Lecture 04: Self-Attention & Transformer overview

Radoslav Neychev

Fall 2021, Moscow, Russia

Outline

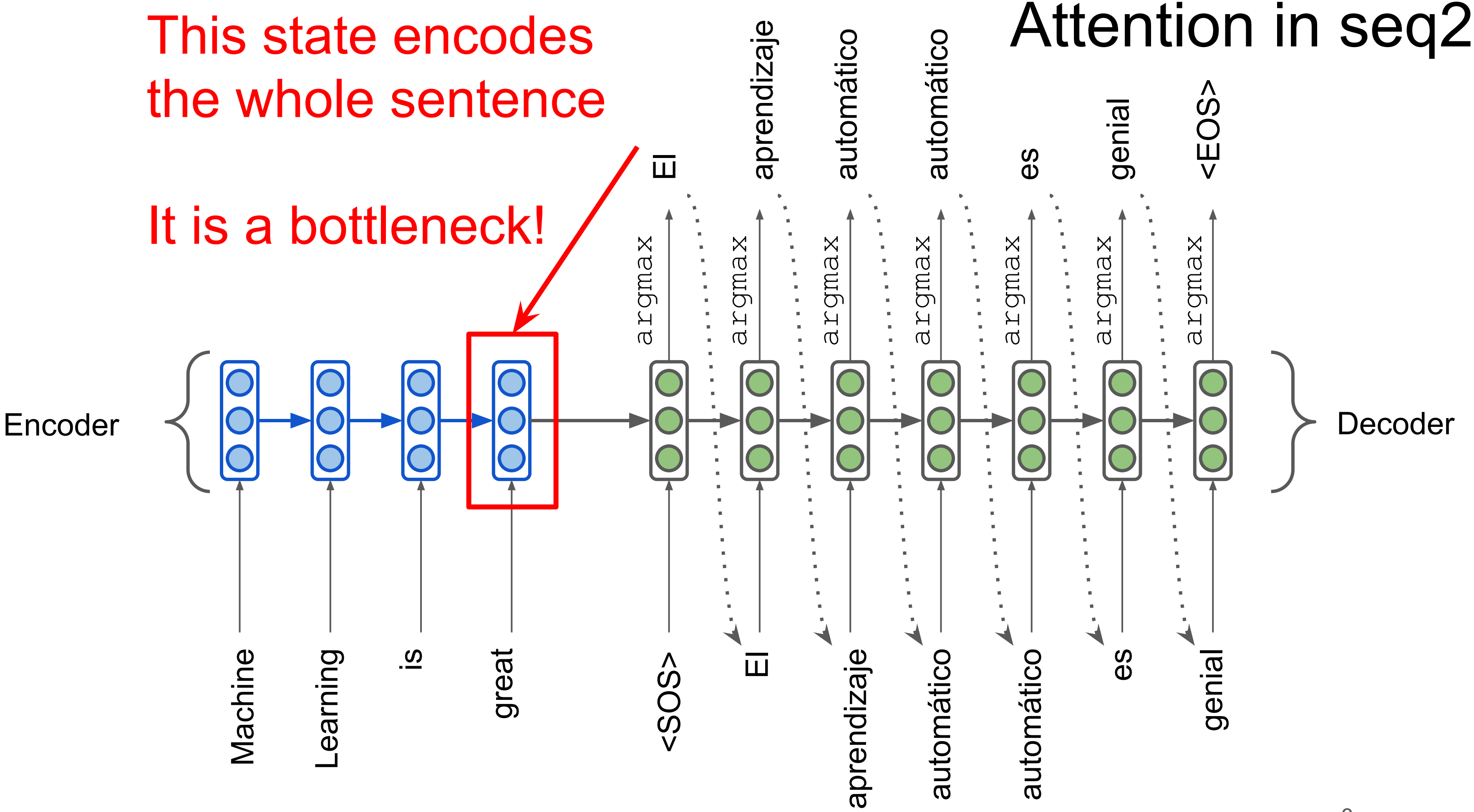
1. recap: Attention in seq2seq
2. Transformer architecture
3. Self-Attention
4. Positional encoding
5. Layer normalization
6. Decoder in Transformer

Based on: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf
https://jalammar.github.io/illustrated-transformer/

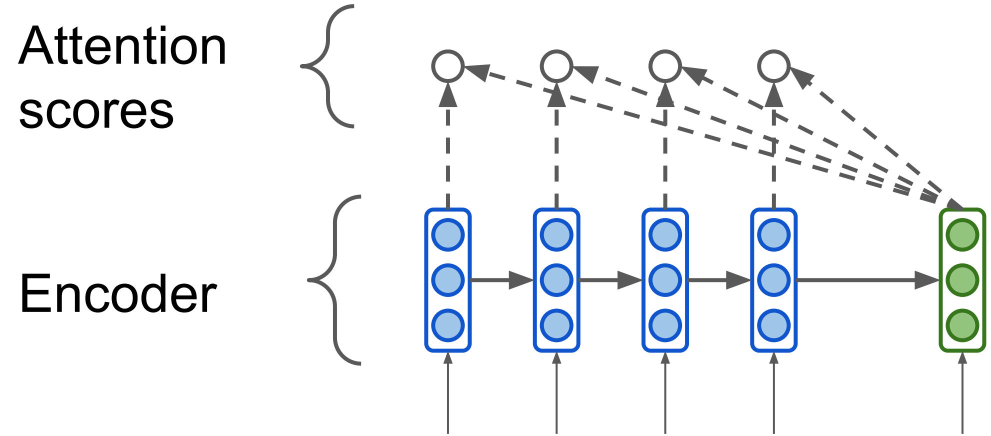
Attention in seq2seq

This state encodes the whole sentence

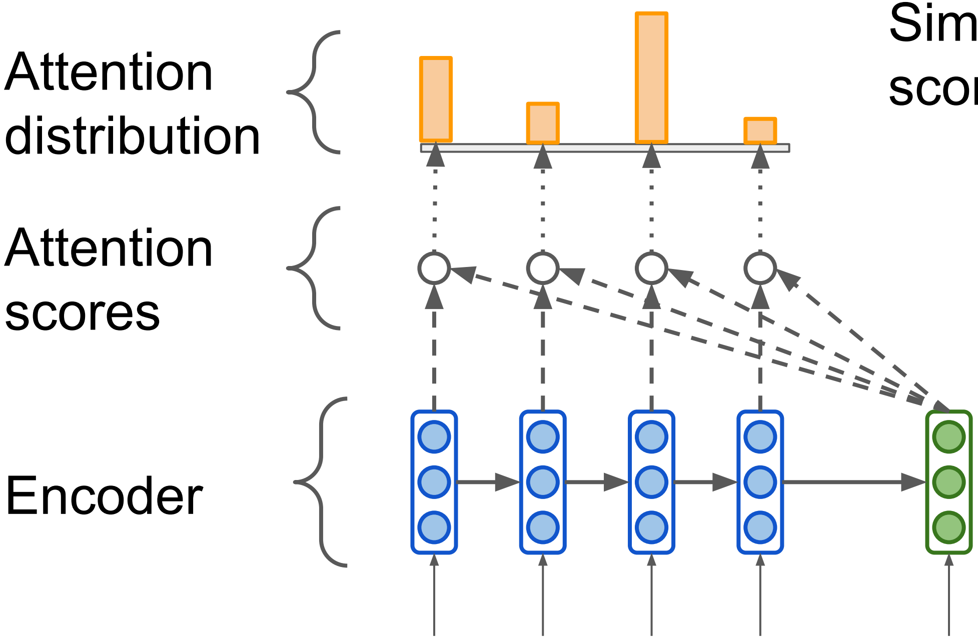
It is a bottleneck!



Seq2seq with attention



Seq2seq with attention



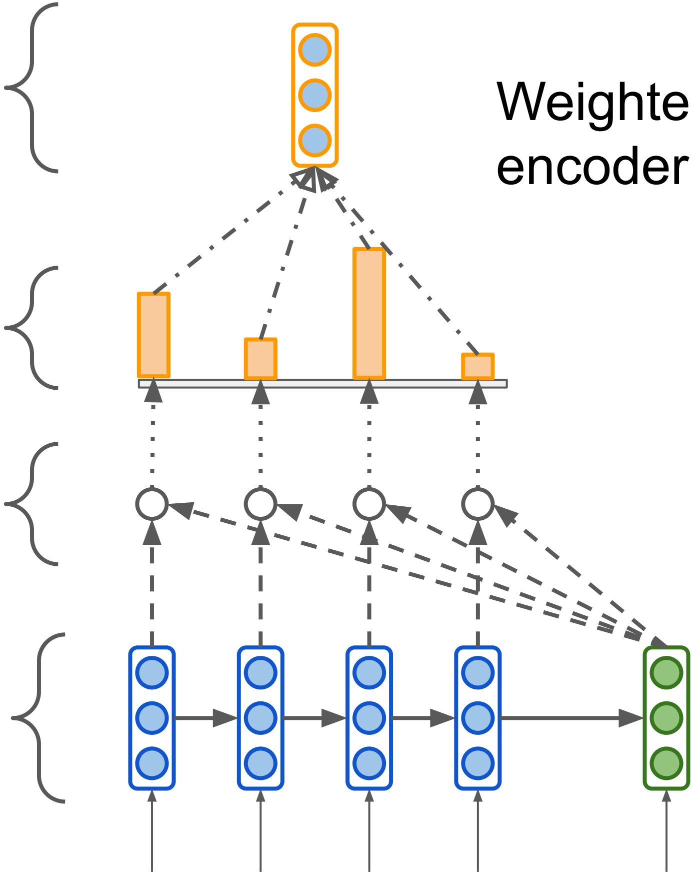
Attention distribution

Attention scores

Encoder

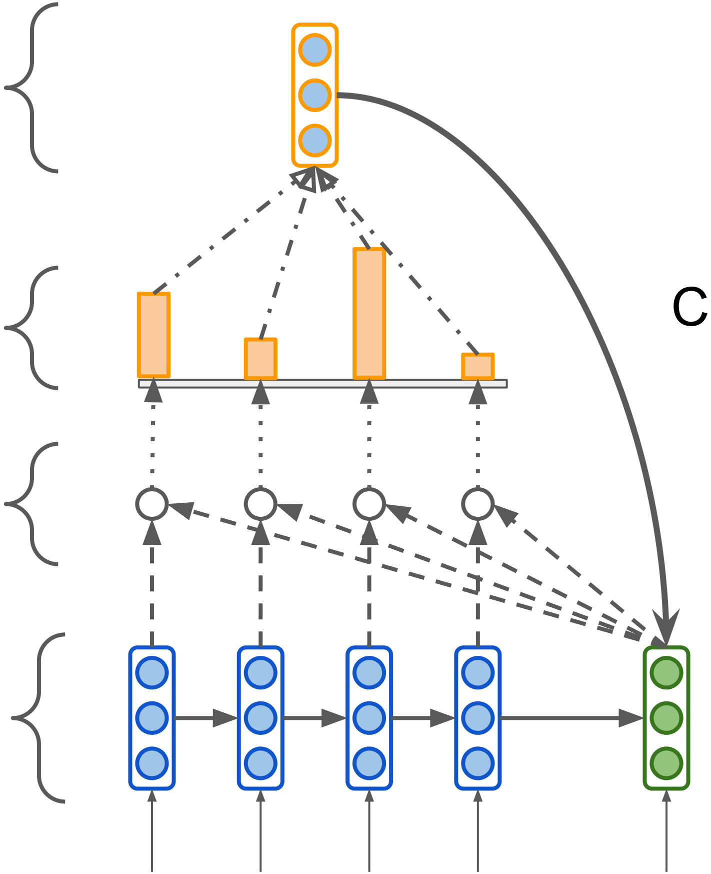
Simply apply softmax to scores

Seq2seq with attention

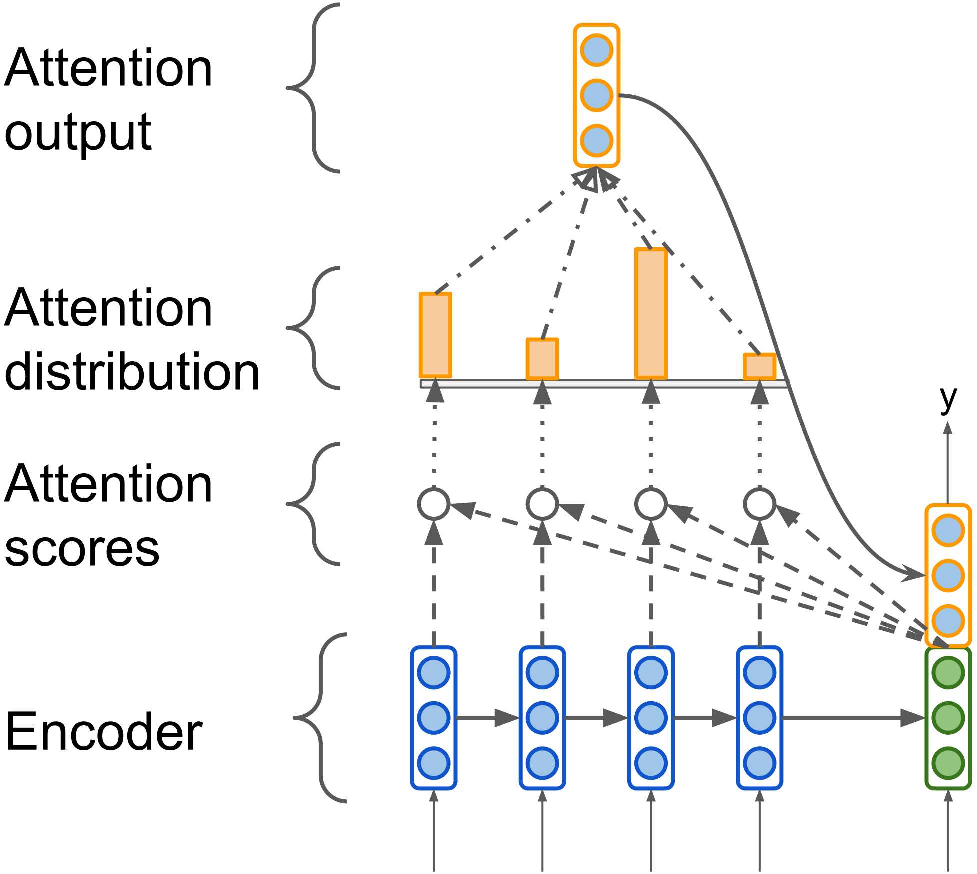


Weighted sum of all encoder states

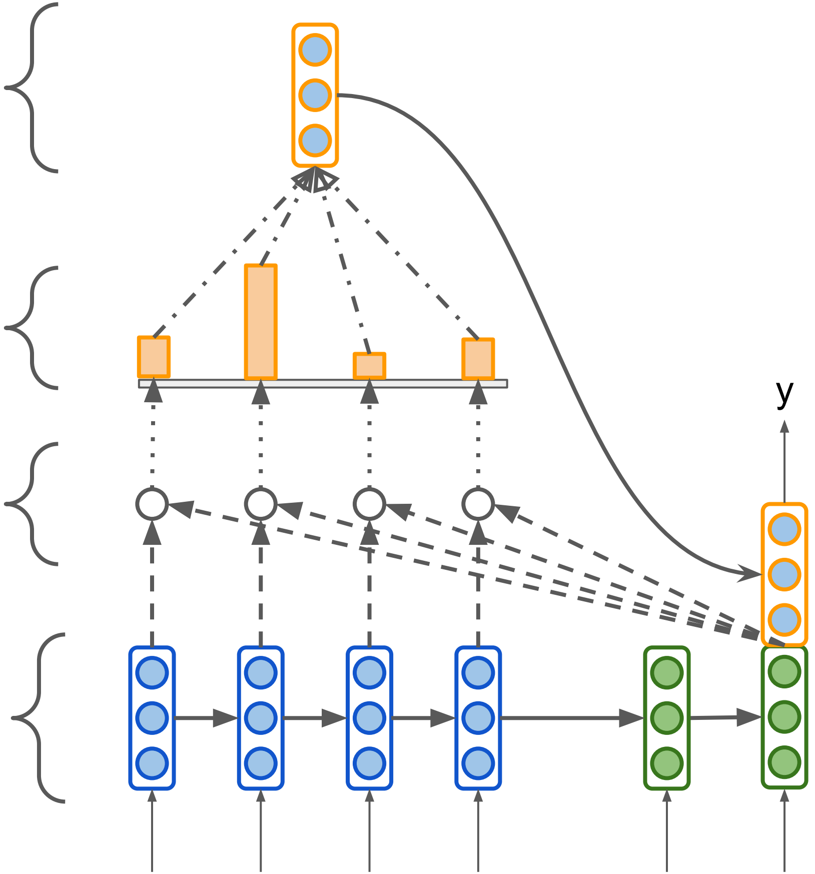
Seq2seq with attention



Seq2seq with attention



Seq2seq with attention



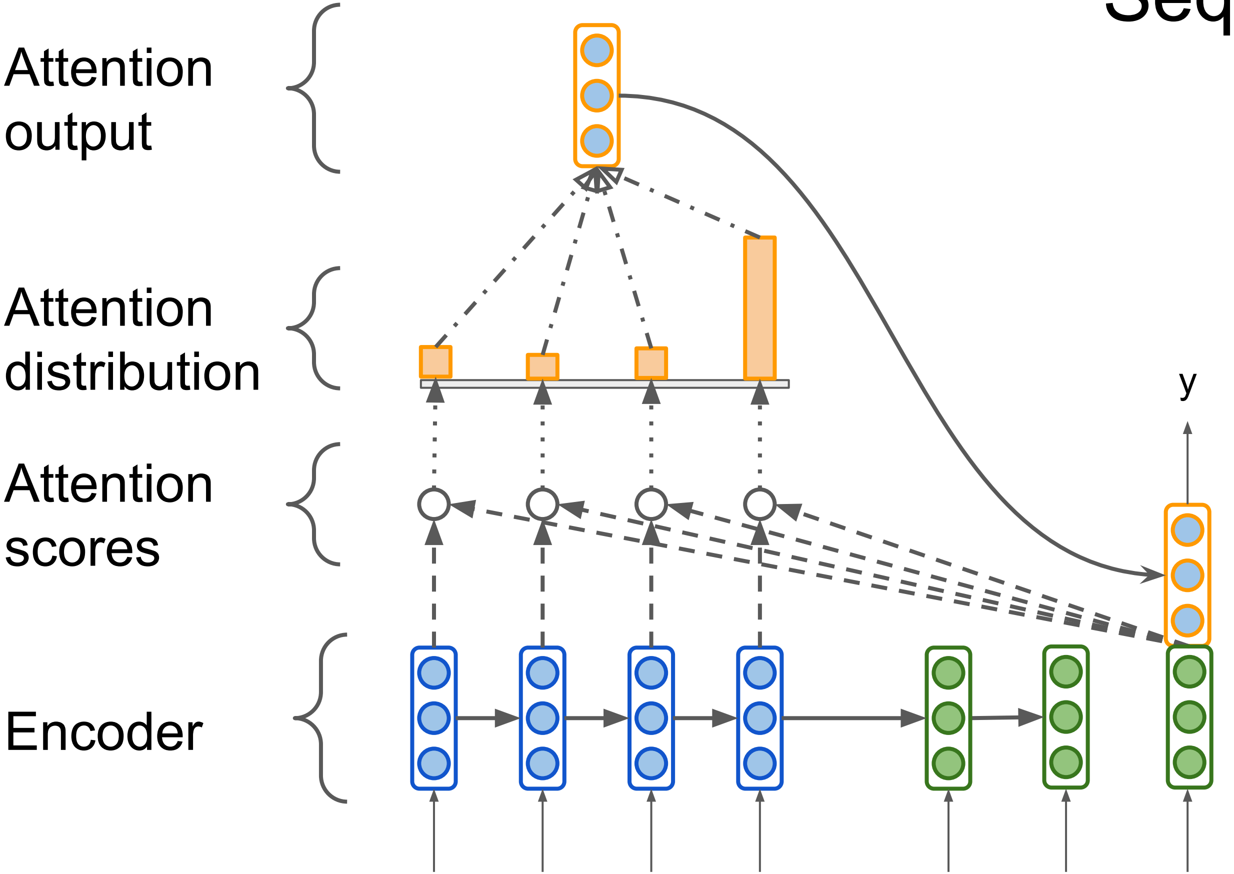
Attention output

Attention distribution

Attention scores

Encoder

Seq2seq with attention



Attention output

Attention distribution

Attention scores

Encoder

Attention in equations

Denote encoder hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_N \in \mathbb{R}^k$

