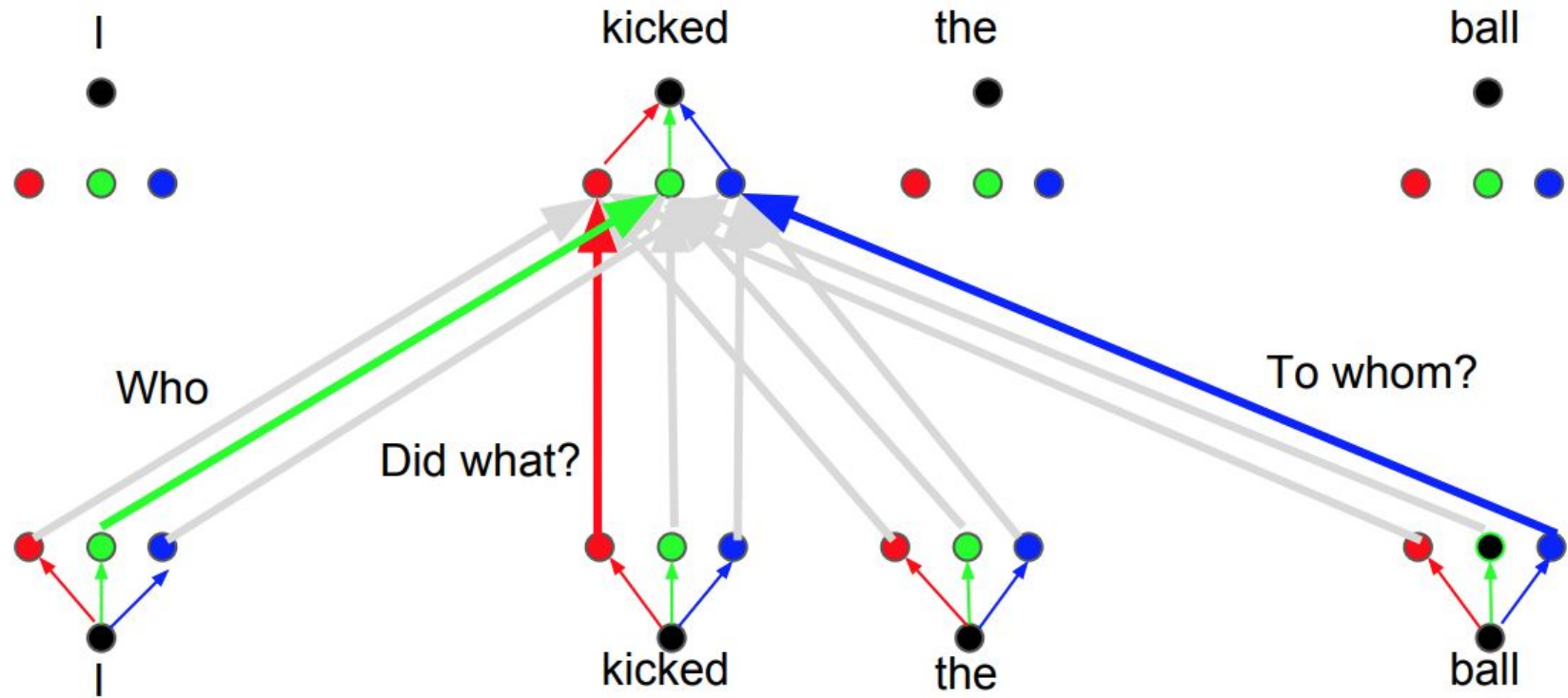
Layer: 5 Attention: Input - Input



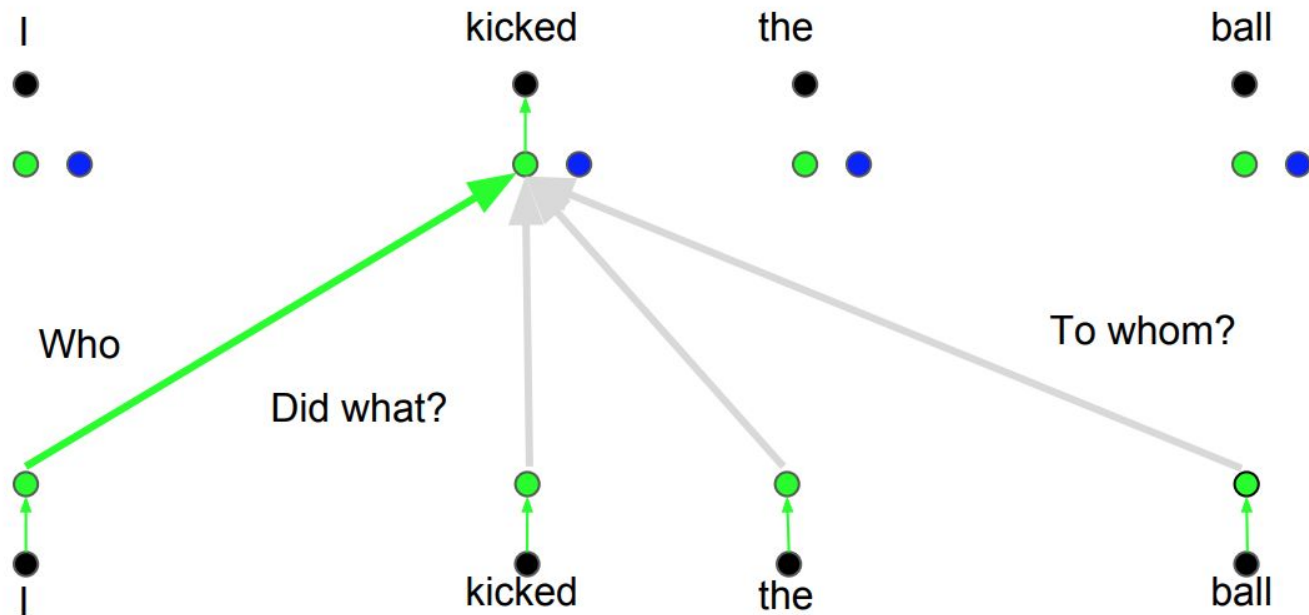
Layer: 5 Attention: Input - Input



# Why Multi-Head Attention?



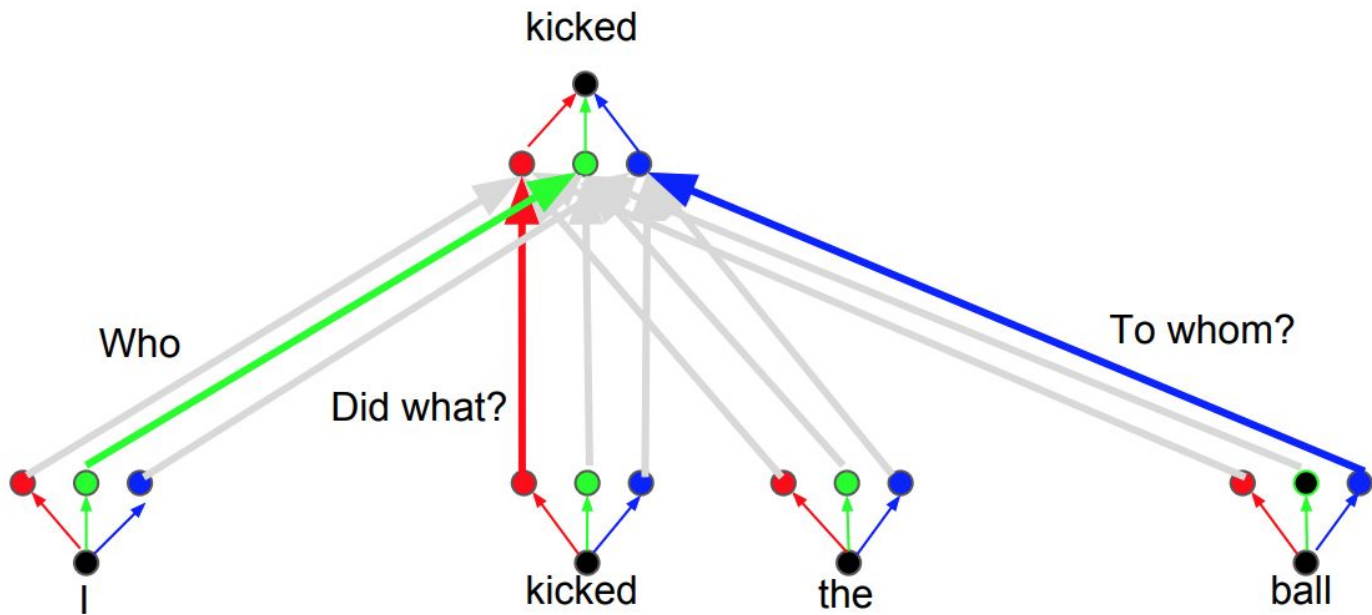
# Attention head: Who





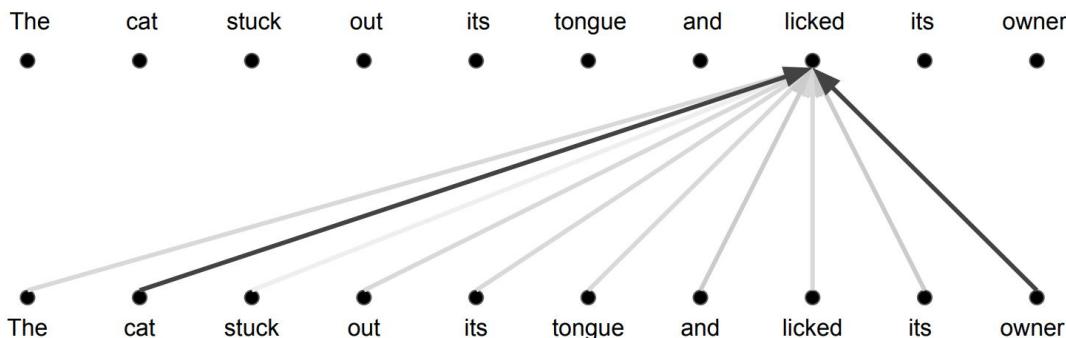


# Attention head: To Whom?



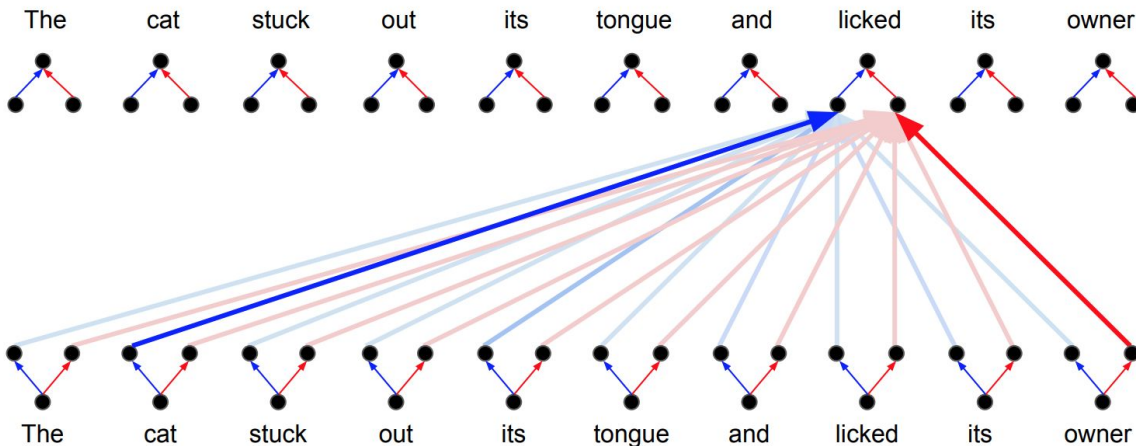