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

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

Then the attention vector is a linear combination of encoder states

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

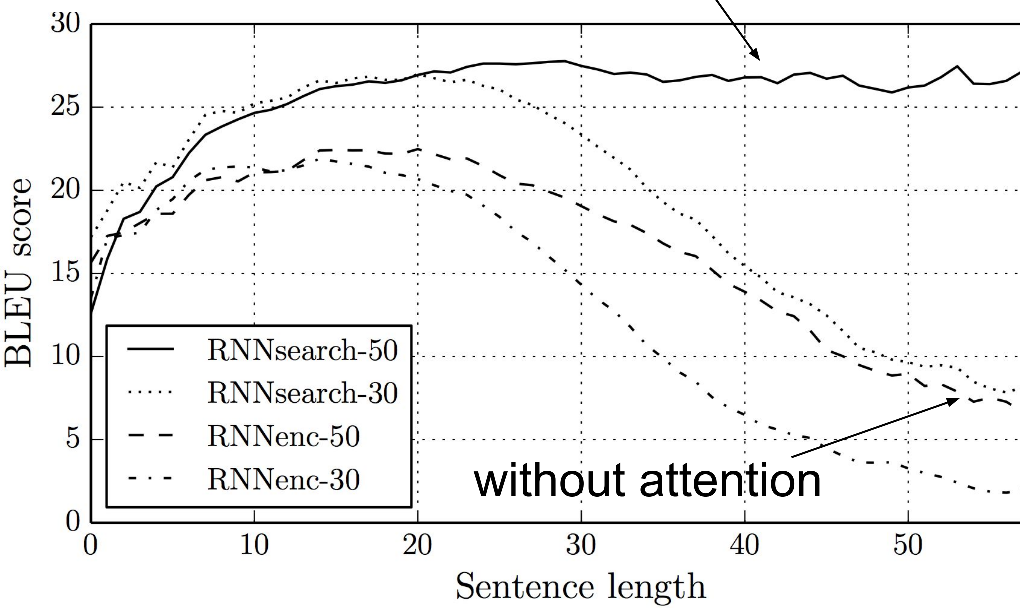
Attention variants

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

Attention advantages

- "Free" word alignment
- Better results on long sequences

with attention



without attention

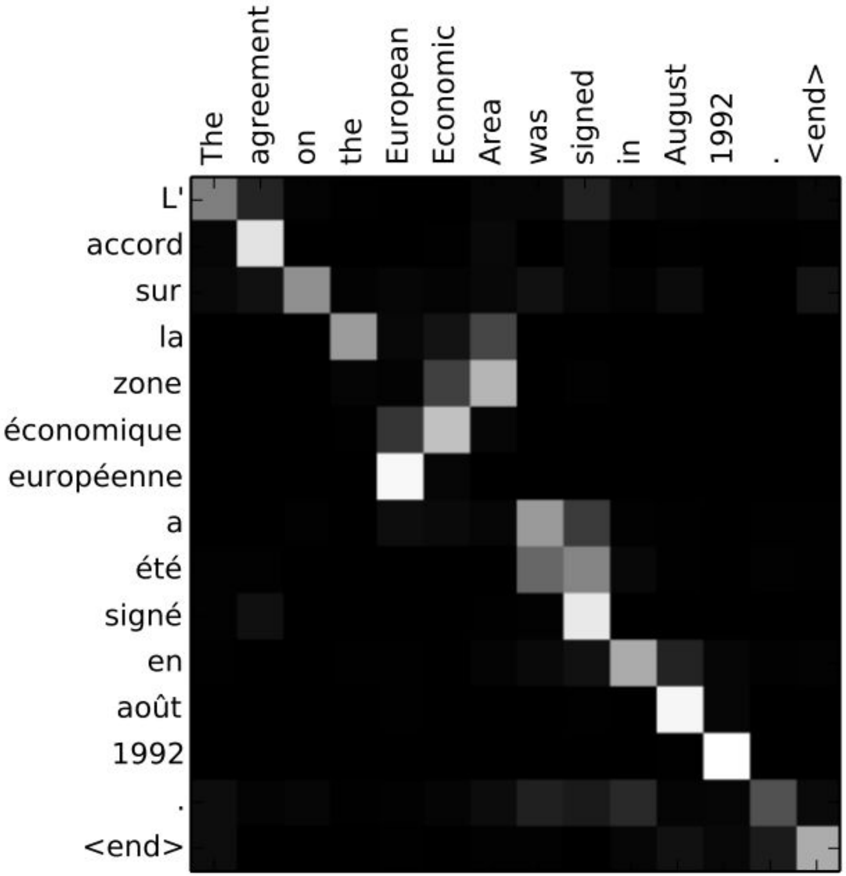


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

The Transformer

The Transformer

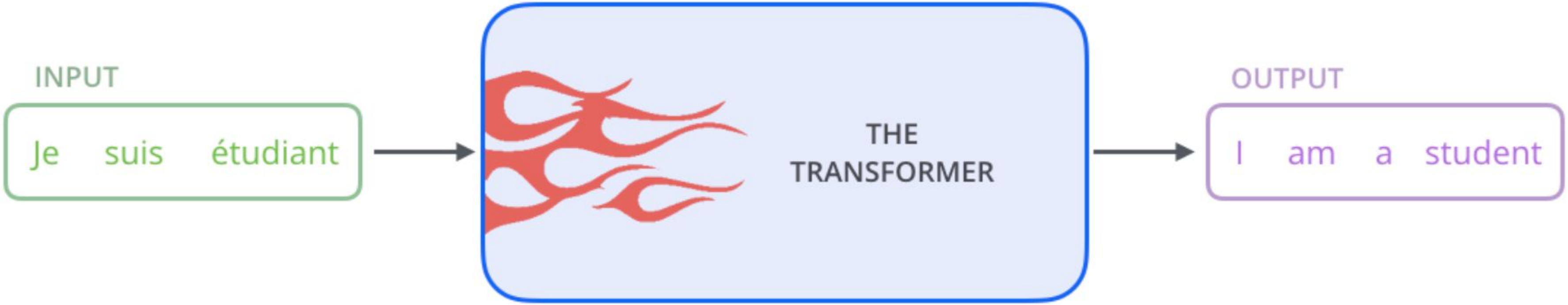


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

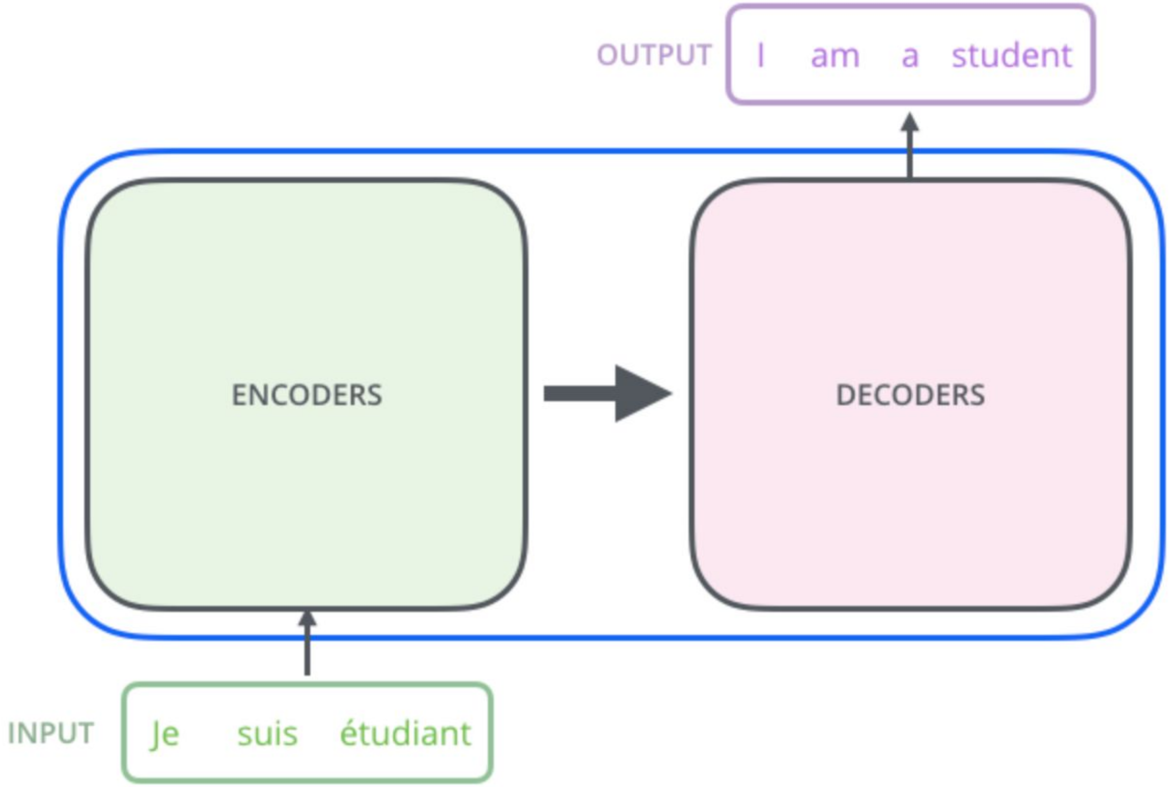


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

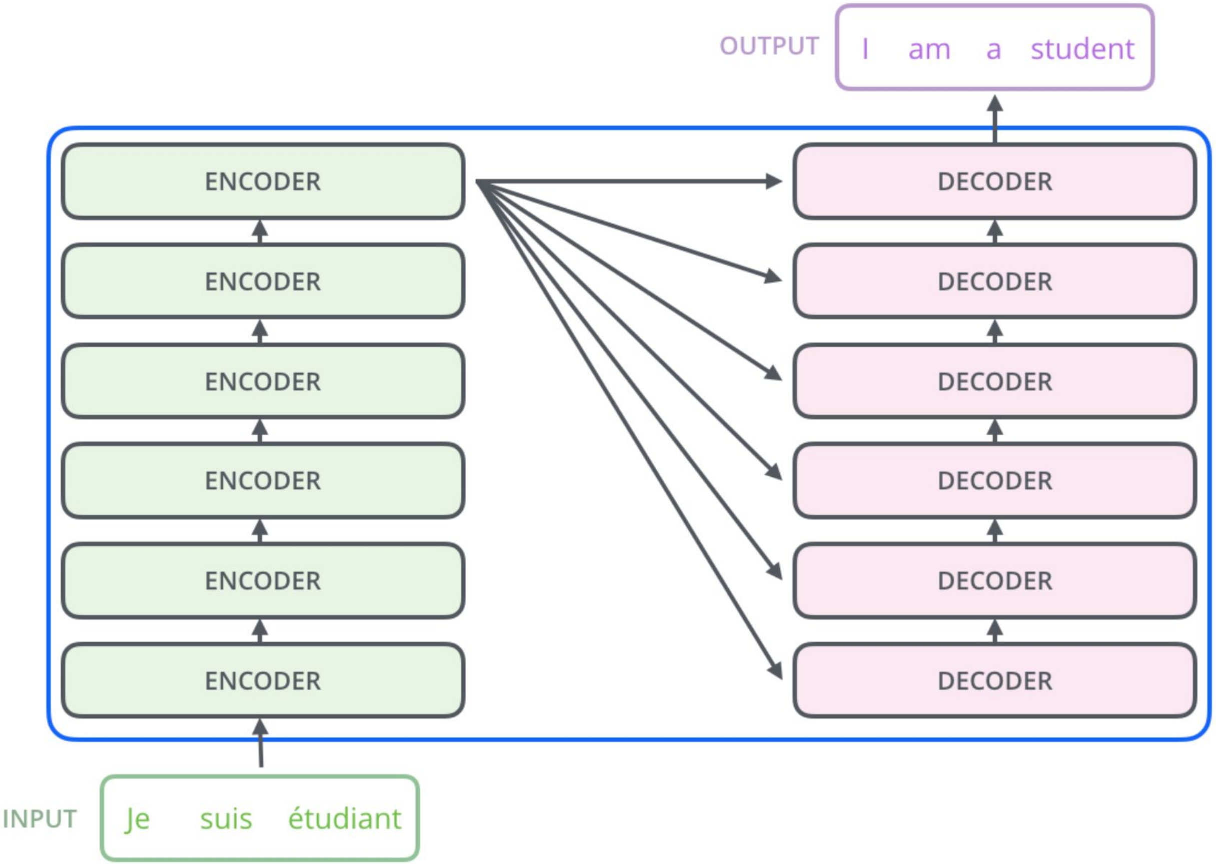
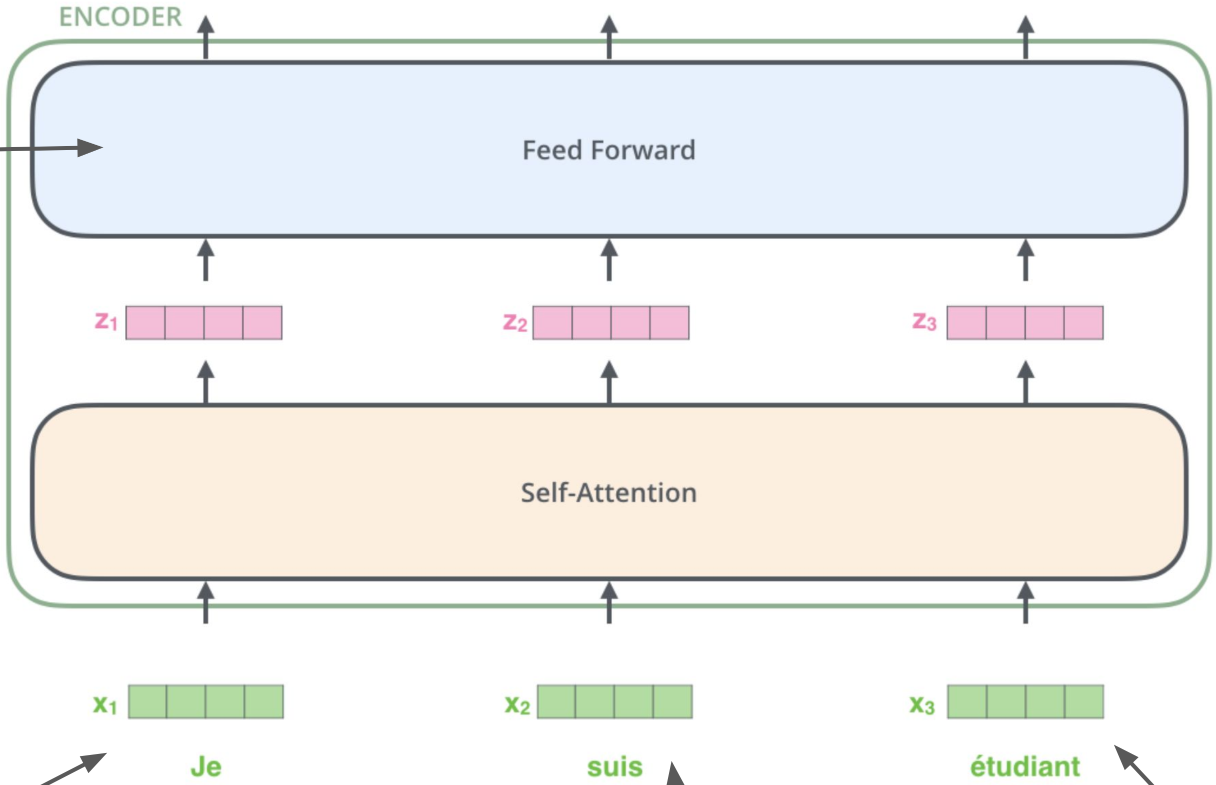


Image source: https://jalammar.github.io/illustrated-transformer/

The Encoder Side

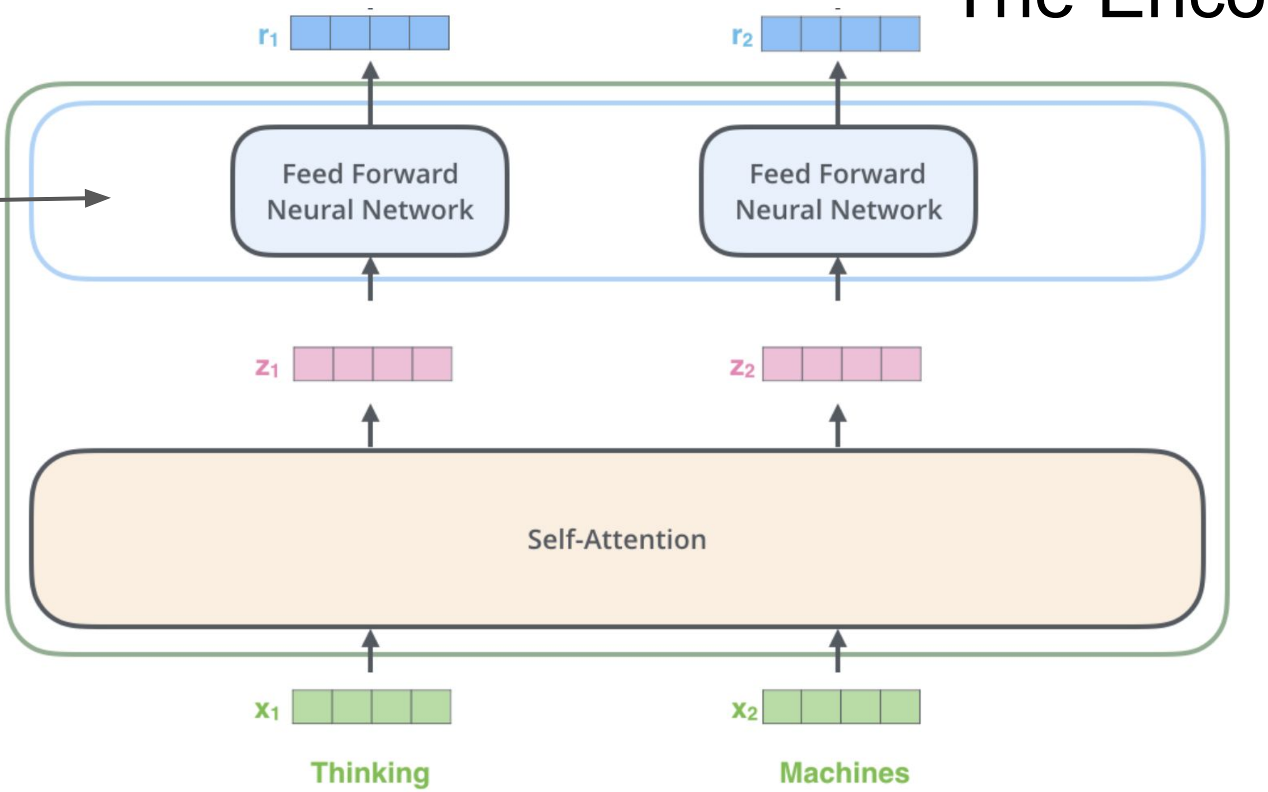
Can be parallelized



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

Image source: https://jalammar.github.io/illustrated-transformer/

The Encoder Side



Can be parallelized

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

Image source: https://jalammar.github.io/illustrated-transformer/

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

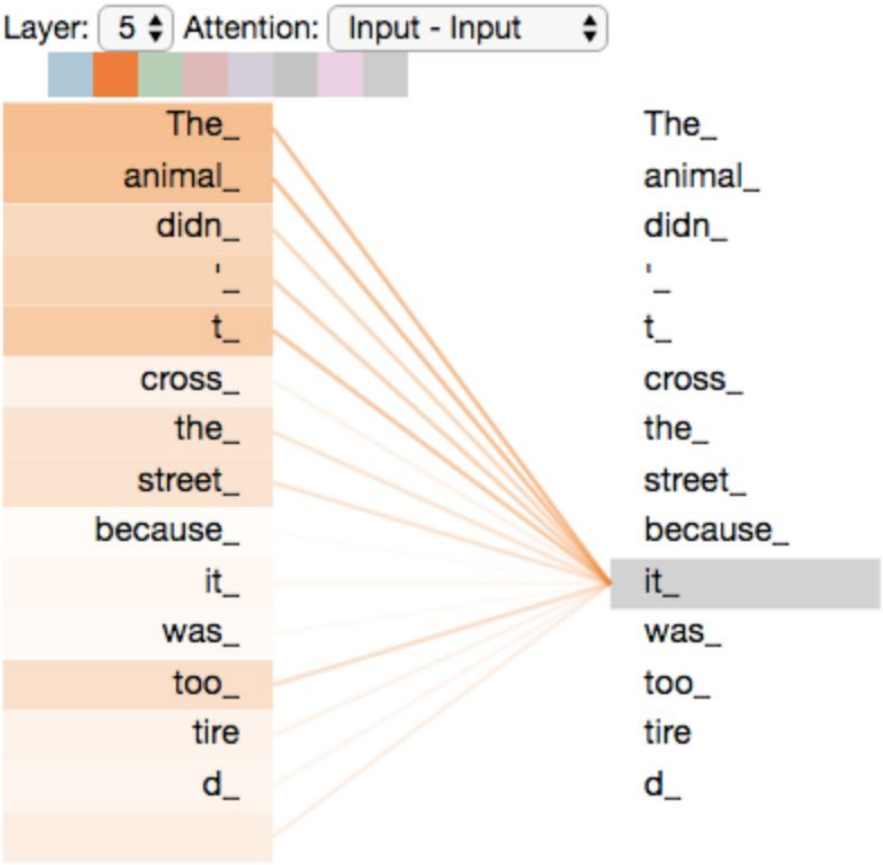
Self-Attention at a High Level

”`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



Layer: 5 Attention: Input - Input

Self-Attention: detailed explanation

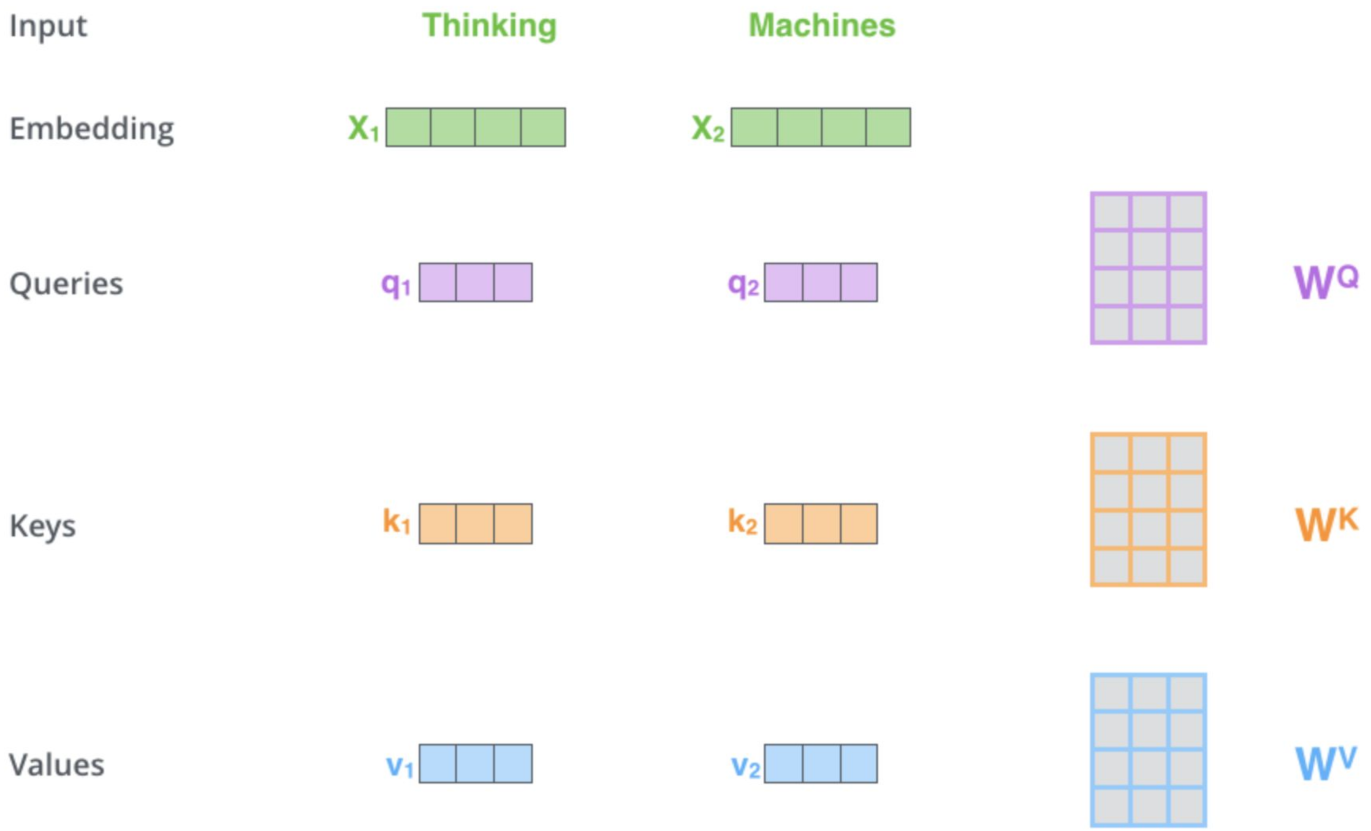


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 1:

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

from each of the encoder's input vectors

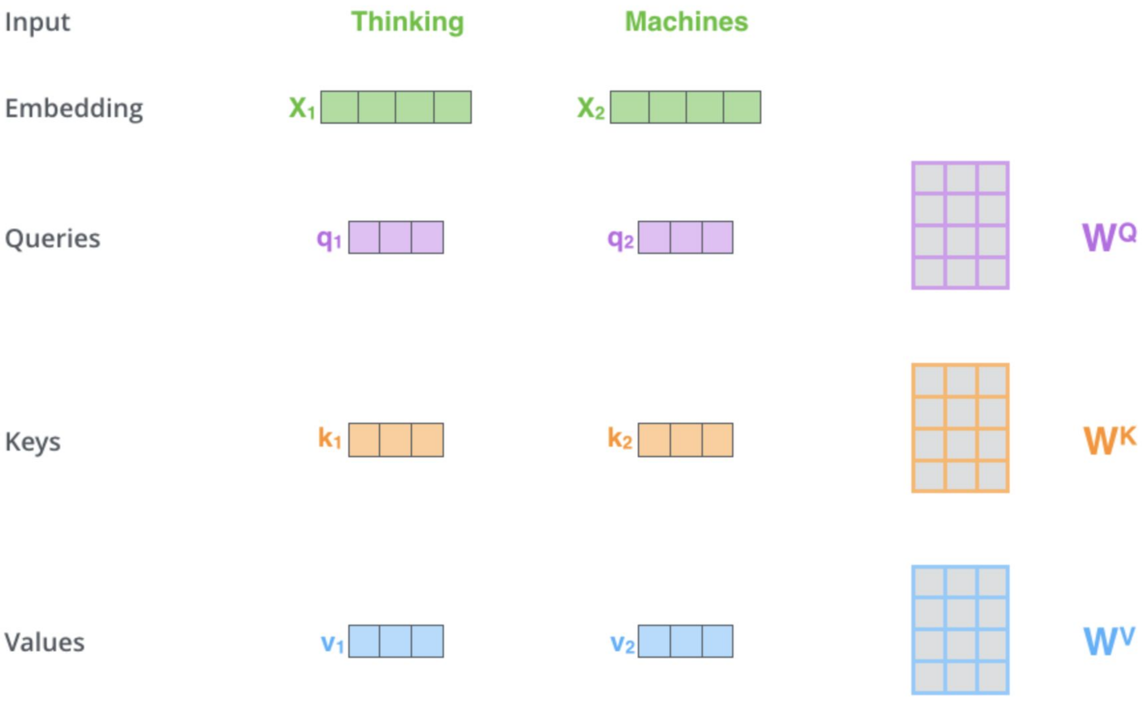


Image source: https://jalammar.github.io/illustrated-transformer/

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)

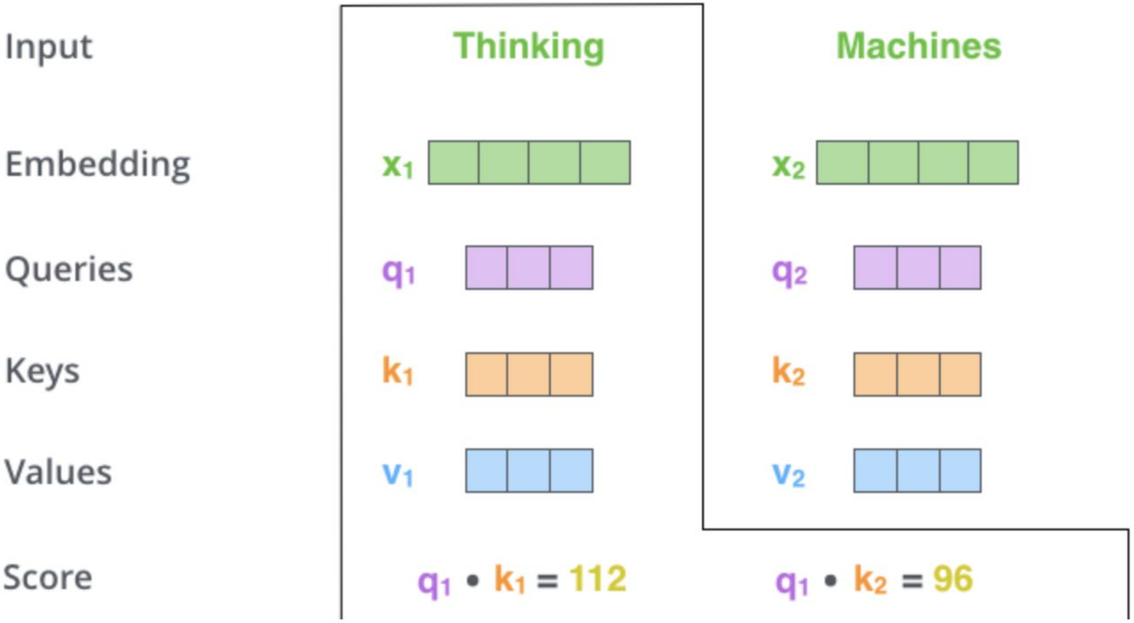


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 3:

divide the scores by 8

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

STEP 4:

softmax

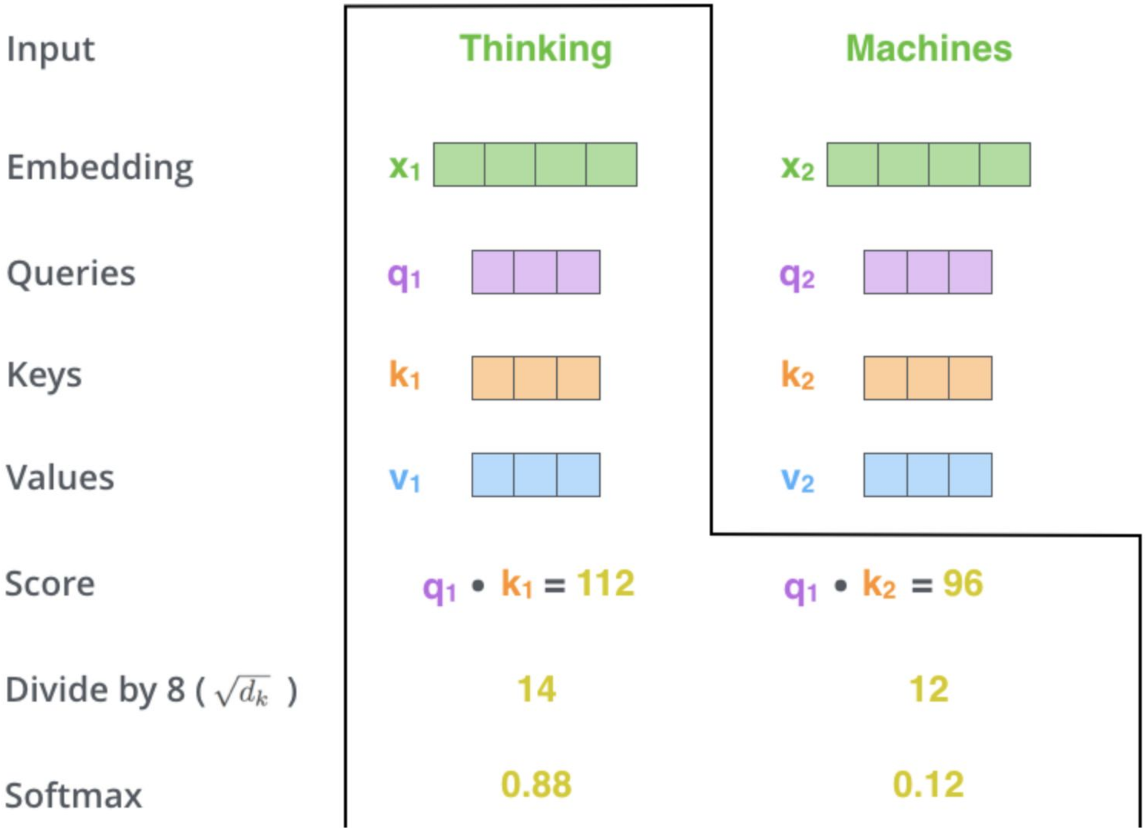


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors

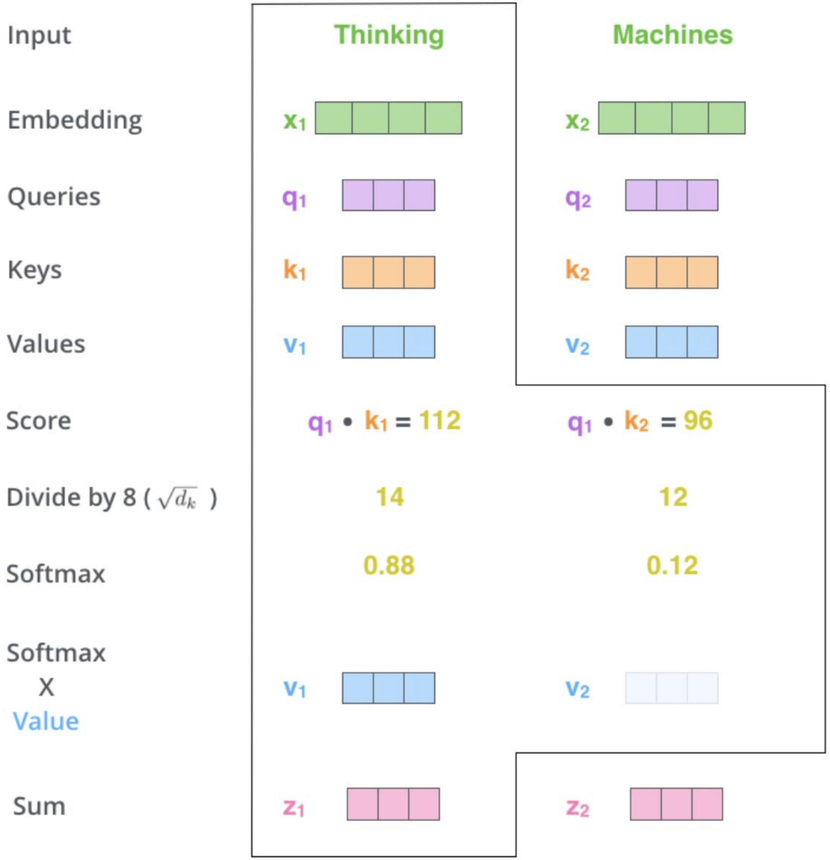
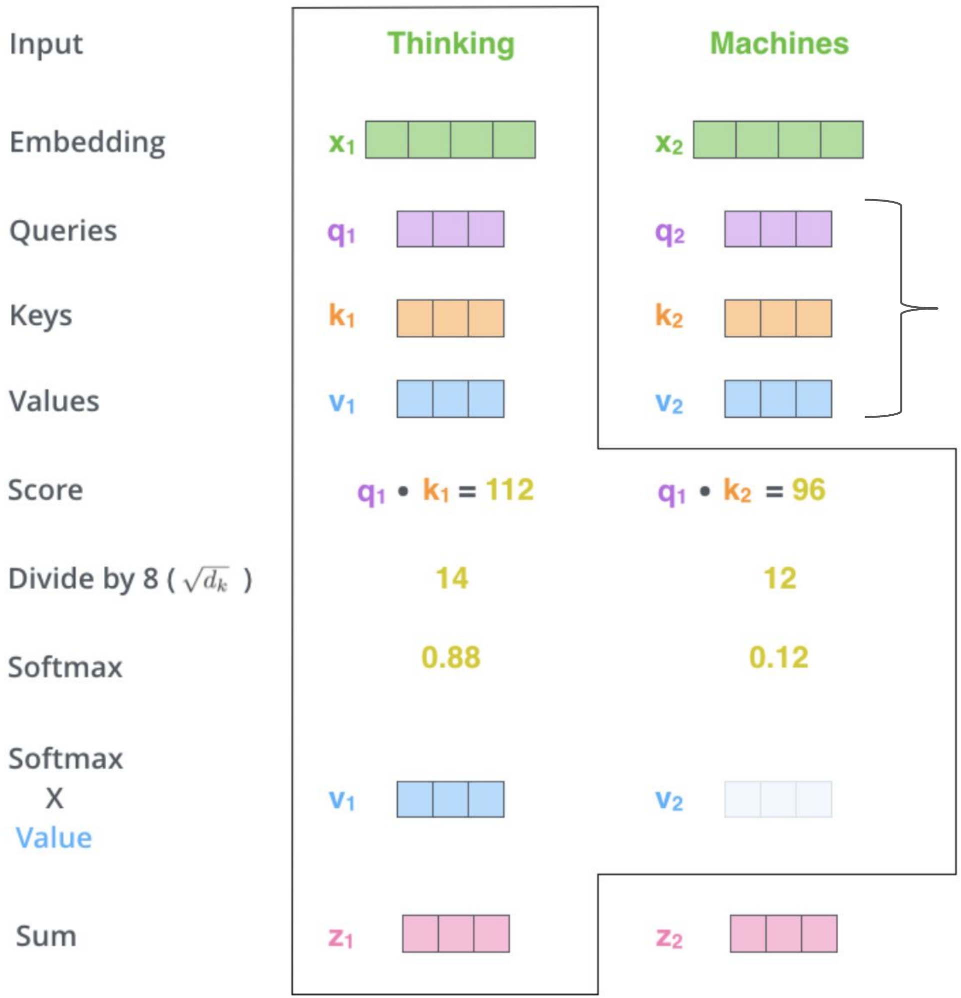


Image source: https://jalammar.github.io/illustrated-transformer/

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

Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

Pack embeddings into matrix **X**

Multiply **X** by weight matrices we've trained (**Wk, Wq, Wv**)

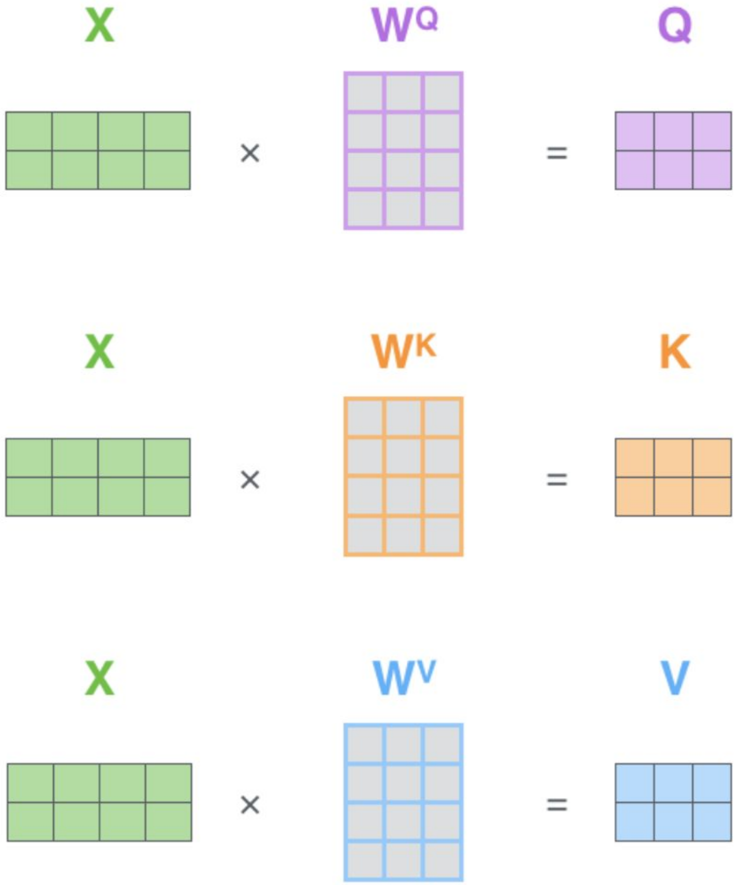


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

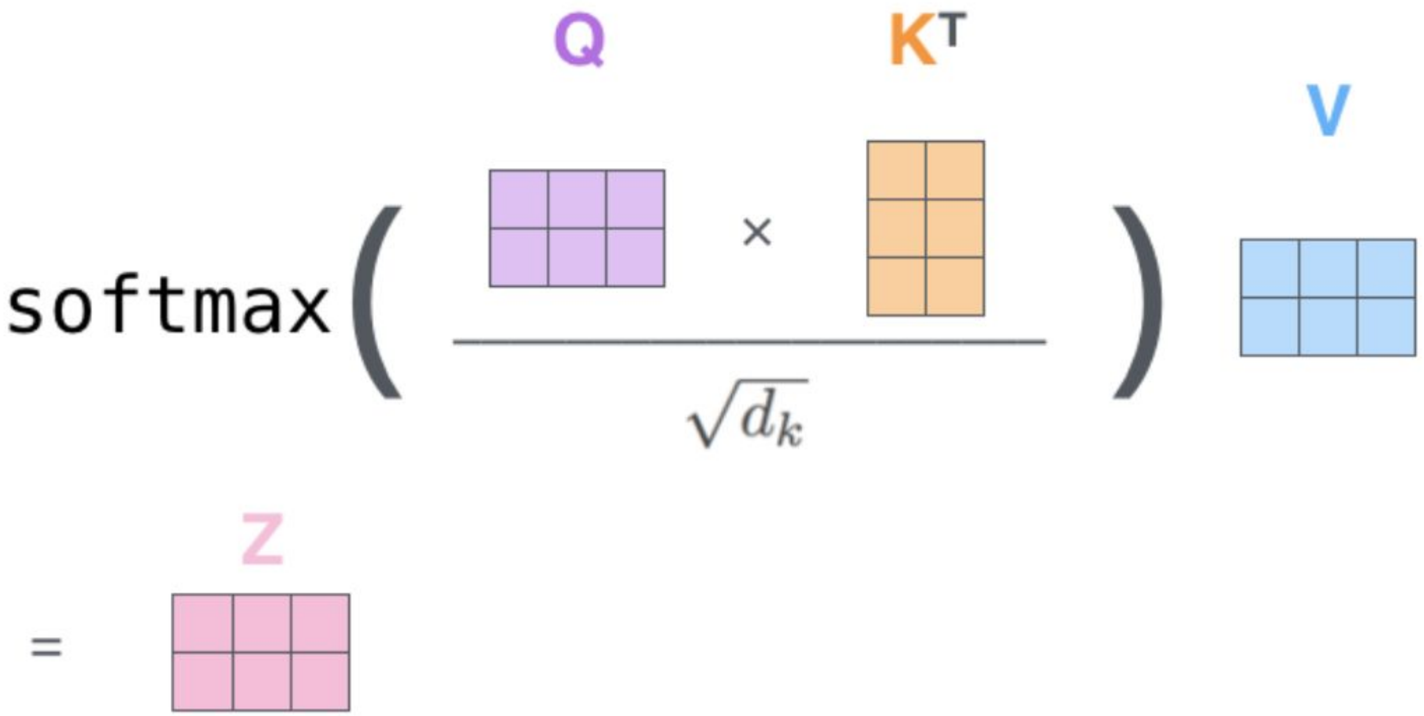


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

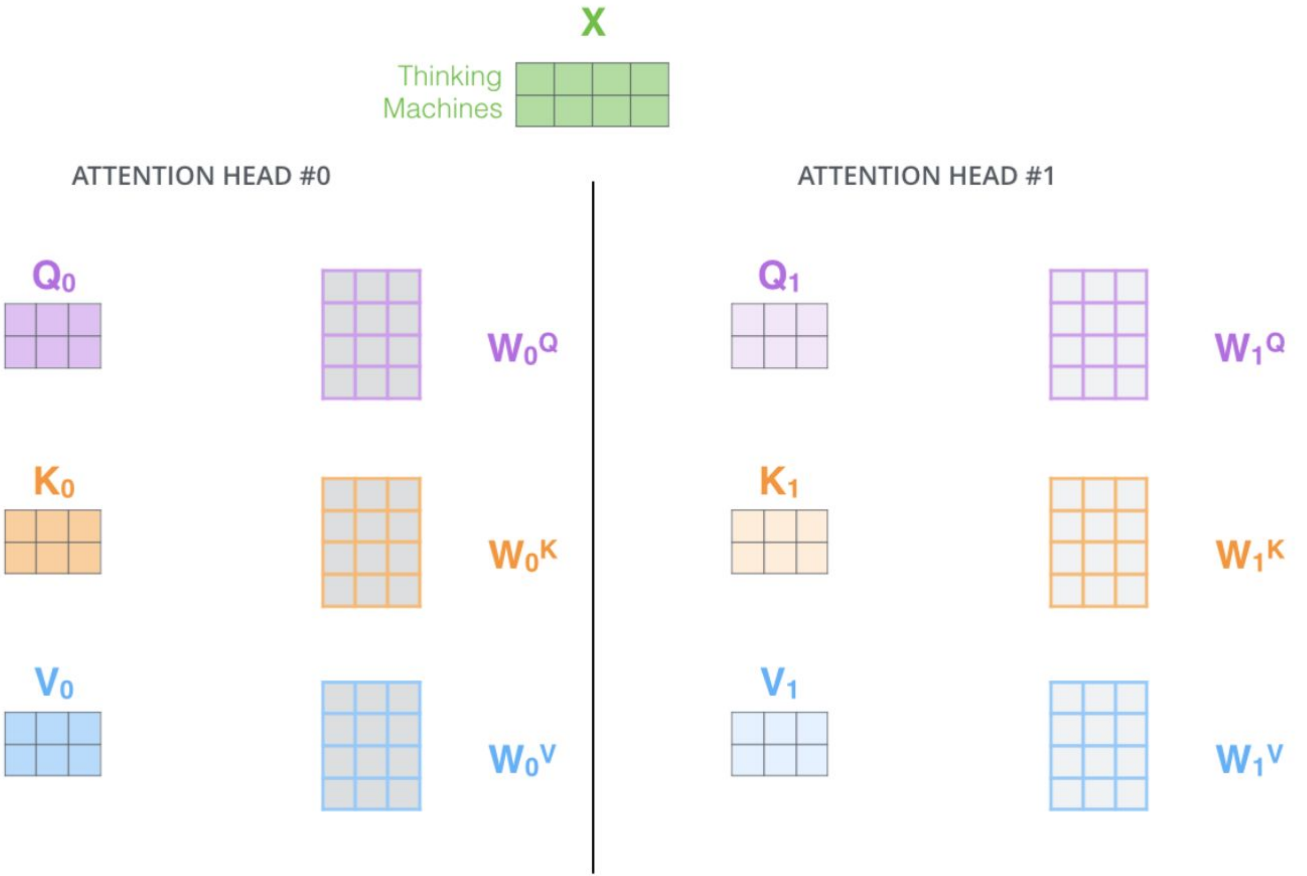


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

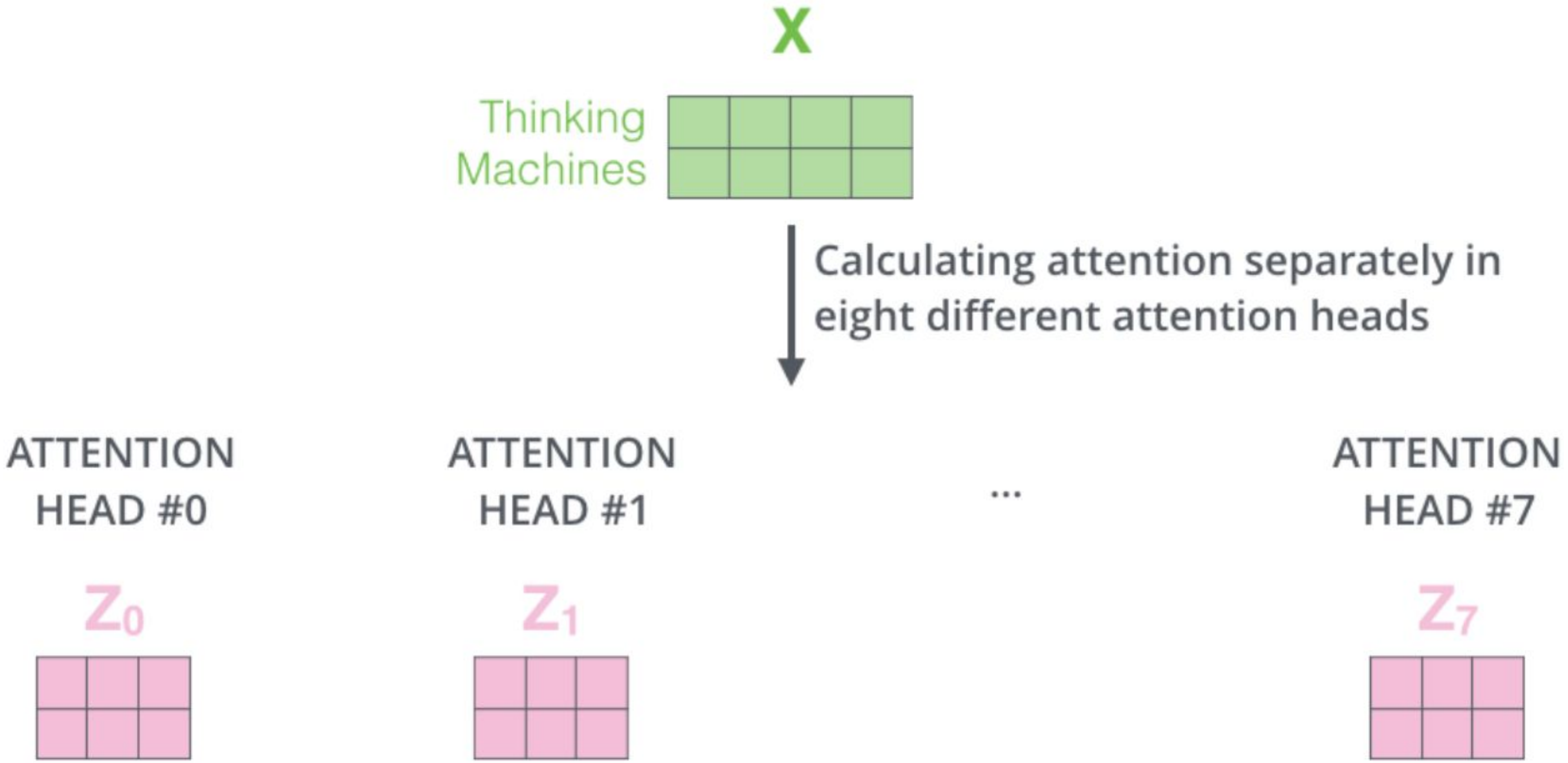
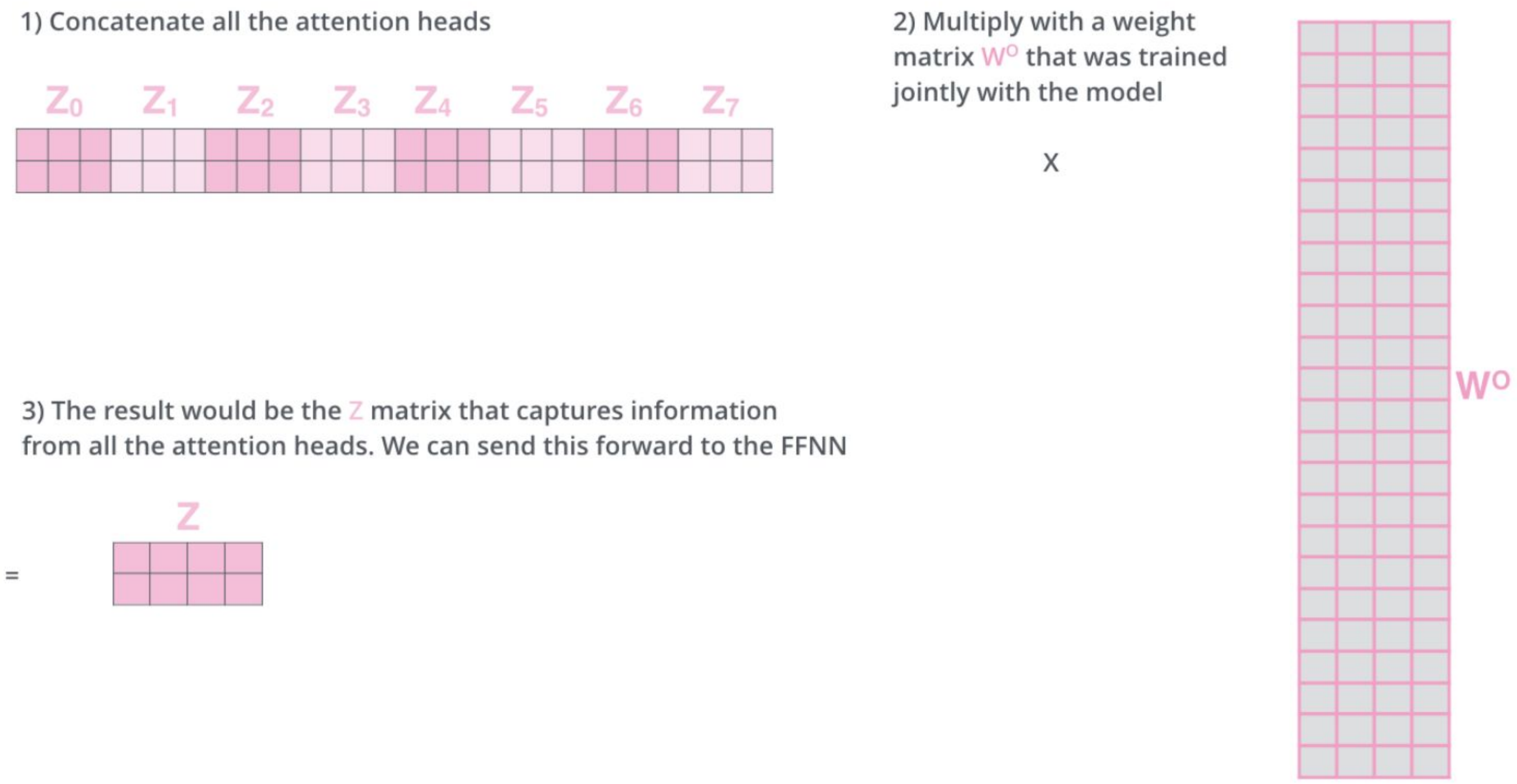