# Attention vs. Multi-Head Attention

**Attention:** a weighted average



**Multi-Head Attention:**

parallel attention layers  
with different linear  
transformations on input  
and output.



# Performance: WMT 2014 BLEU

	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	<b>28.4</b>	<b>41.8</b>

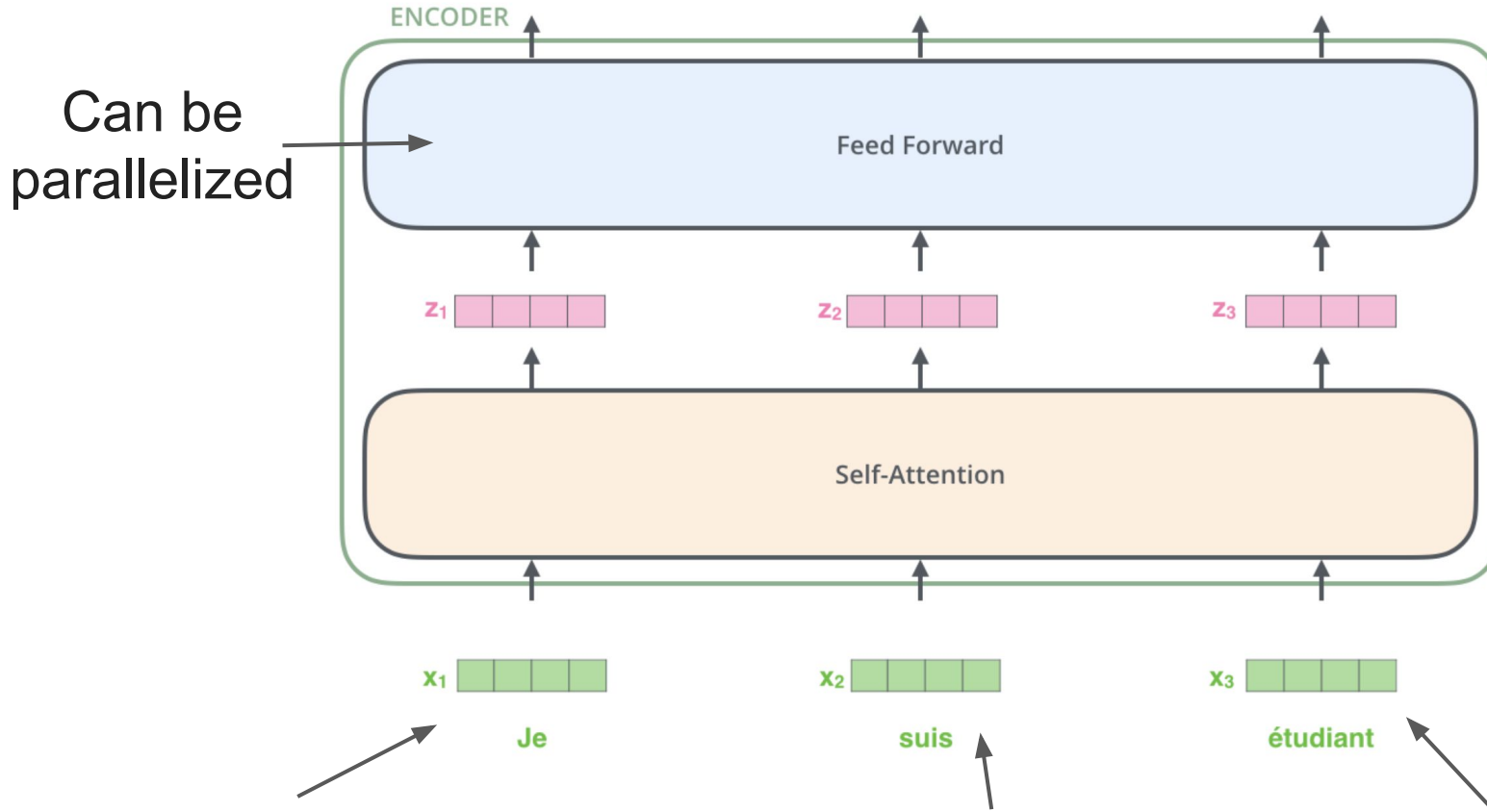
\*Transformer models trained >3x faster than the others.

# Research Challenges

- Constant 'path length' between any two positions.
- Unbounded memory.
- Trivial to parallelize (per layer).
- Models Self-Similarity.
- Relative attention provides expressive timing, equivariance, and extends naturally to graphs.

# Positional Encoding

# The Encoder Side

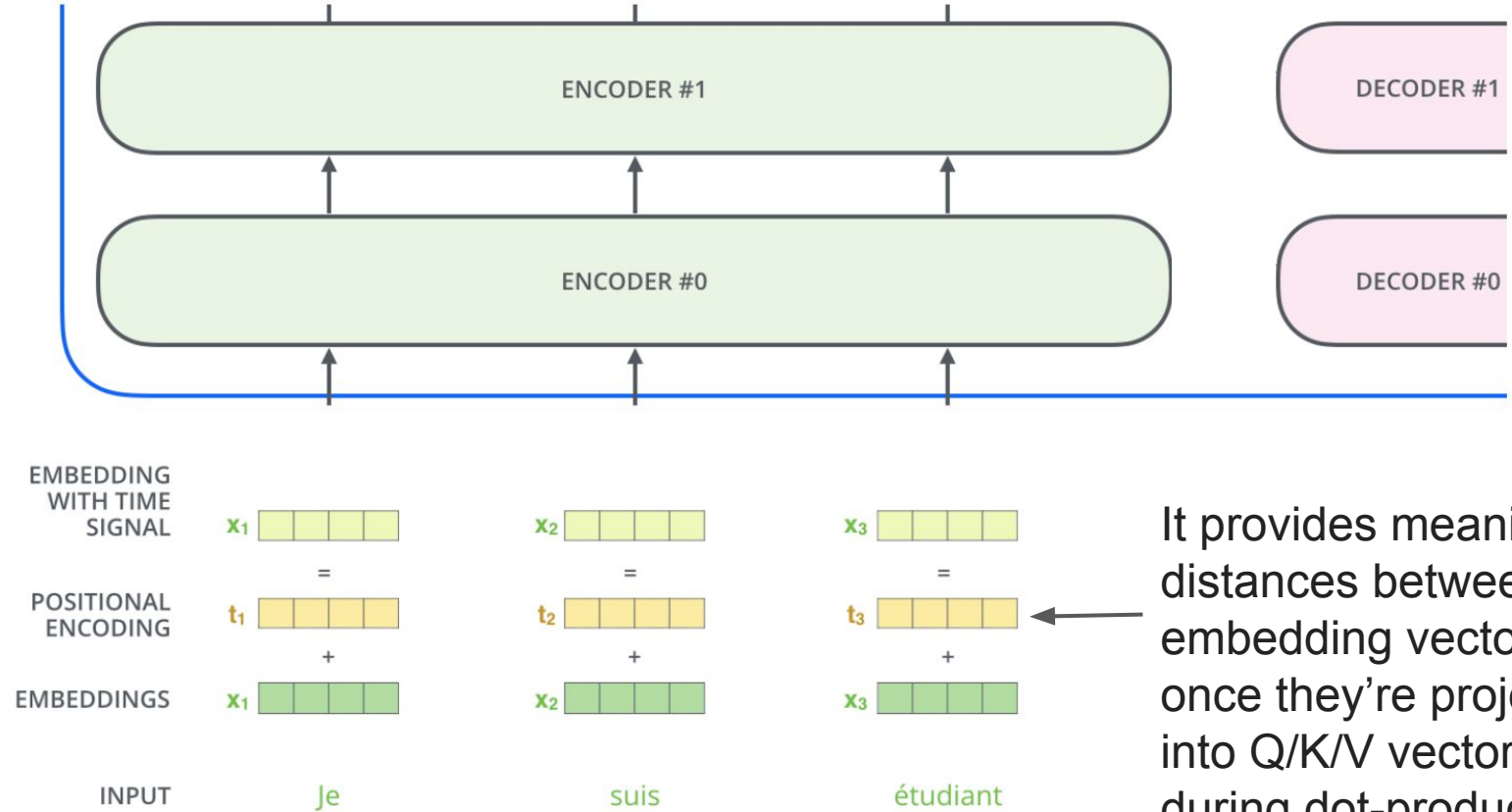


the word in each position flows through its own path in the encoder

# Positional encoding requirements

- Positional encoding should be unique for every position in the sequence
- Distance between two same positions should be preserved with sequences of different length
- The positional encoding should be deterministic
- *It would be great if it would work with long sequences (longer than any sequence in the training set)*

# Positional Encoding





# Positional Encoding: why sin and cos?

$$\vec{p}_t^{(i)} = f(t)^{(i)} = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k \\ \cos(\omega_k t), & \text{if } i = 2k + 1 \end{cases}$$
$$\omega_k = \frac{1}{10000^{2k/d}}$$
$$\vec{p}_t = \begin{bmatrix} \sin(\omega_1.t) \\ \cos(\omega_1.t) \\ \\ \sin(\omega_2.t) \\ \cos(\omega_2.t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2}.t) \\ \cos(\omega_{d/2}.t) \end{bmatrix}_{d \times 1}$$

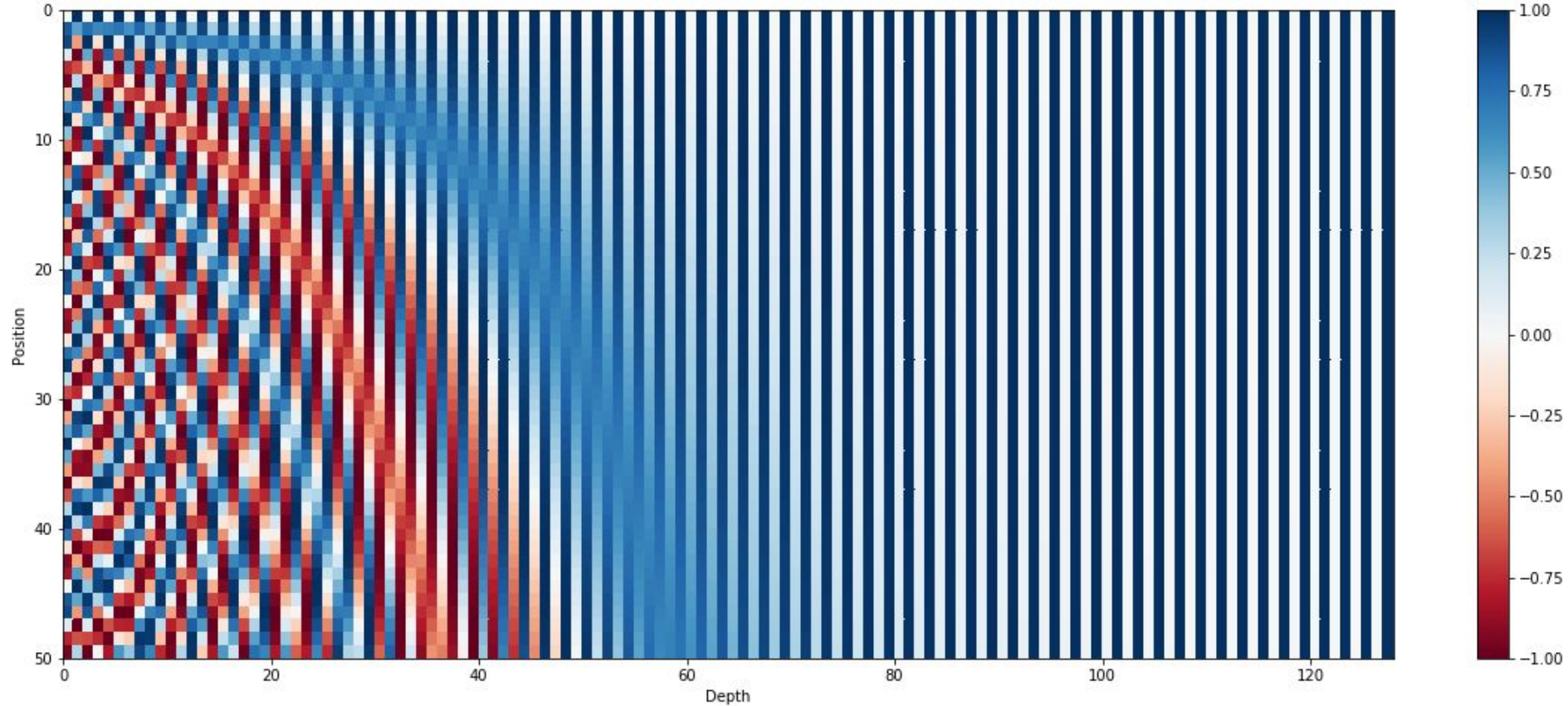
t stays for position in the original sequence