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

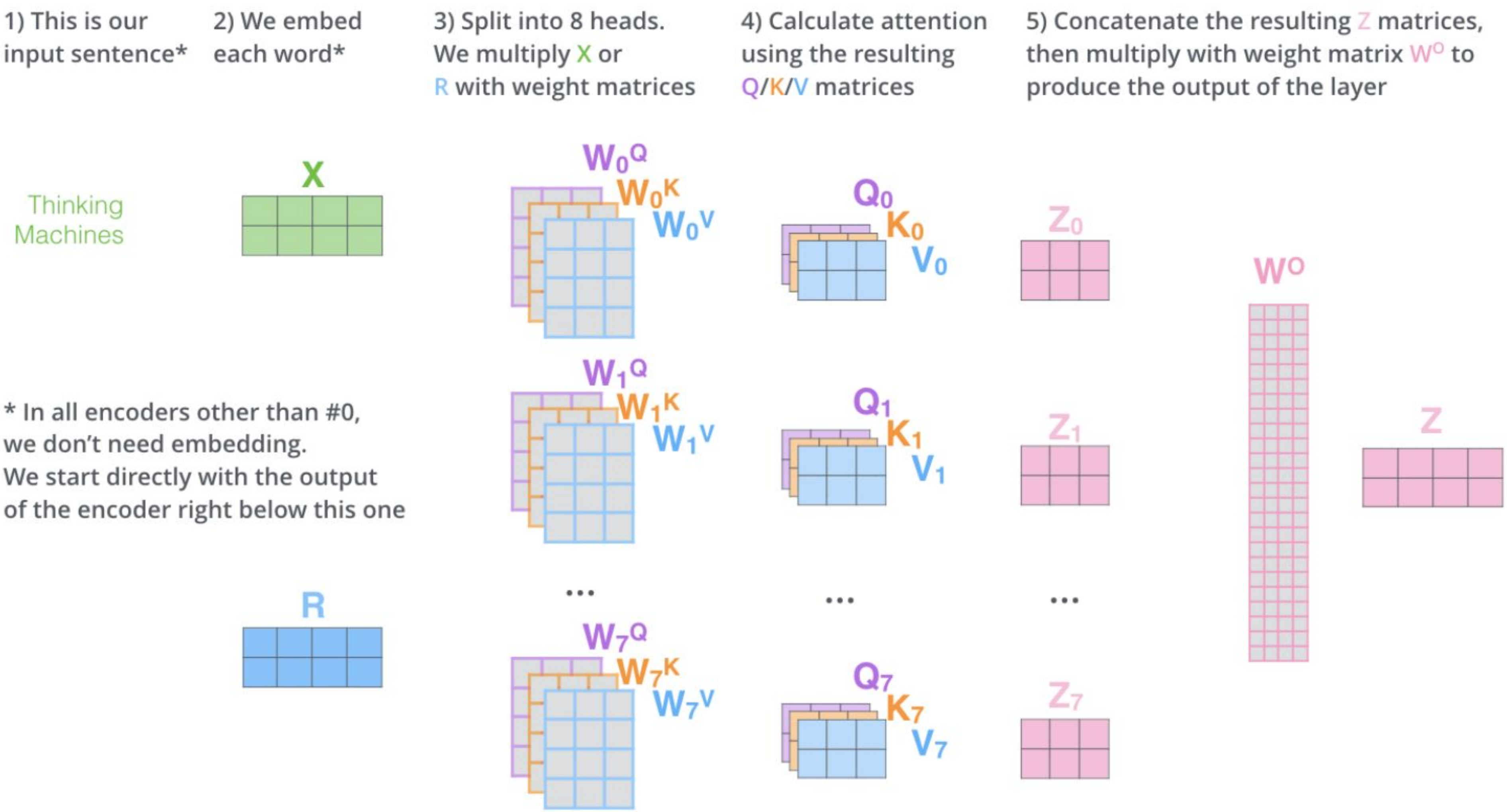


1) Concatenate all the attention heads

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

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

Image source: https://jalammar.github.io/illustrated-transformer/



1) This is our input sentence*

2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

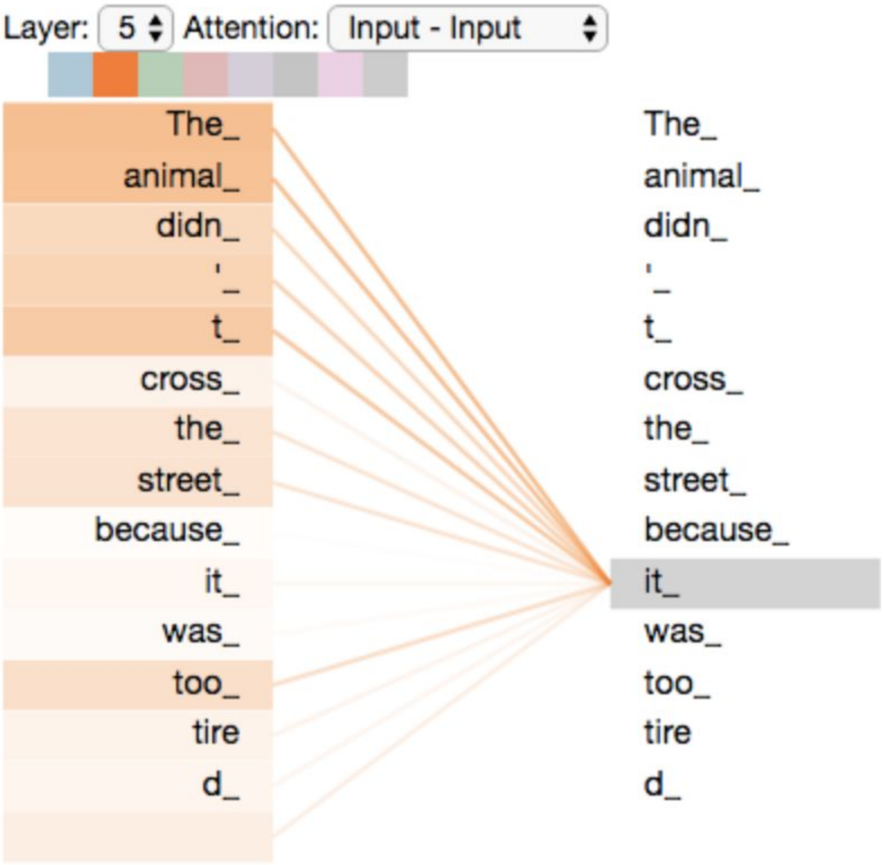
4) Calculate attention using the resulting Q/K/V matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

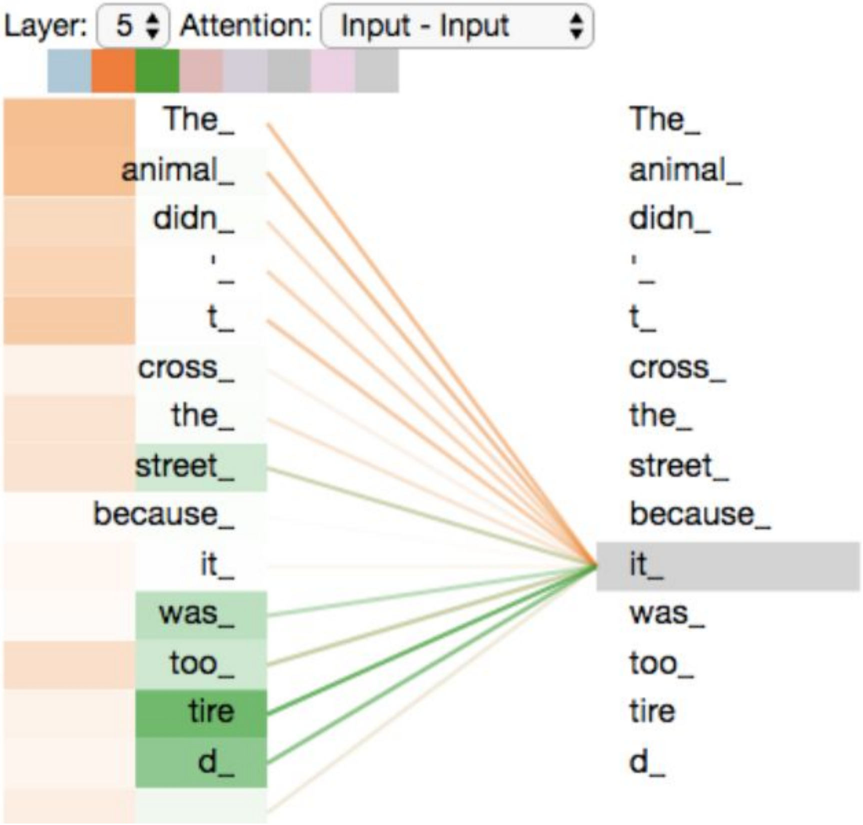
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