k is the index of the element in the positional vector

# Positional Encoding

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

# Positional Encoding



# Positional Encoding: why sin and cos?

*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  $PE_{pos}$ .*

$$M \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k (t + \phi)) \\ \cos(\omega_k (t + \phi)) \end{bmatrix}$$

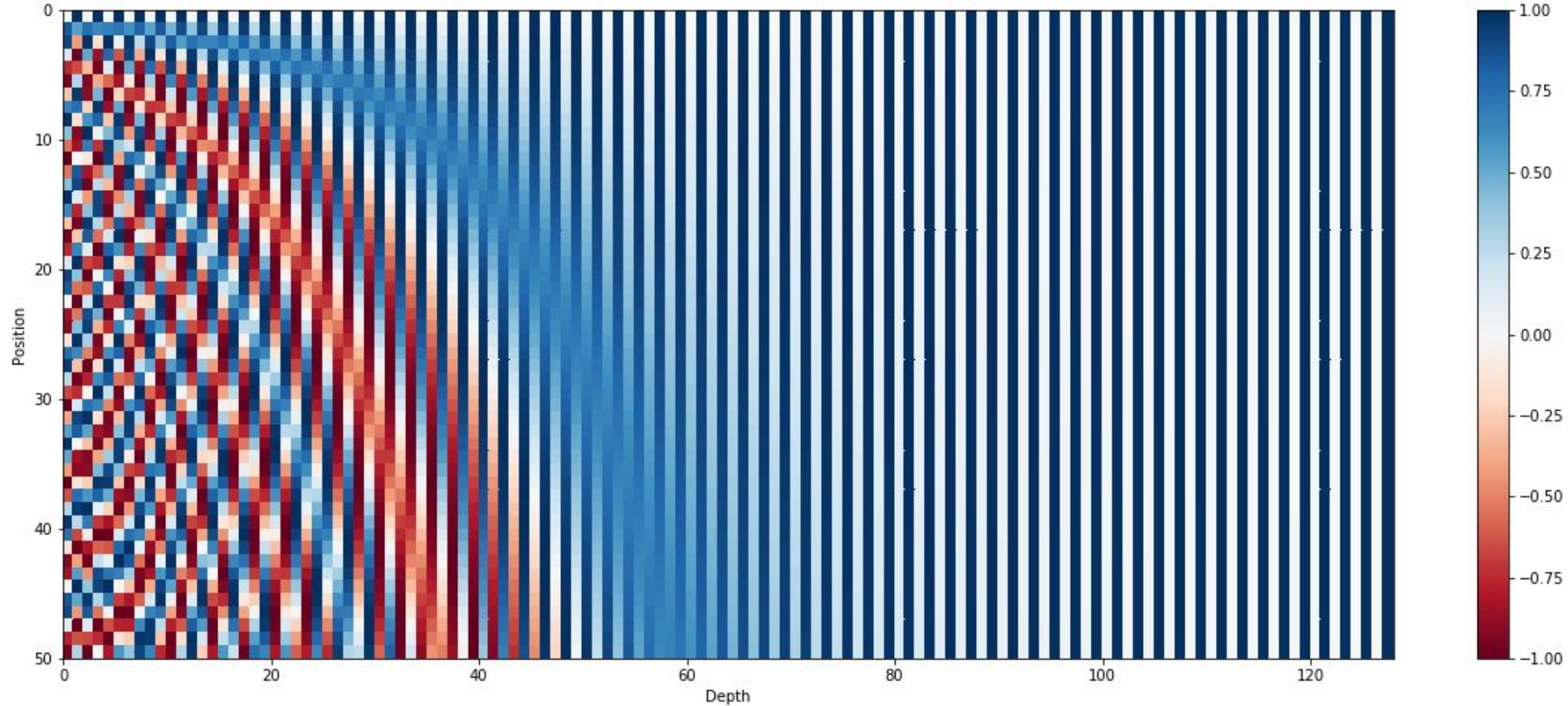
## Positional Encoding: why sin and cos?

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k(t + \phi)) \\ \cos(\omega_k(t + \phi)) \end{bmatrix}$$

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \end{bmatrix} \begin{bmatrix} \sin(\omega_k t) \\ \cos(\omega_k t) \end{bmatrix} = \begin{bmatrix} \sin(\omega_k t) \cos(\omega_k \phi) + \cos(\omega_k t) \sin(\omega_k \phi) \\ \cos(\omega_k t) \cos(\omega_k \phi) - \sin(\omega_k t) \sin(\omega_k \phi) \end{bmatrix}$$

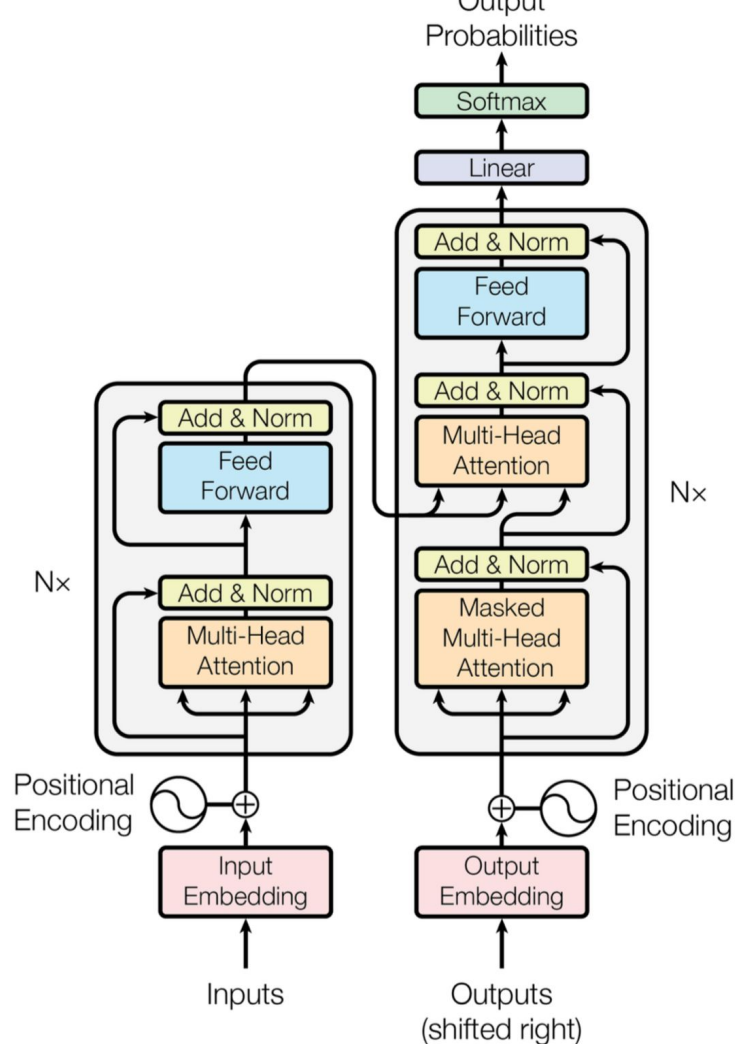
$$M_{\phi, k} = \begin{bmatrix} \cos(\omega_k \phi) & \sin(\omega_k \phi) \\ -\sin(\omega_k \phi) & \cos(\omega_k \phi) \end{bmatrix}$$

# Positional Encoding



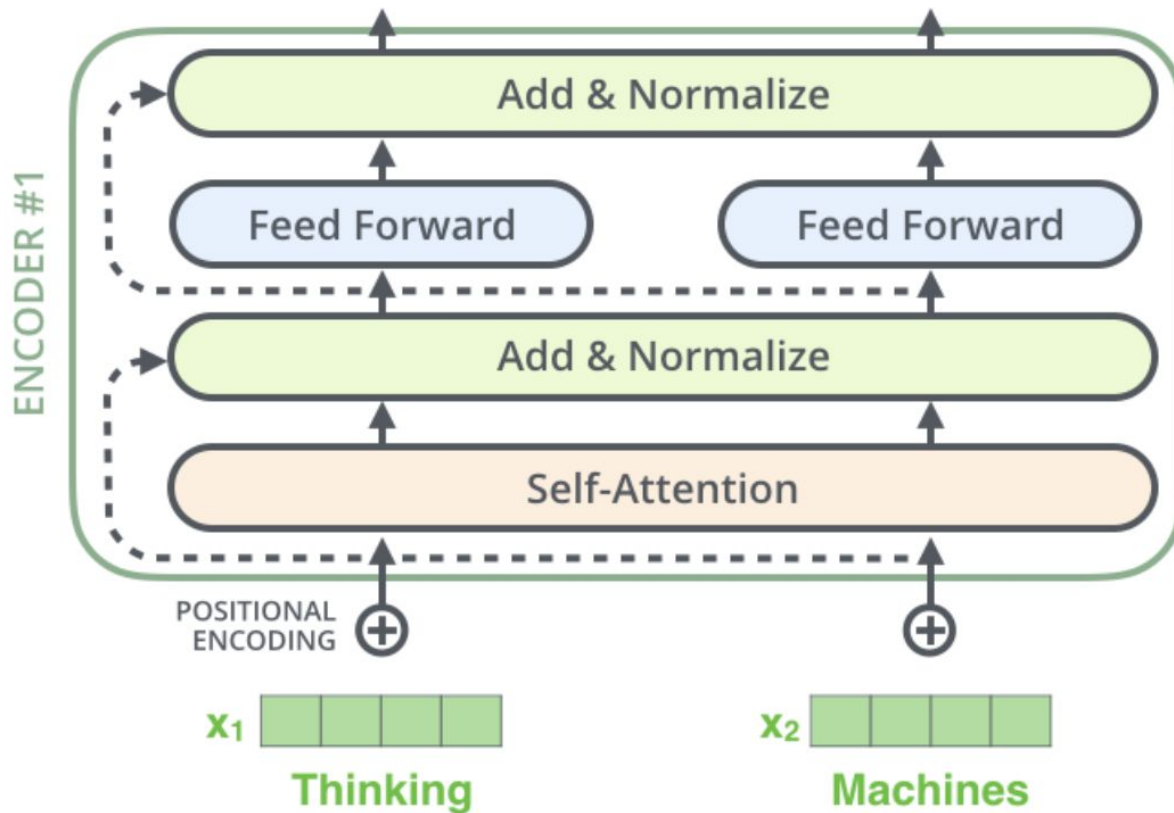
# Layer Normalization

# The Transformer: recap

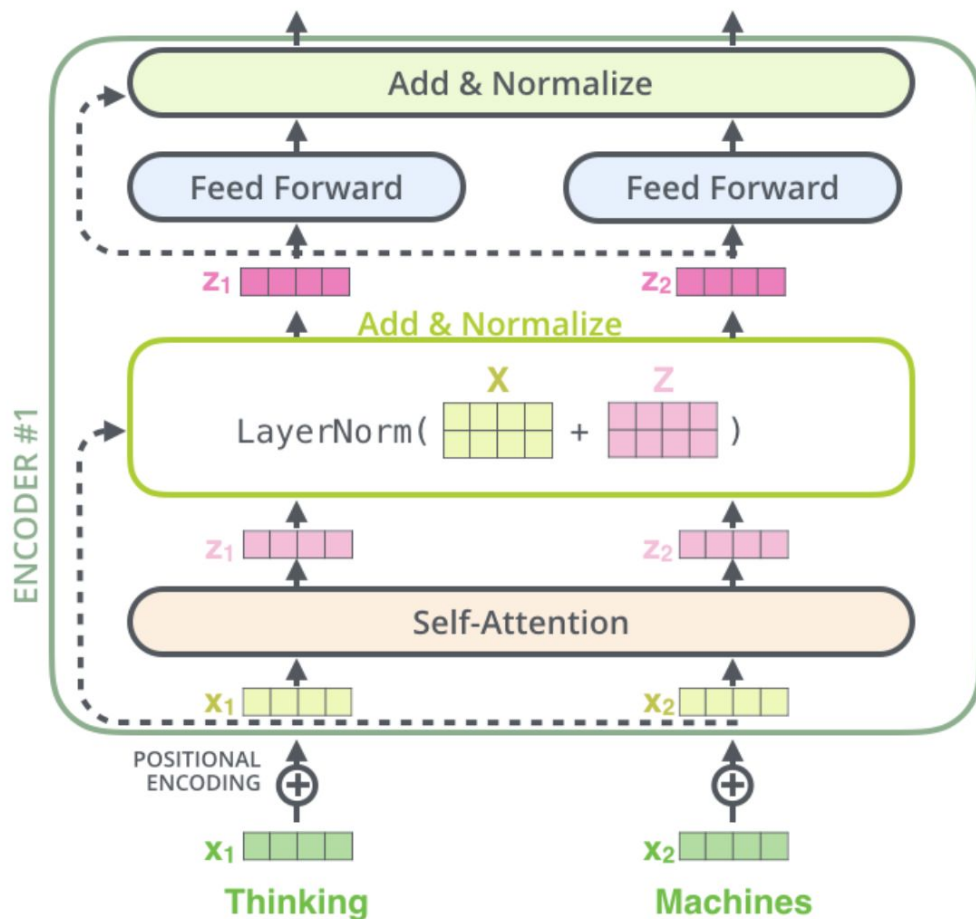




# Layer Normalization



# Layer Normalization



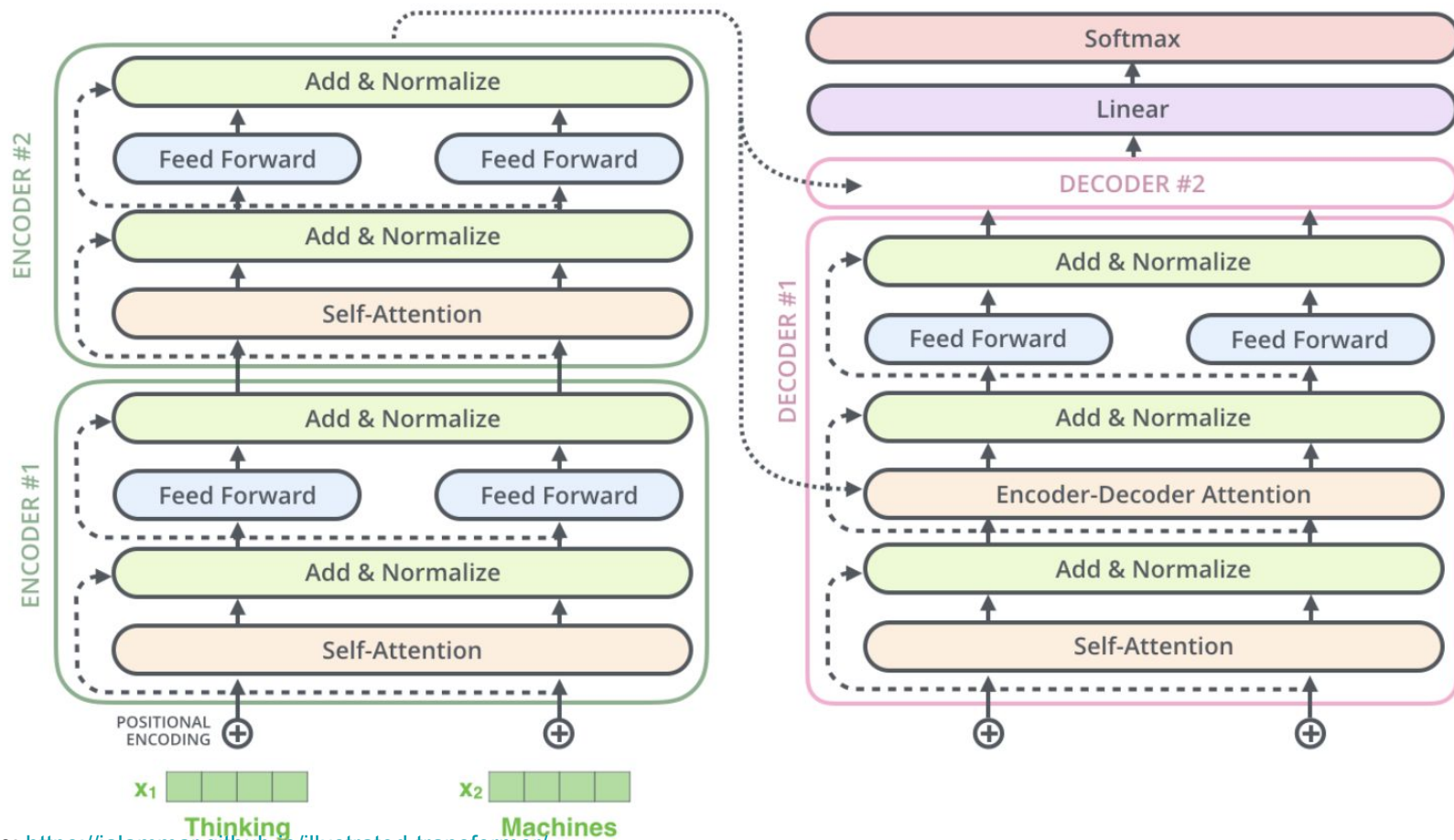