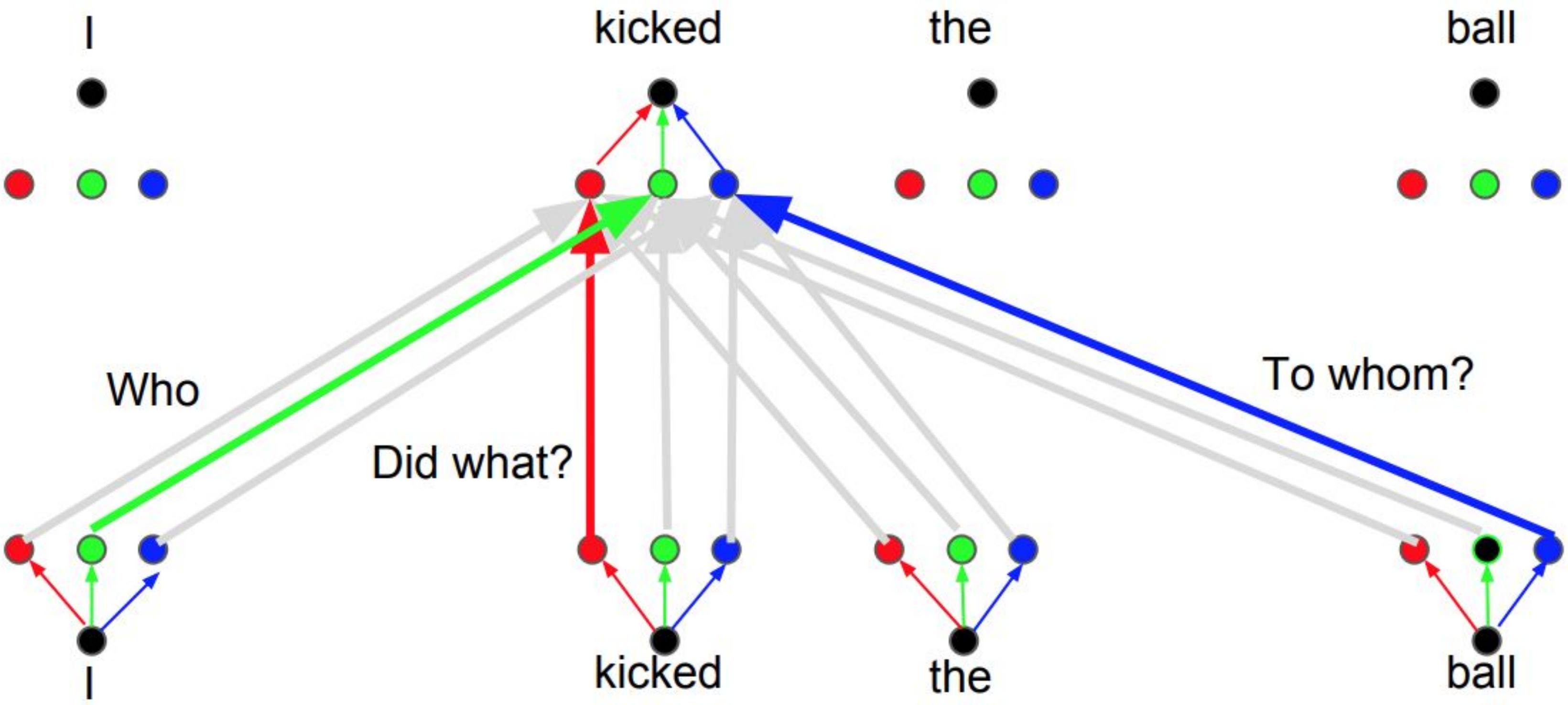


Layer: 5 Attention: Input - Input

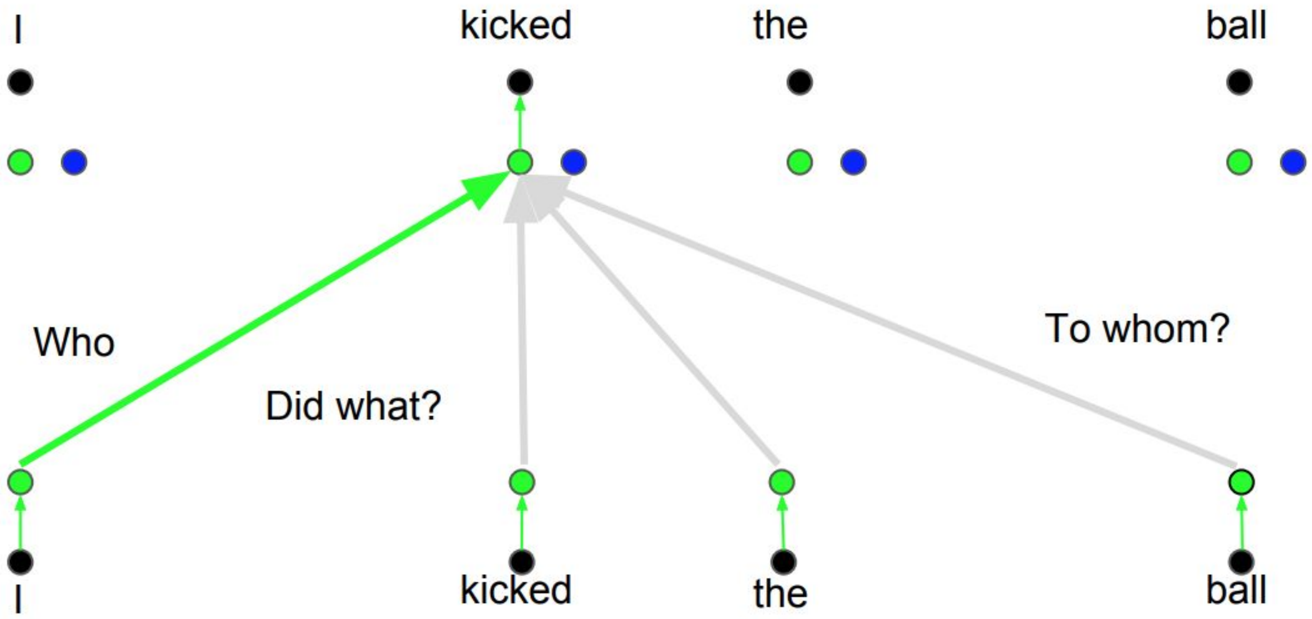


Layer: 5 Attention: Input - Input

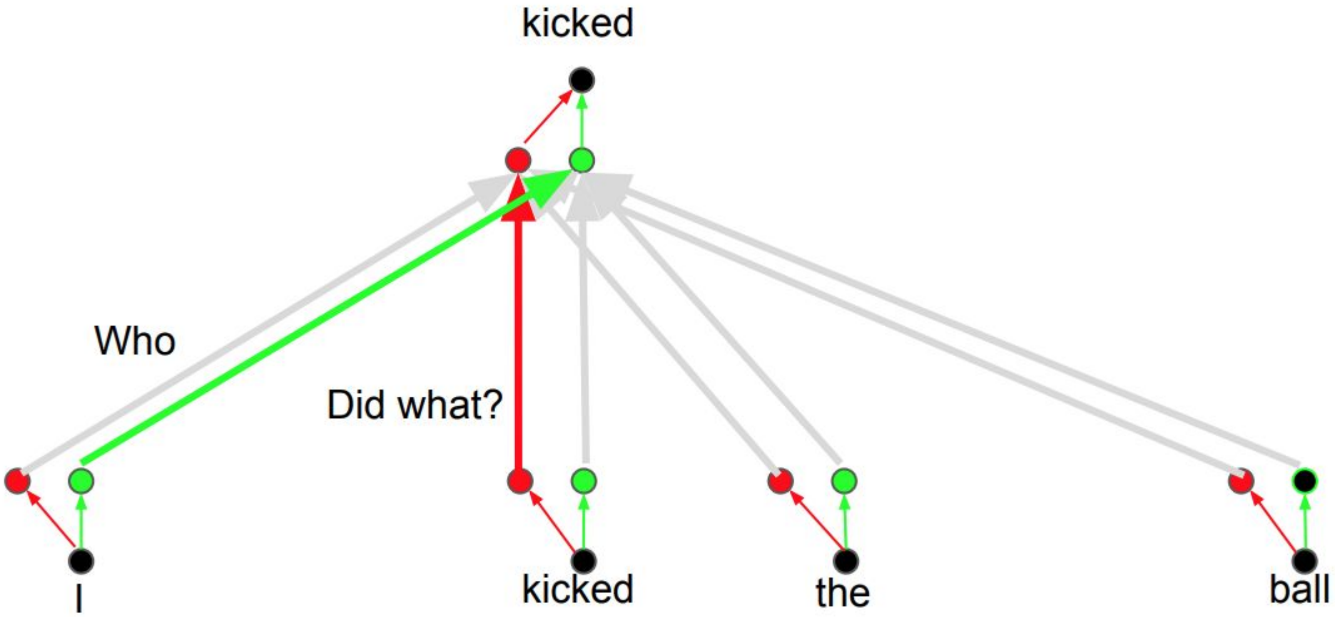
Why Multi-Head Attention?



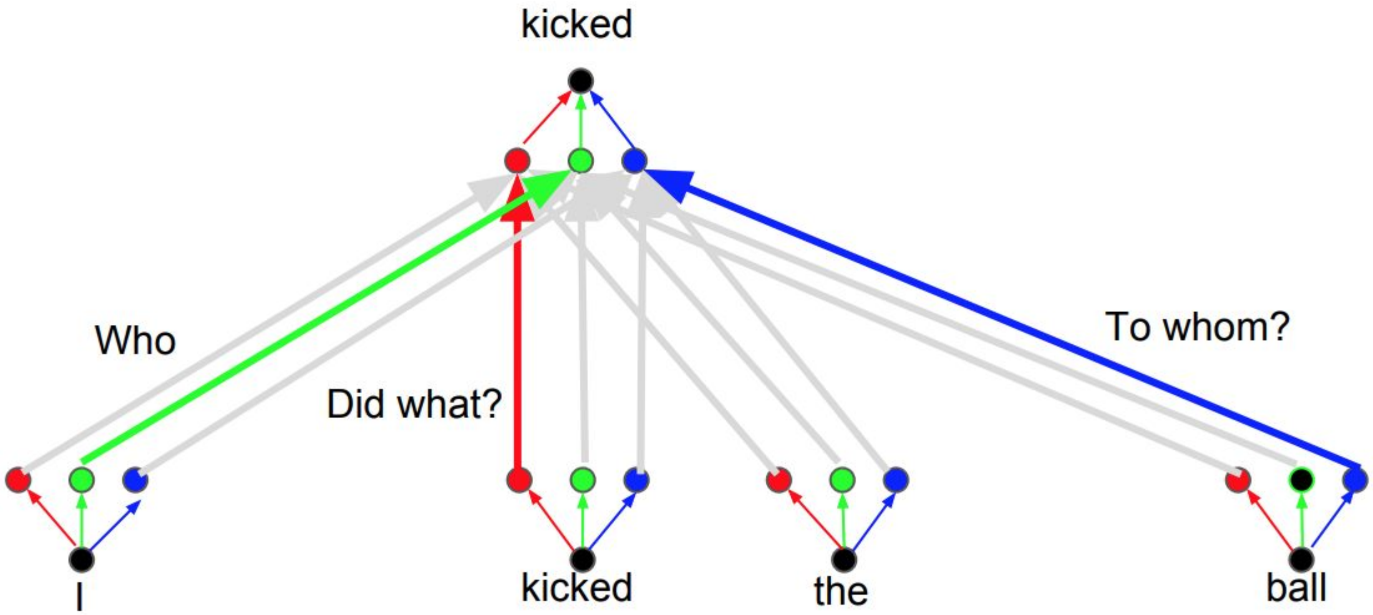
Attention head: Who



Attention head: Did What?

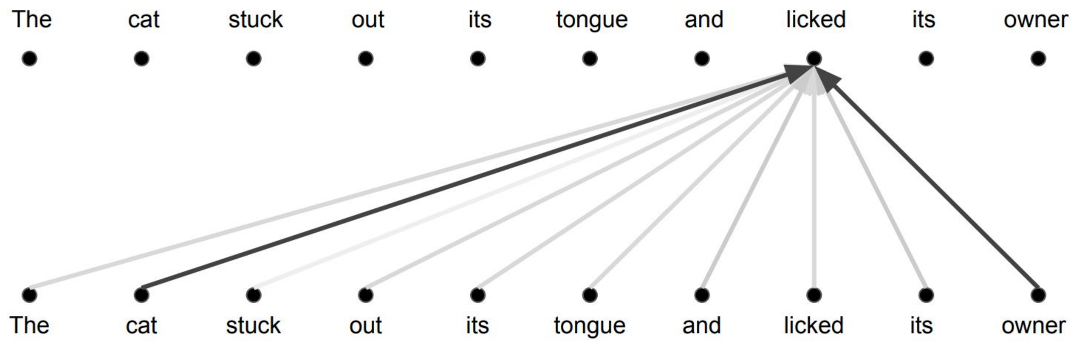


Attention head: To Whom?



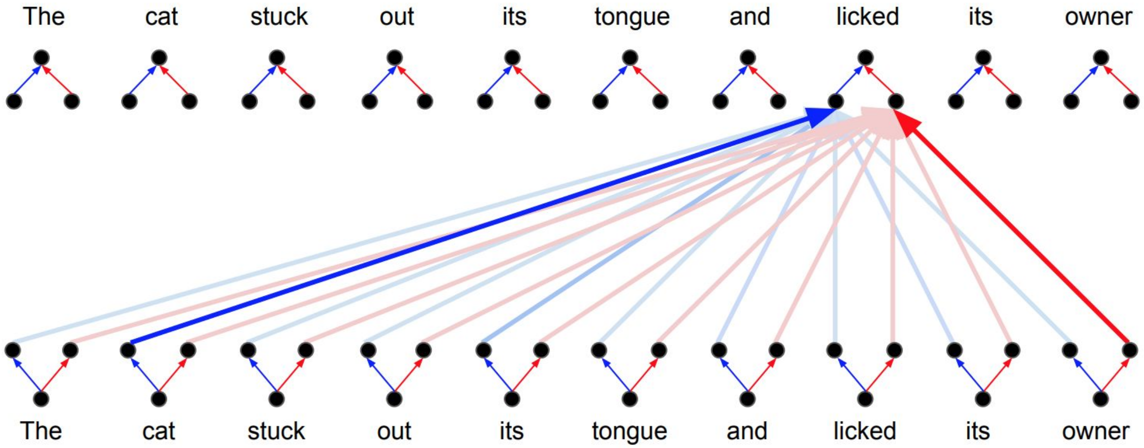
Attention vs. Multi-Head Attention

Attention: a weighted average

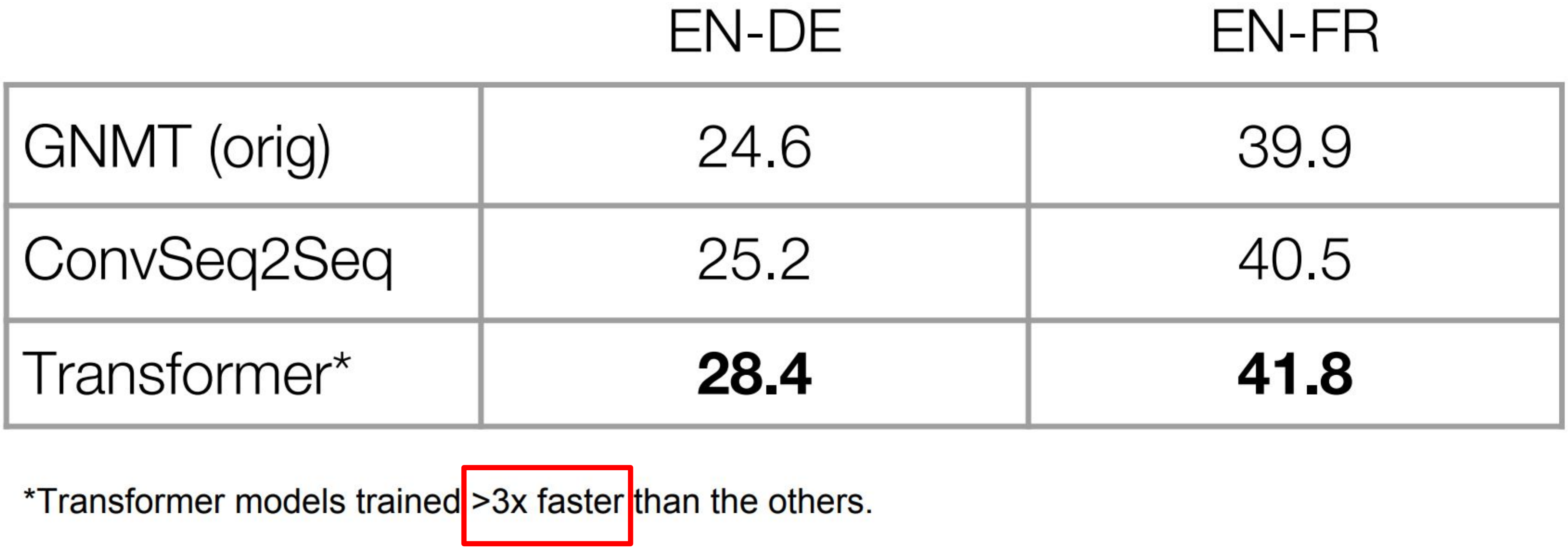


Multi-Head Attention:

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*	**28.4**	**41.8**

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

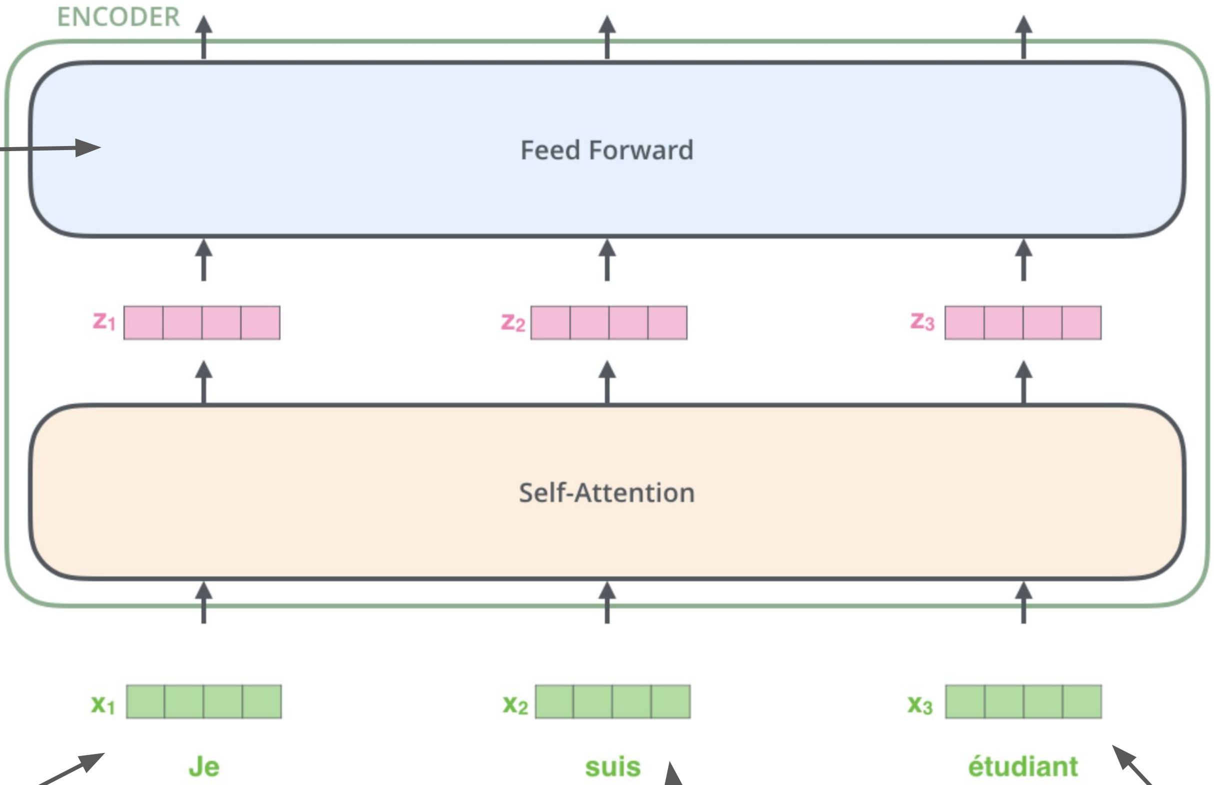
Research Challenges

- 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

Can be parallelized



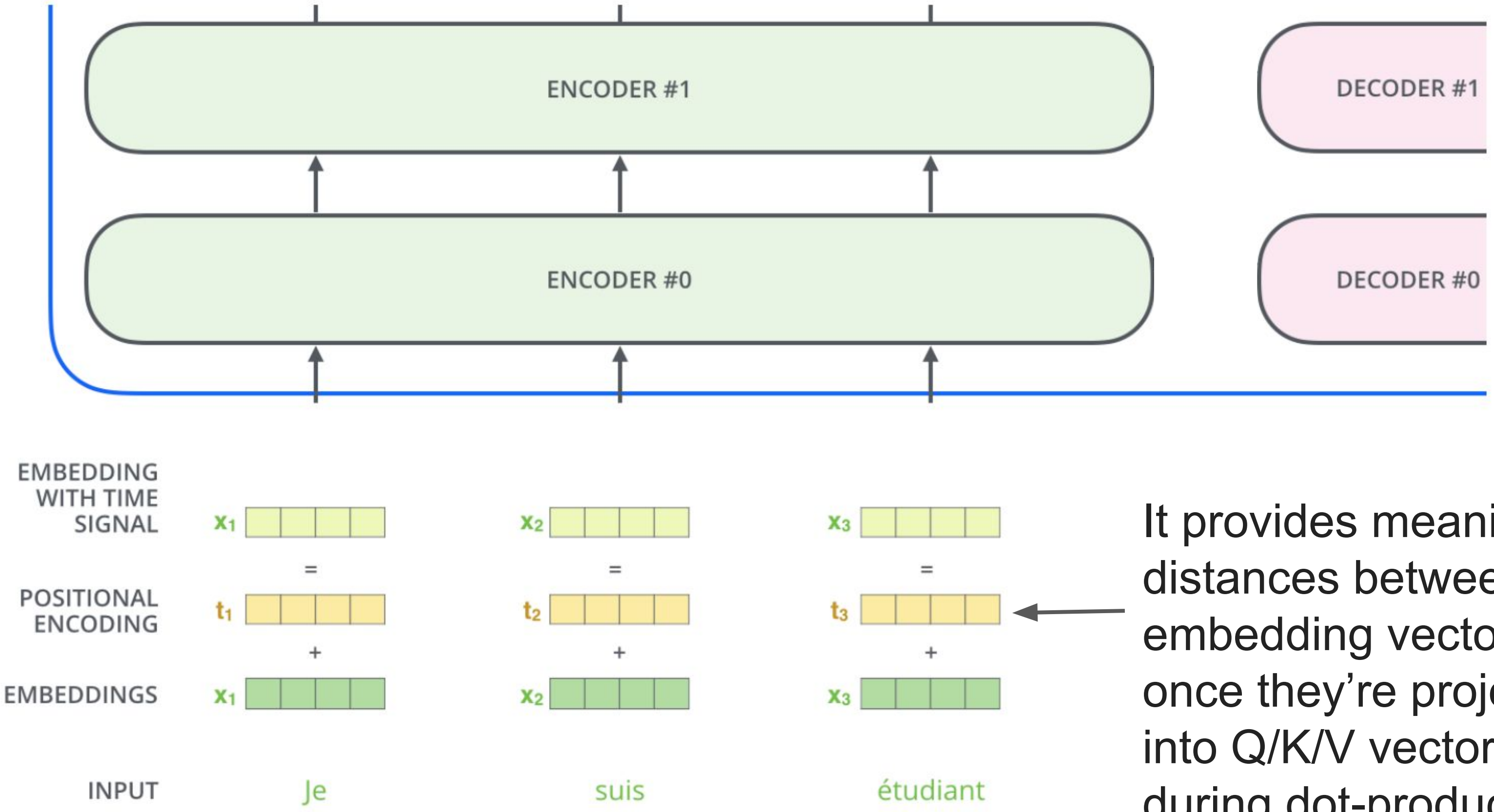
the word in each position flows through its own path in the encoder