Like BatchNorm

but normalize along  
all features  
representing latent  
vector

More info:

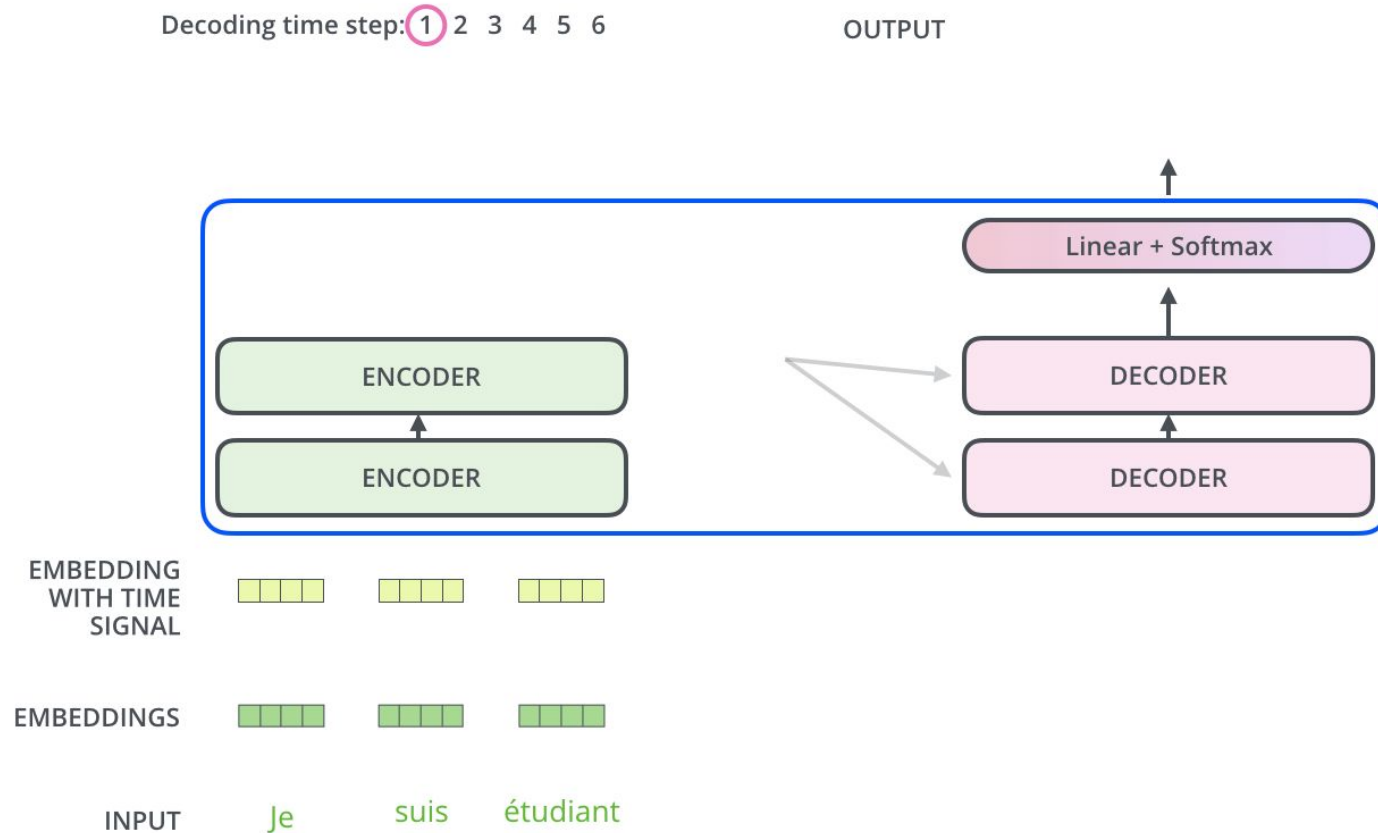
[Layer Normalization](https://jalammar.github.io/illustrated-transformer/)