Image source: https://jalammar.github.io/illustrated-transformer/

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



It provides meaningful distances between the embedding vectors once they're projected into Q/K/V vectors and during dot-product attention

Image source: https://jalammar.github.io/illustrated-transformer/

Positional Encoding: why sin and cos?

$$\vec{p_t}^{(i)} = f(t)^{(i)} \quad = \begin{cases} \sin(\omega_k t), & \text{if } i = 2k \\ \cos(\omega_k t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

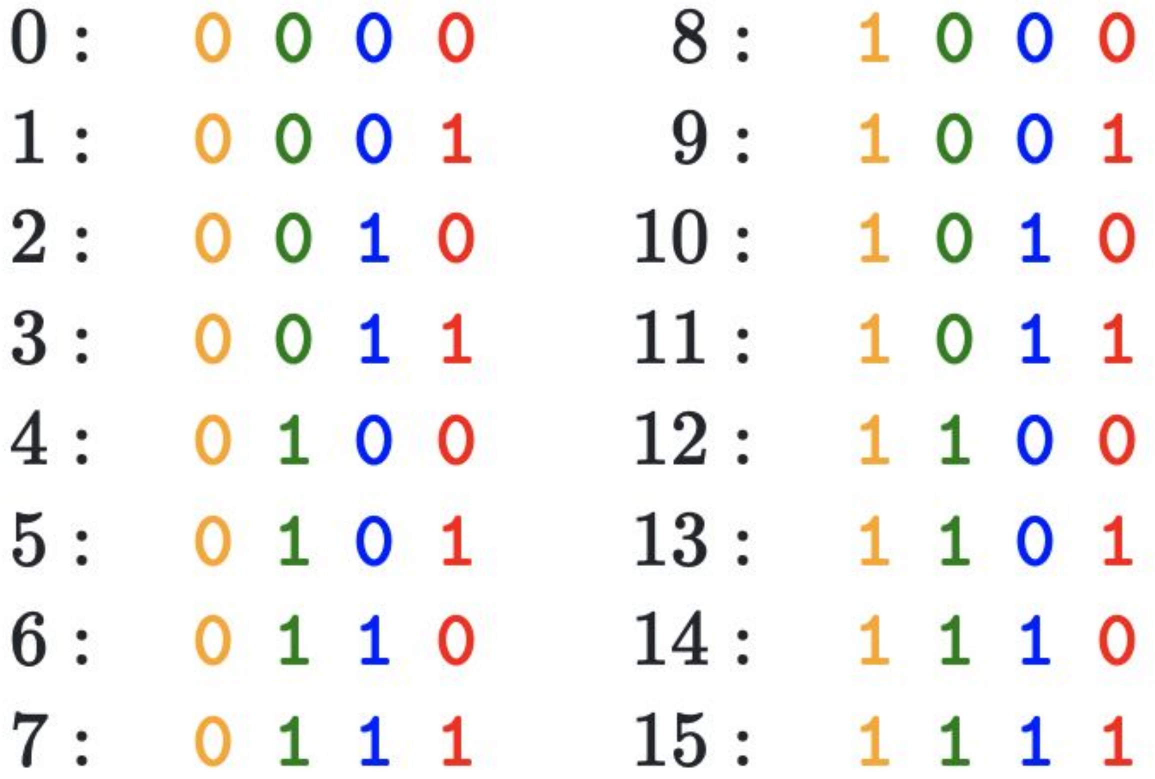
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

t stays for position in the original sequence

k is the index of the element in the positional vector

Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding



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

Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding

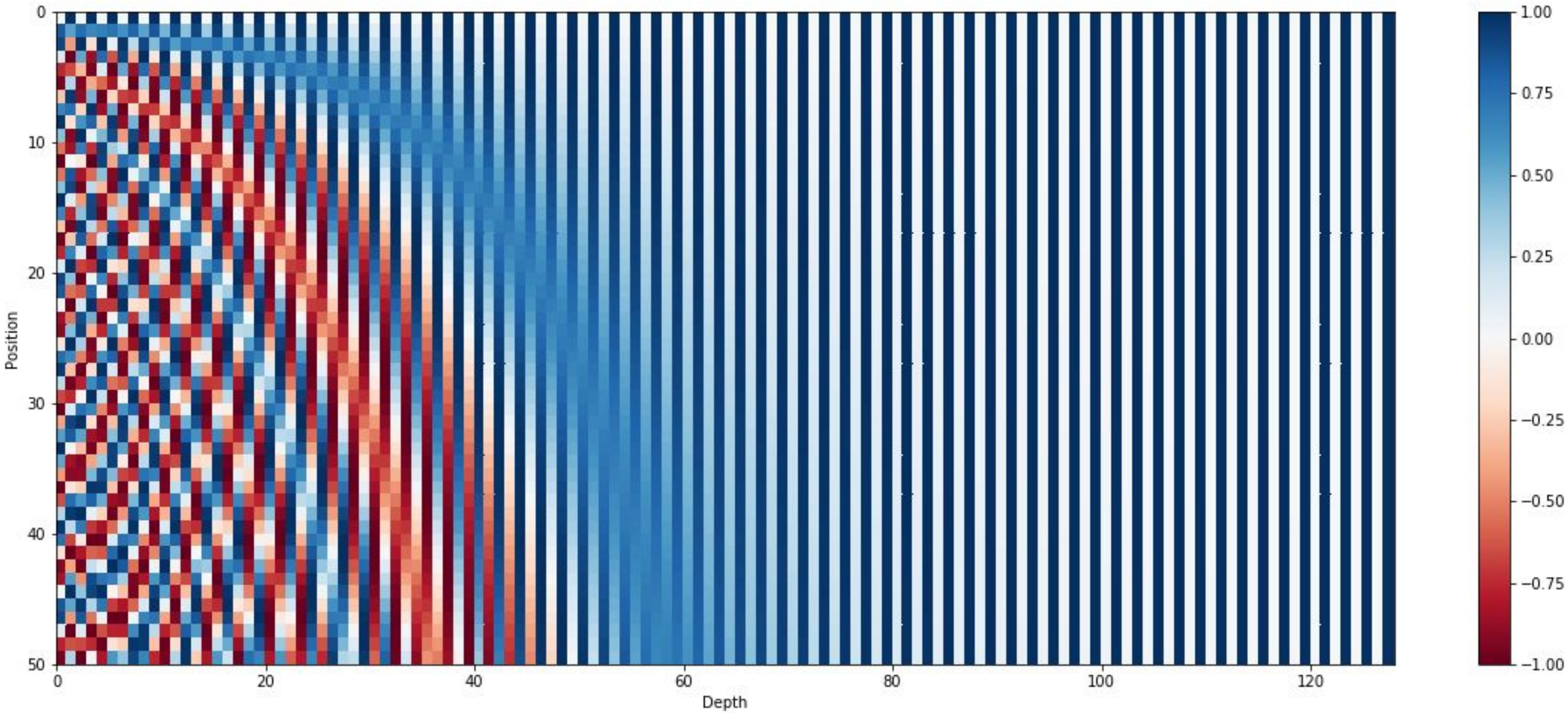
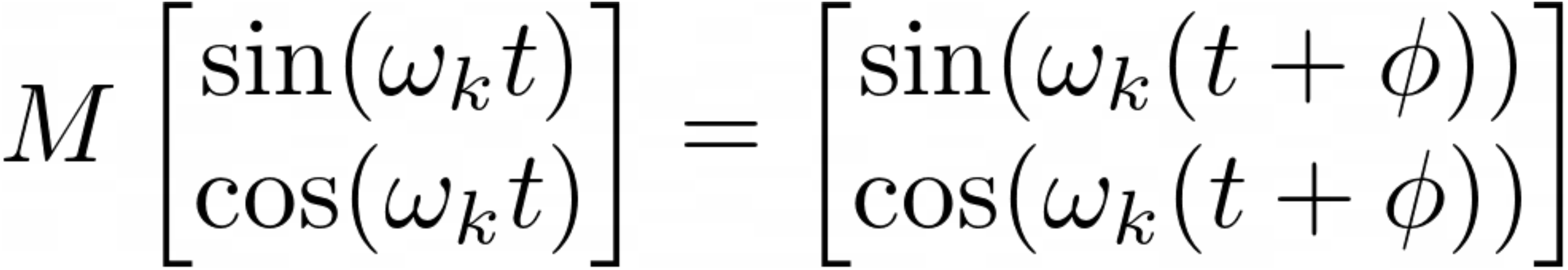


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

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, PEpos+k can be represented as a linear function of PEpos.



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

Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding: why sin and cos?

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

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

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

Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Positional Encoding

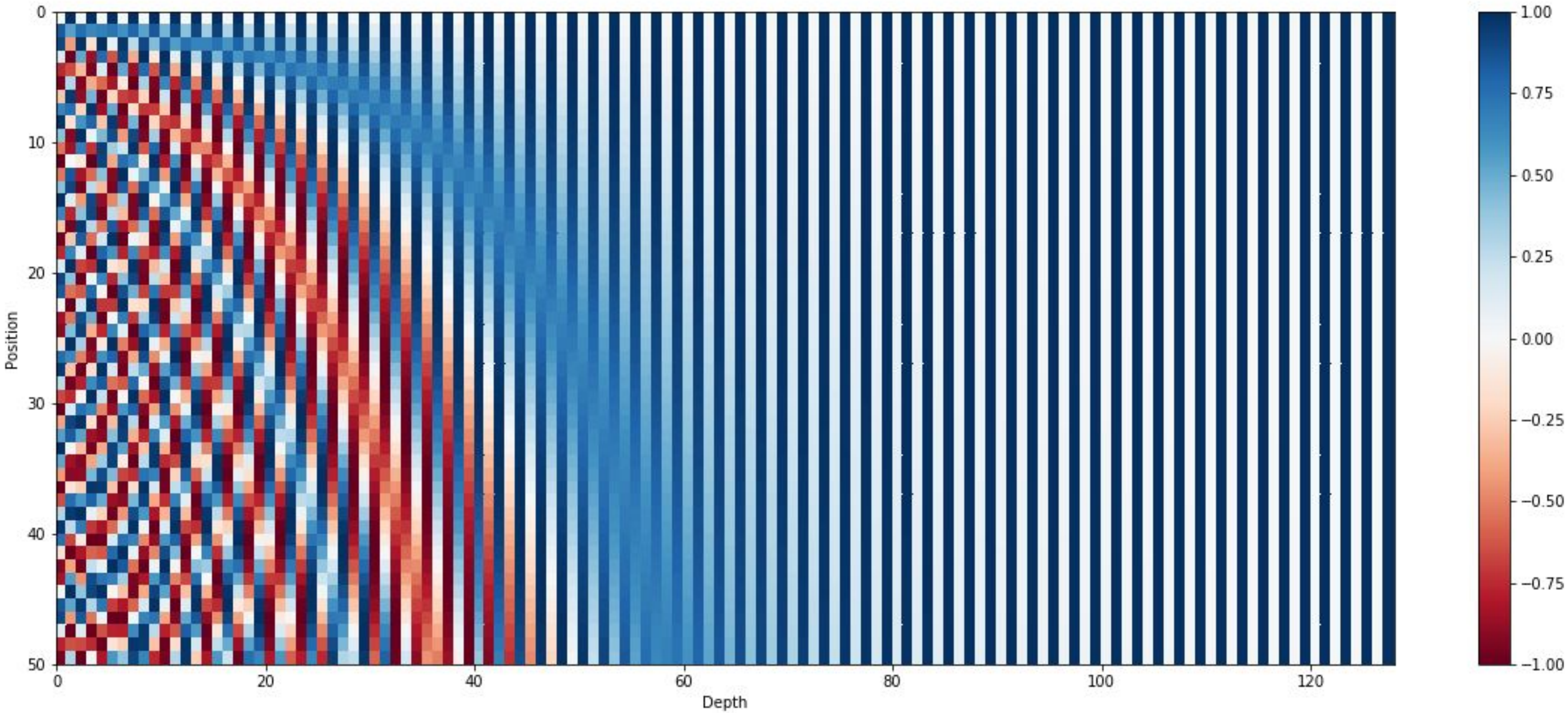


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

Layer Normalization

The Transformer: recap

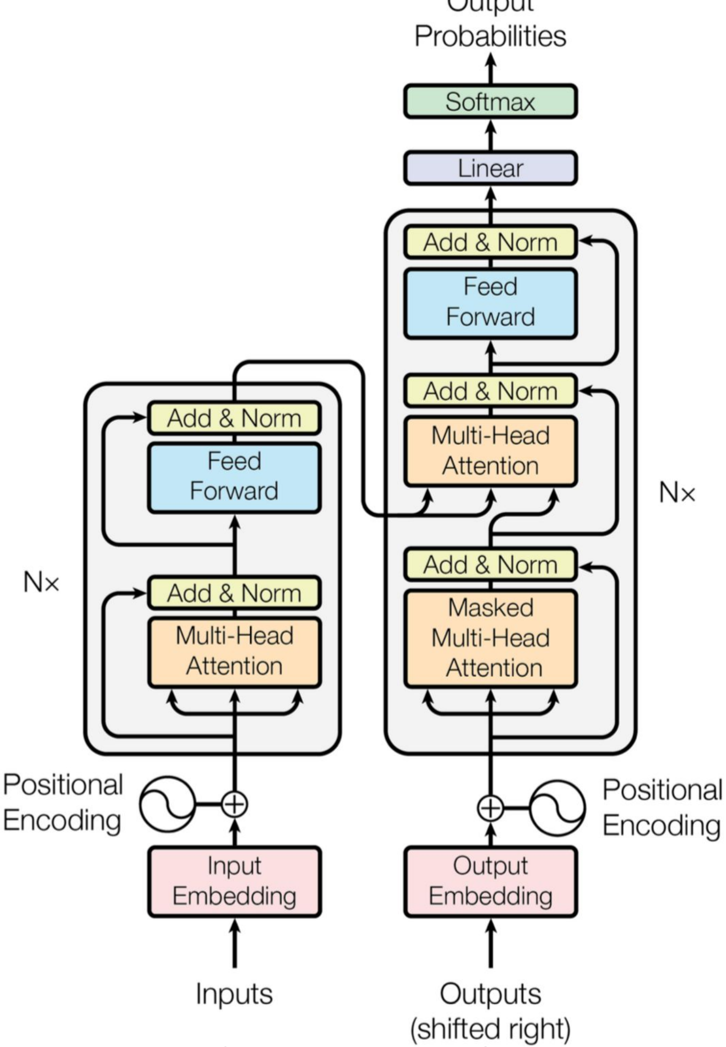


Image source: Attention Is All You Need, Neural Information Processing Systems 2017

Layer Normalization

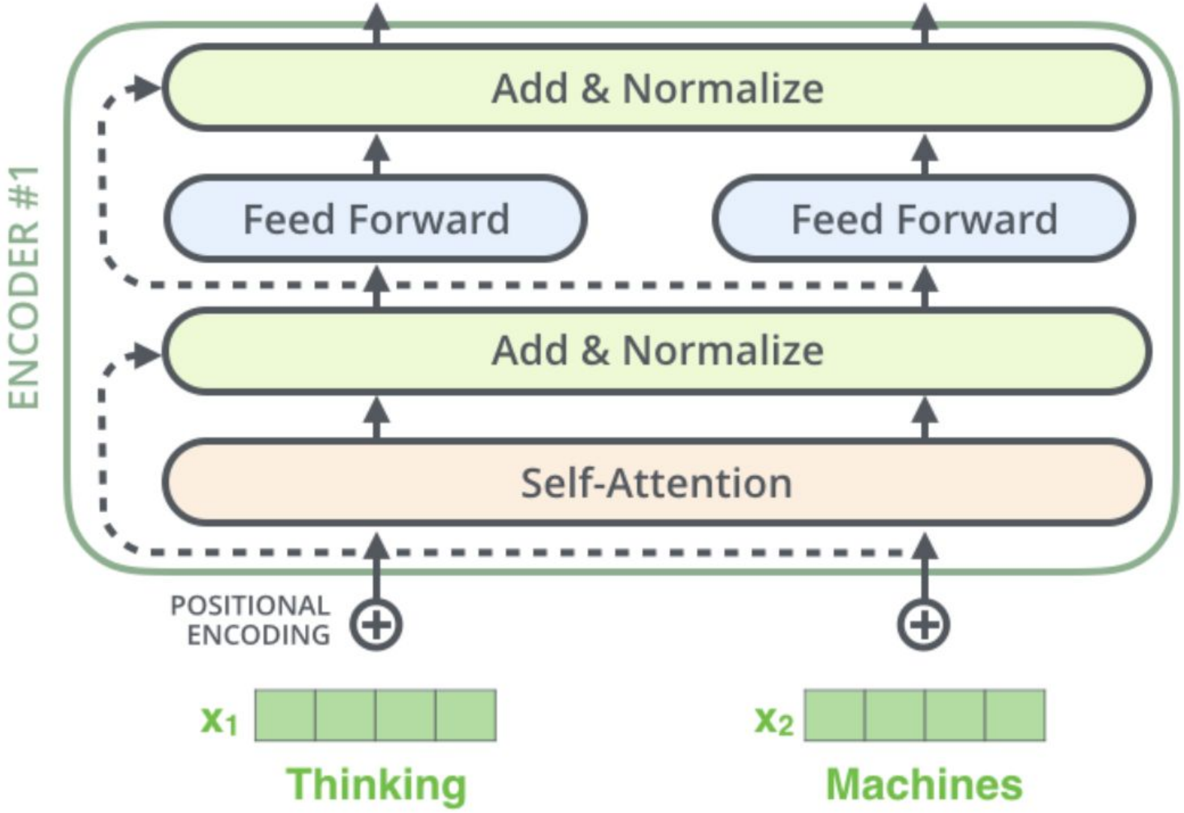
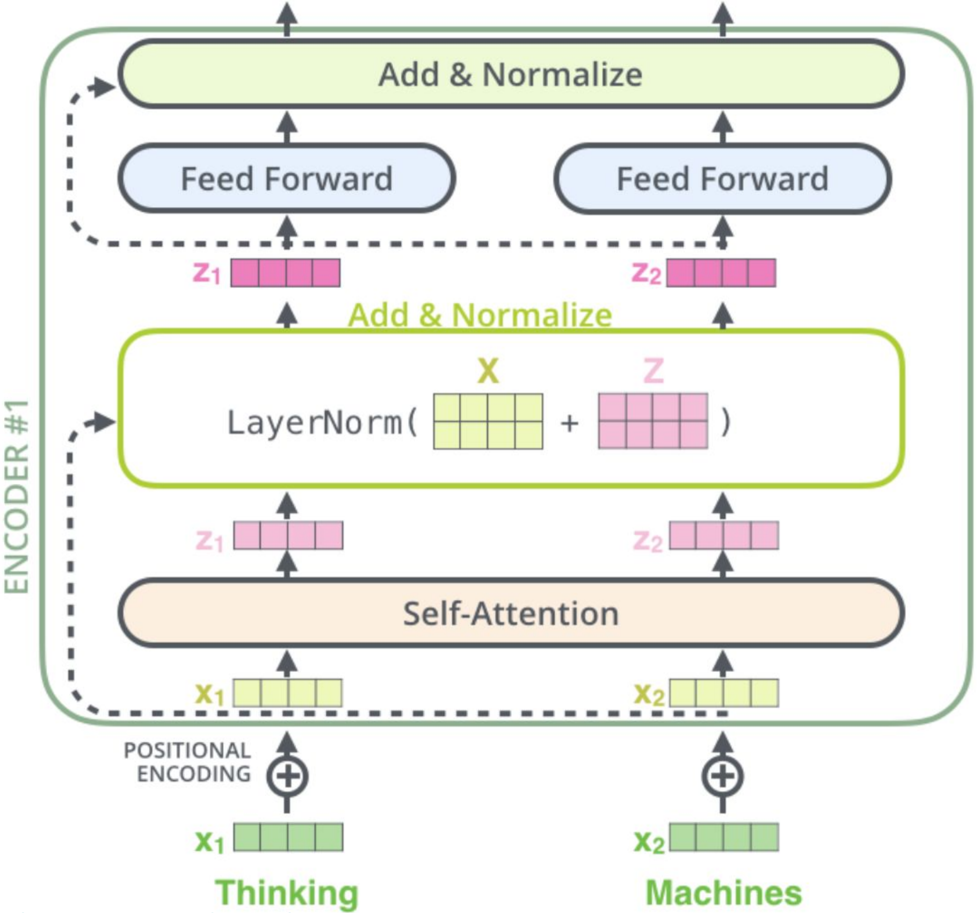