# Layer Normalization

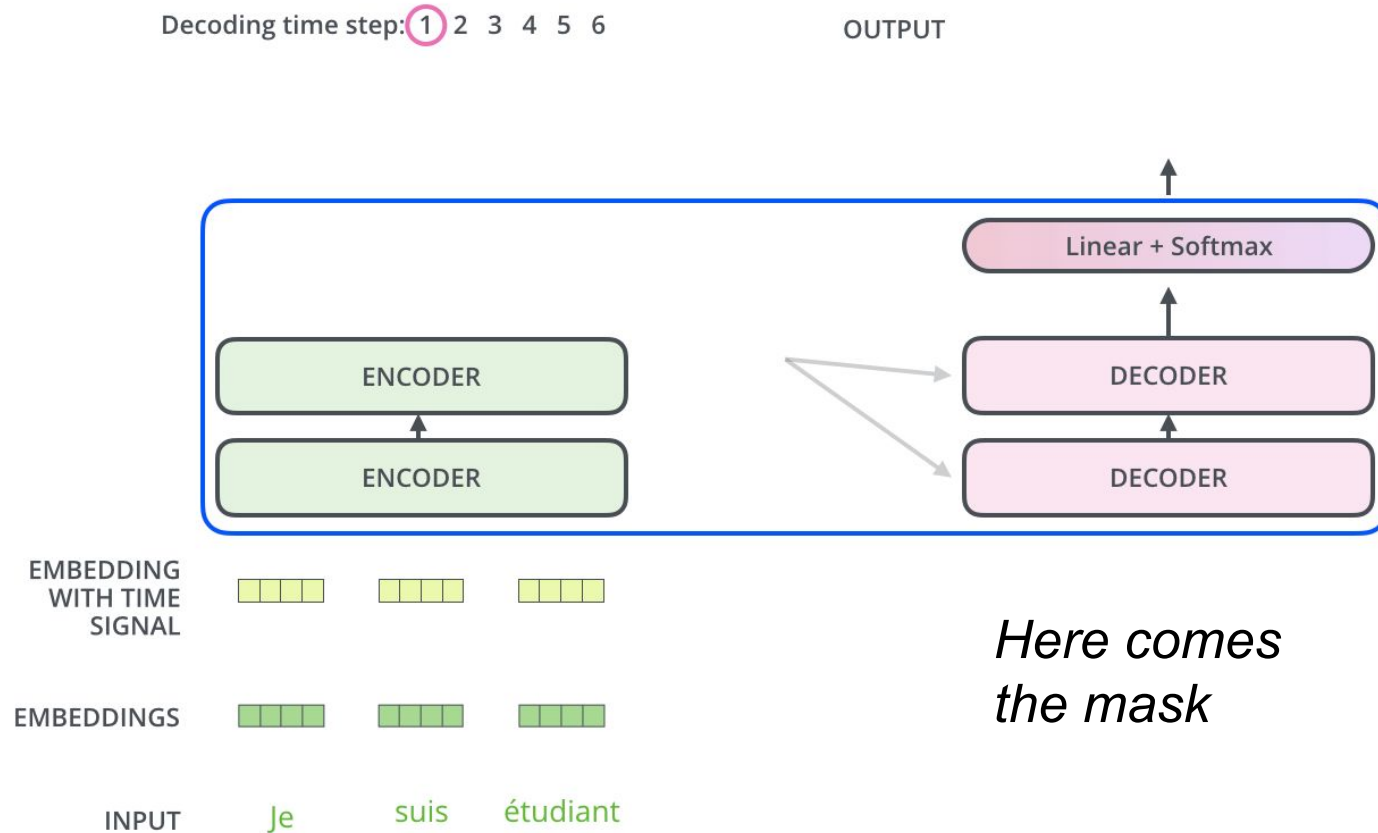


# The Decoder

# The Decoder Side



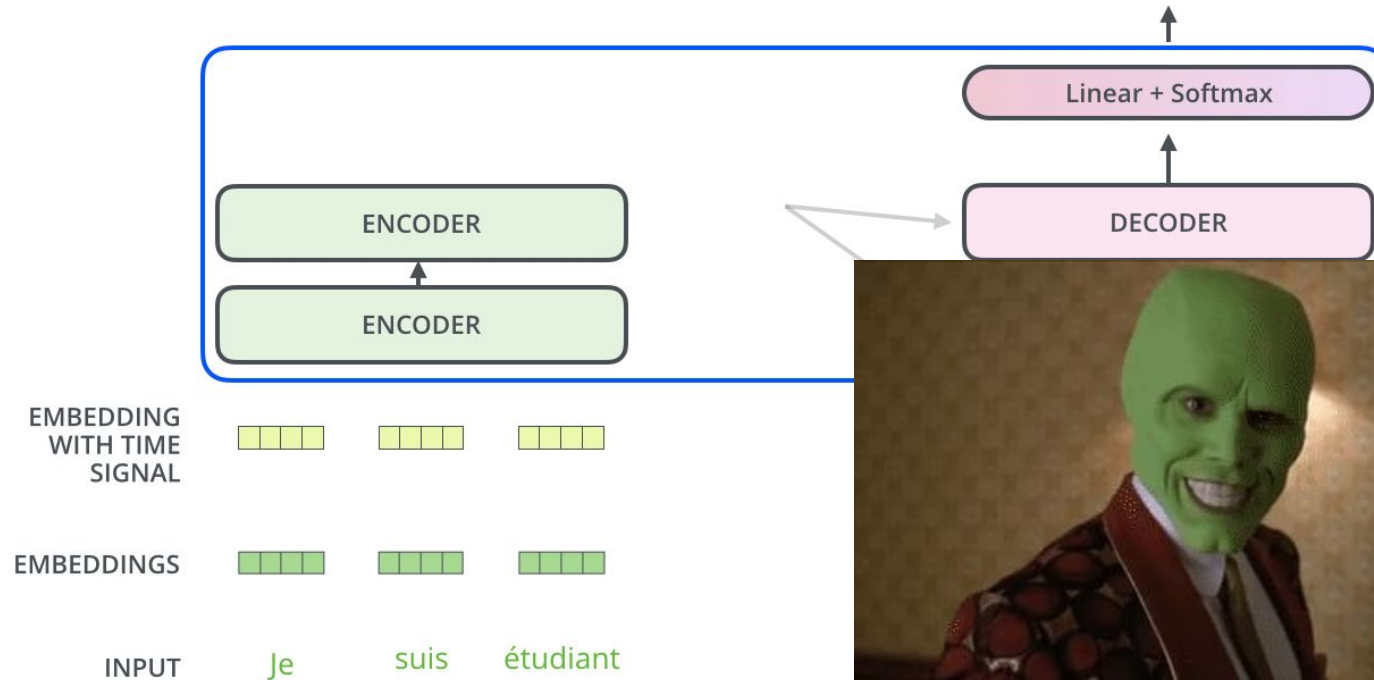
# The Decoder Side



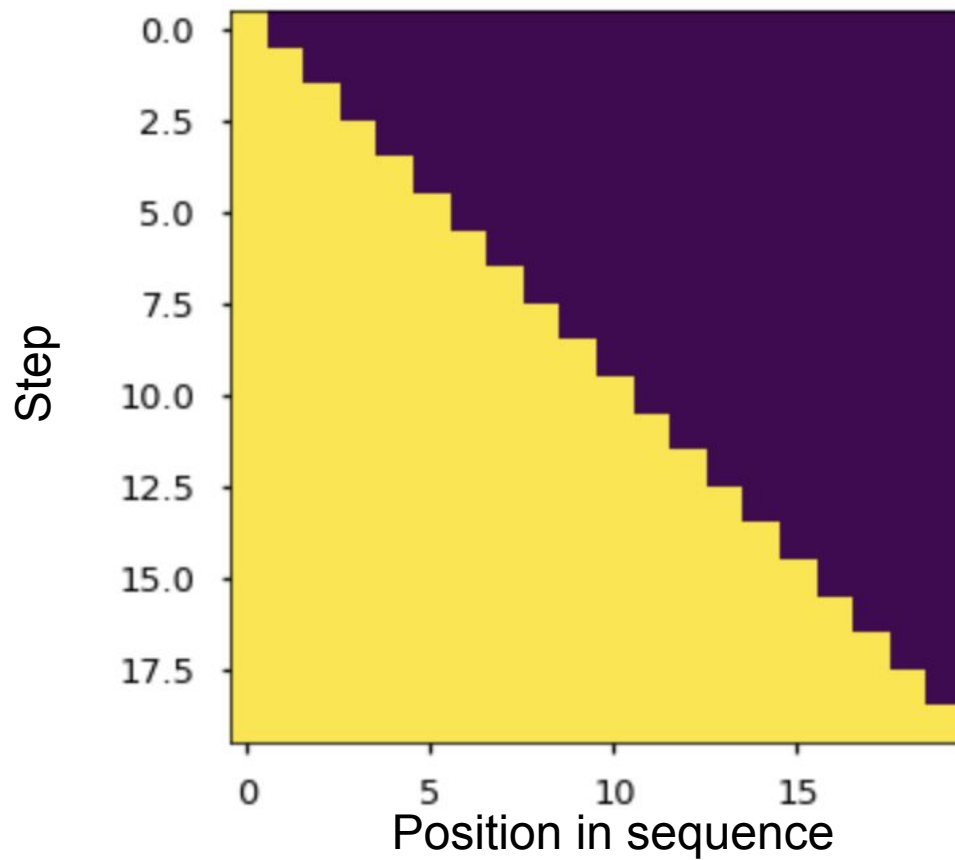
# The Decoder Side

Decoding time step: 1 2 3 4 5 6

OUTPUT

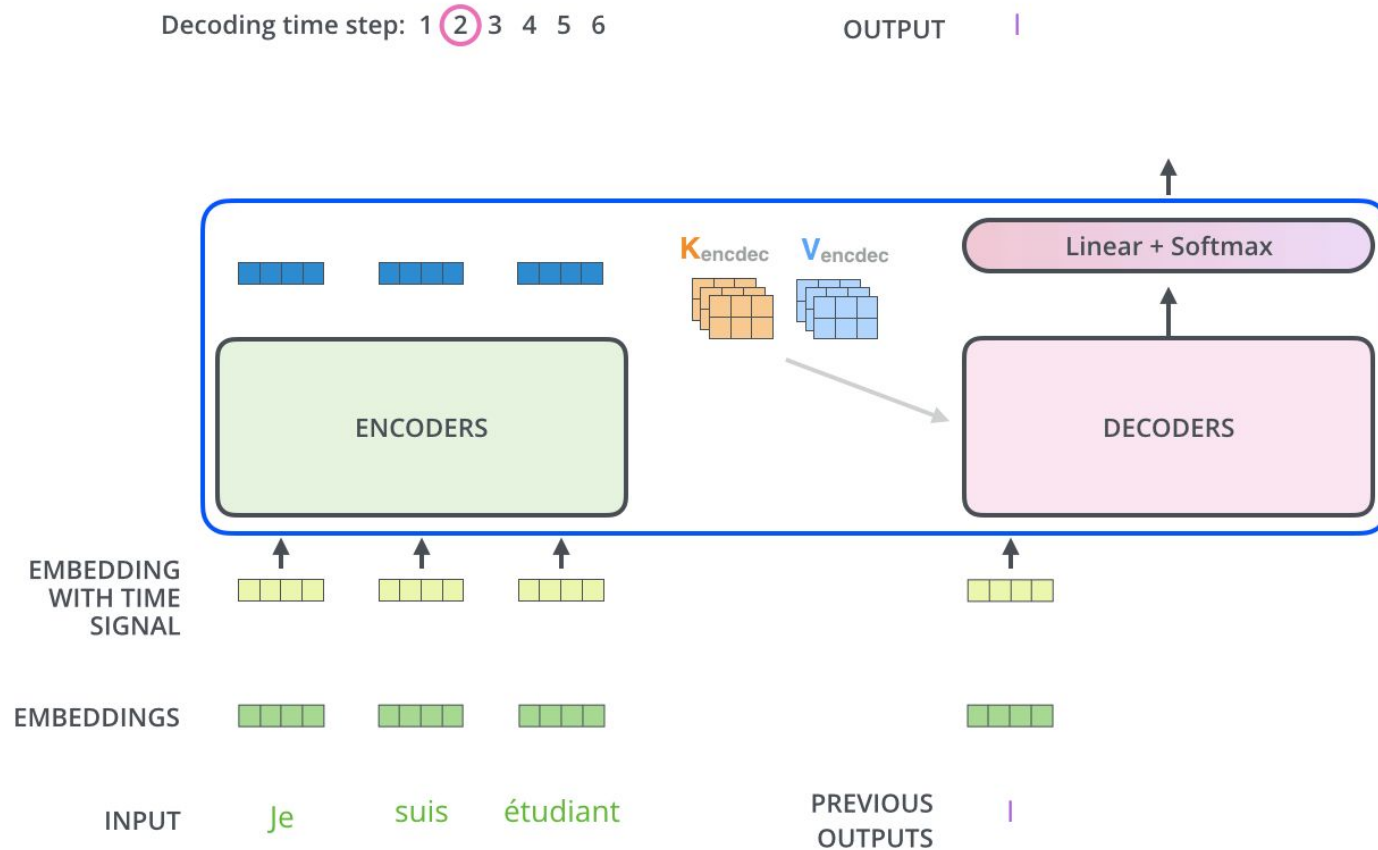


## The masked decoder input





# The Decoder Side

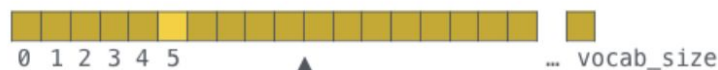


# Final Linear and Softmax Layer

Which word in our vocabulary  
is associated with this index?

Get the index of the cell  
with the highest value  
(argmax)

log\_probs



am

5

Softmax

logits

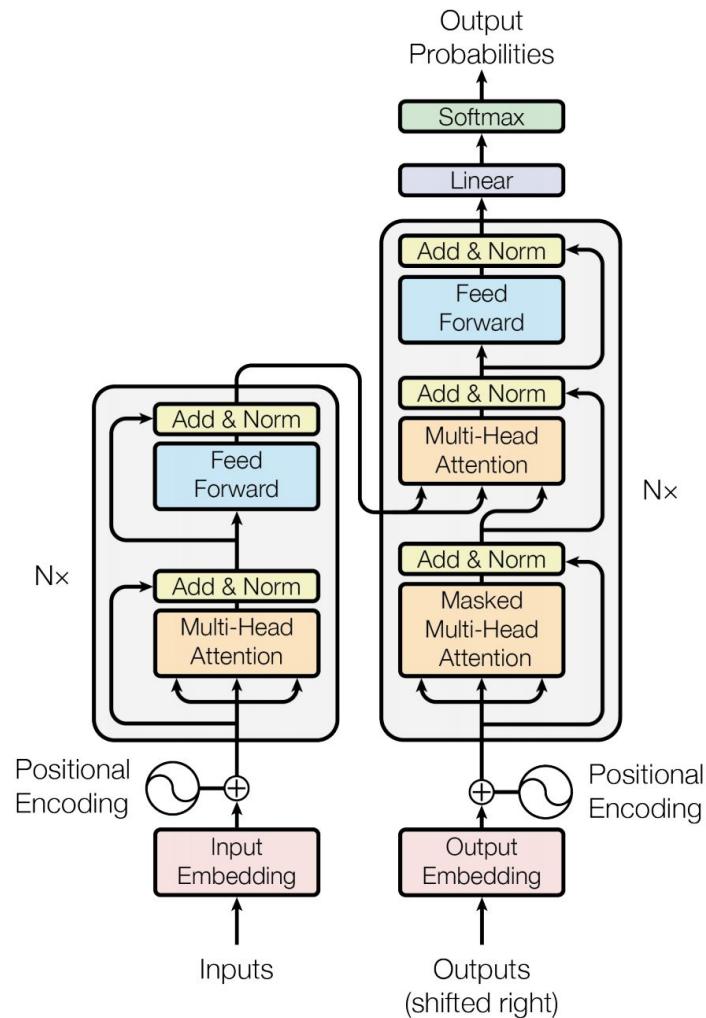


Linear

Decoder stack output



# The Transformer



- Transformer is novel and very powerful architecture
  - It is worth it to understand how Self-Attention works
  - Physical analogues can help you
- 
- Further readings are available in the repo