Image source: https://jalammar.github.io/illustrated-transformer/

Layer Normalization

Like BatchNorm

but normalize along all features representing latent vector



More info: [Layer Normalization](https://)

Image source: https://jalammar.github.io/illustrated-transformer/

Layer Normalization

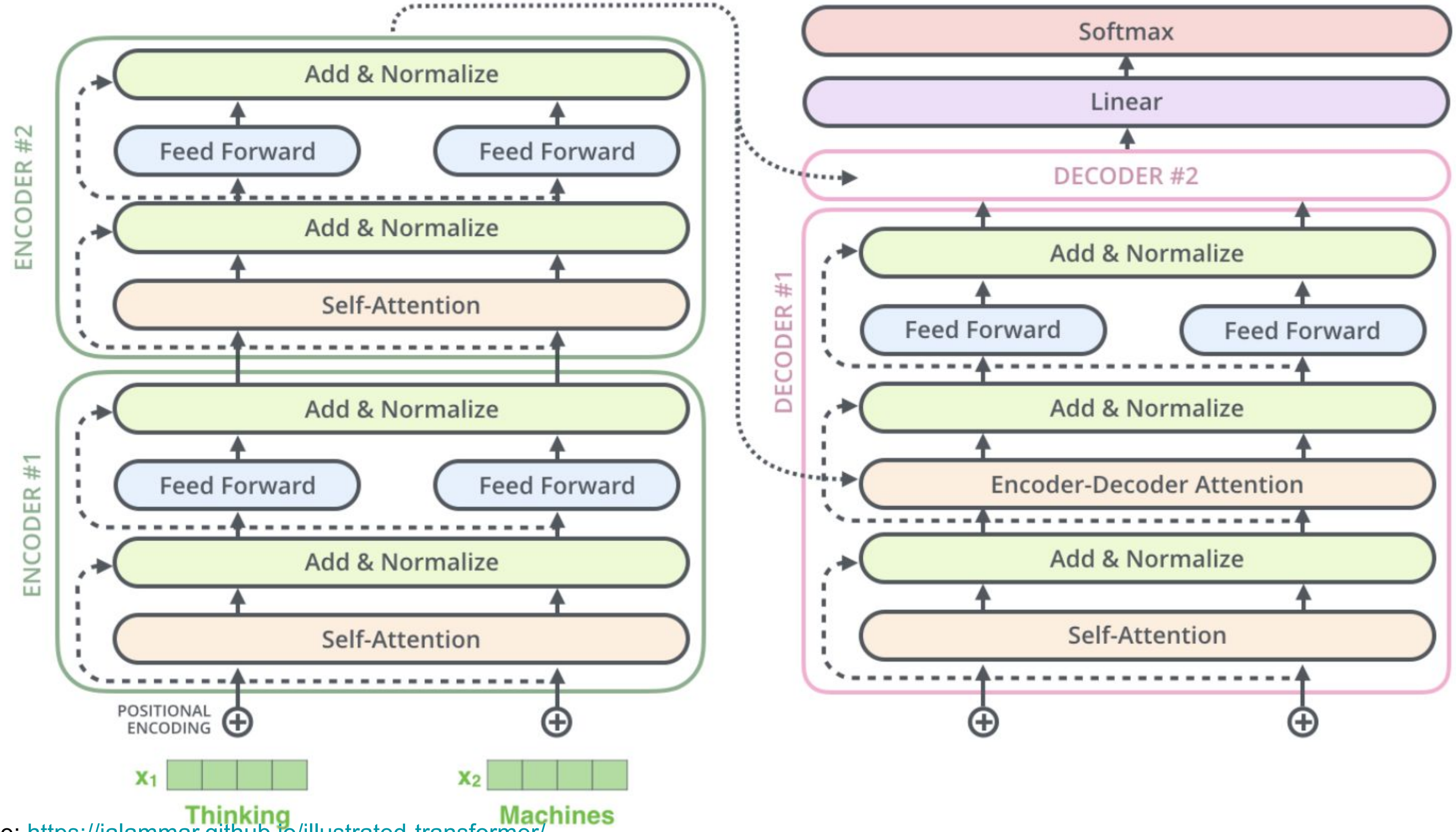


Image source: https://jalammar.github.io/illustrated-transformer/

The Decoder

The Decoder Side

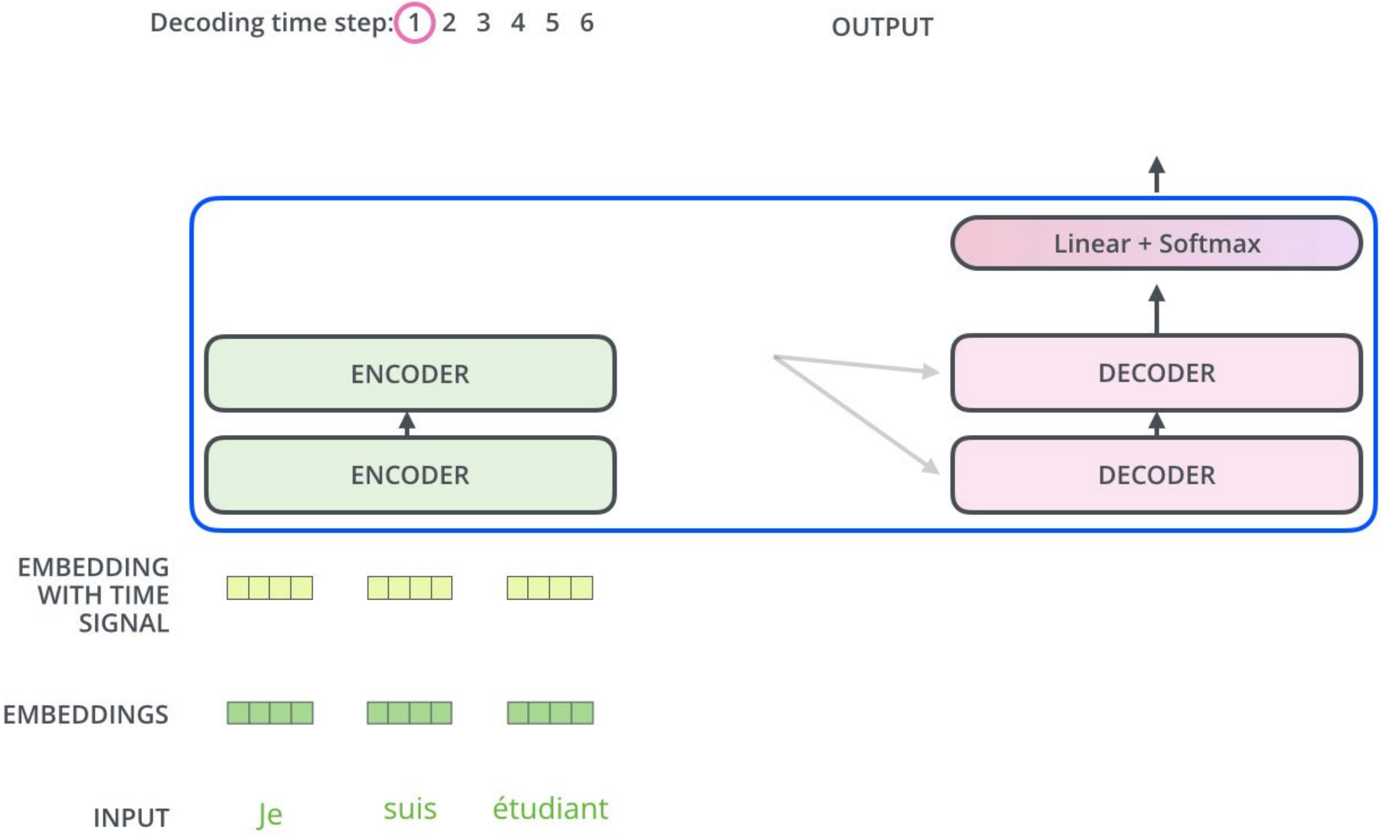
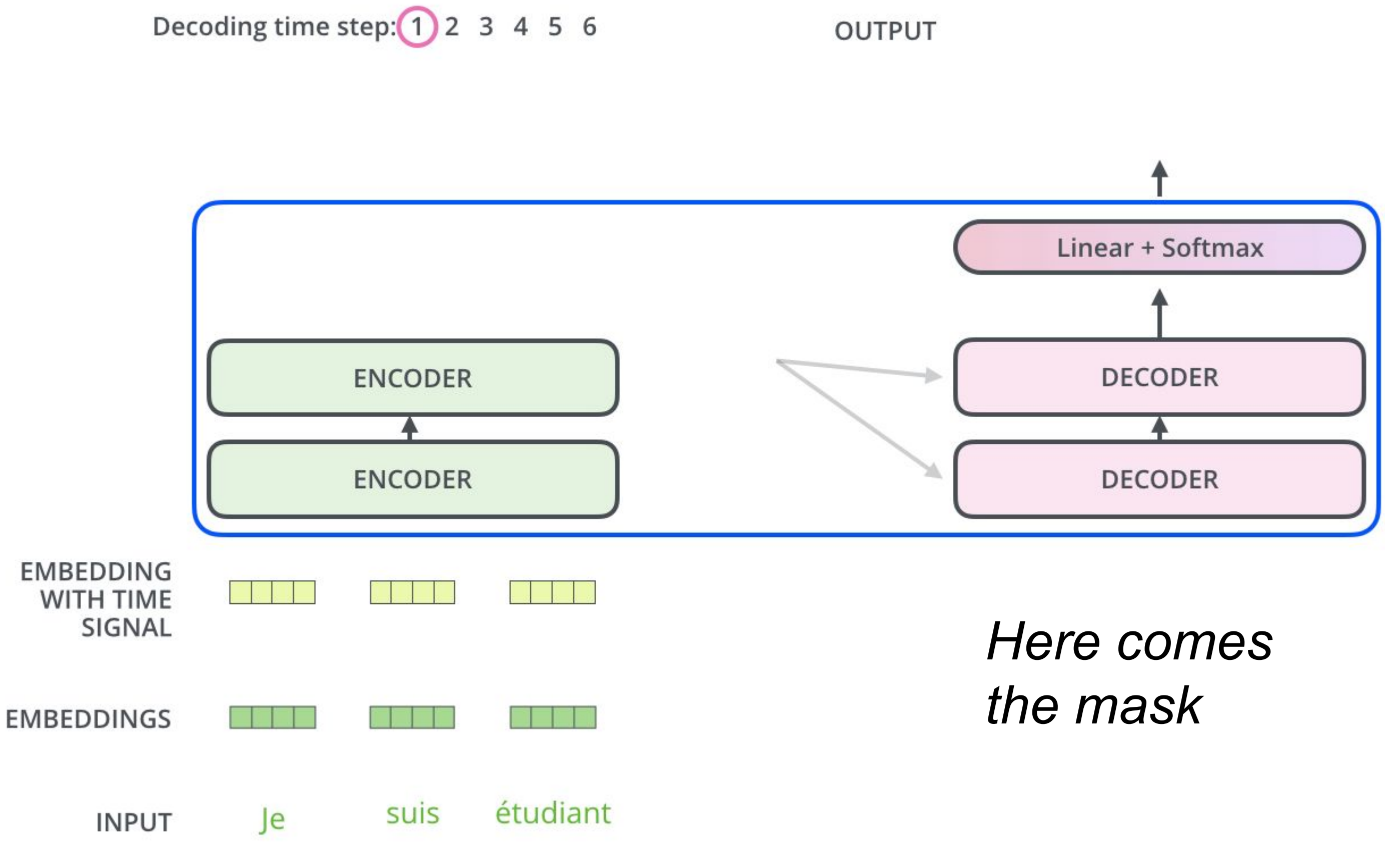


Image source: https://jalammar.github.io/illustrated-transformer/

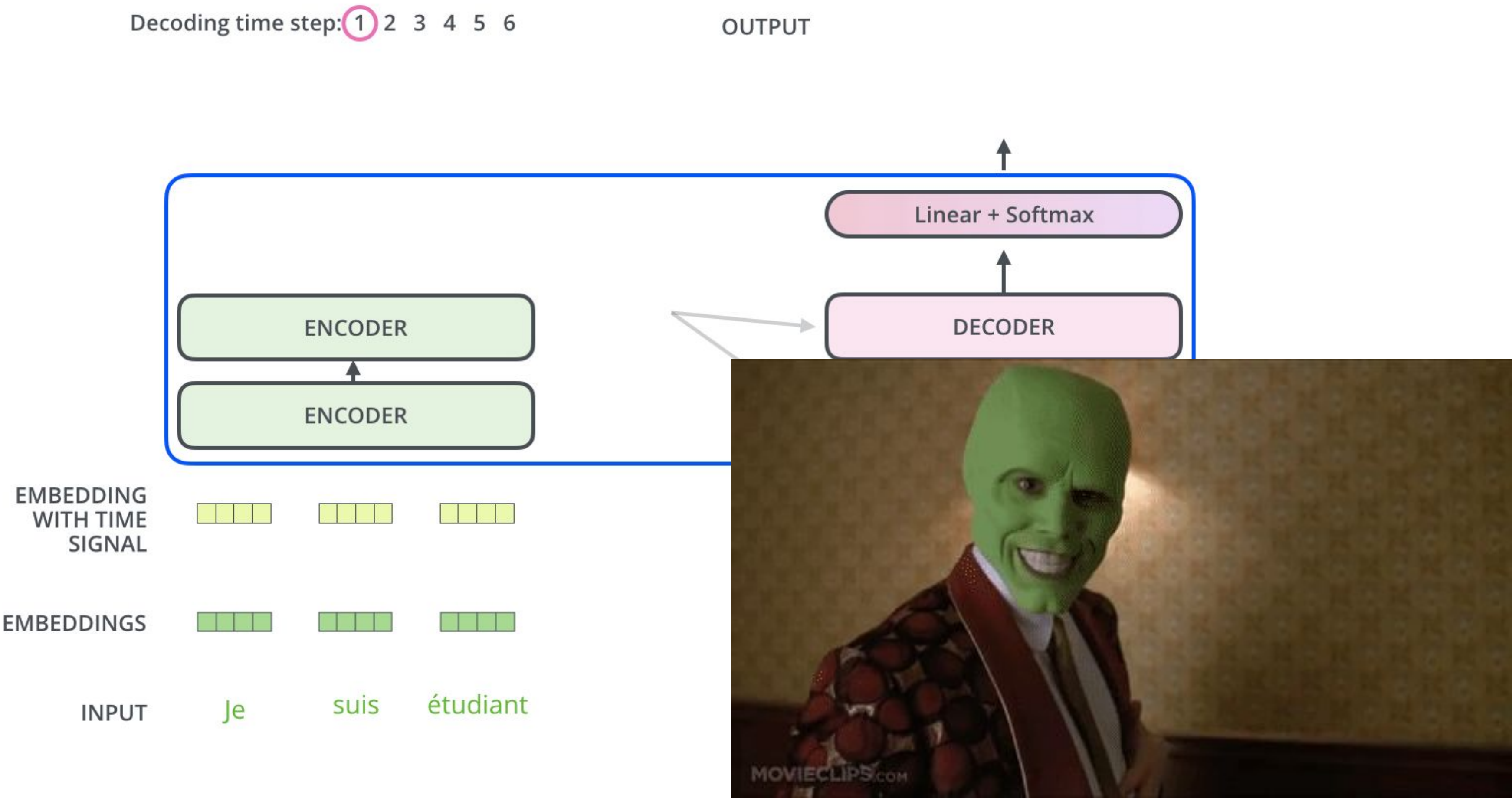
The Decoder Side



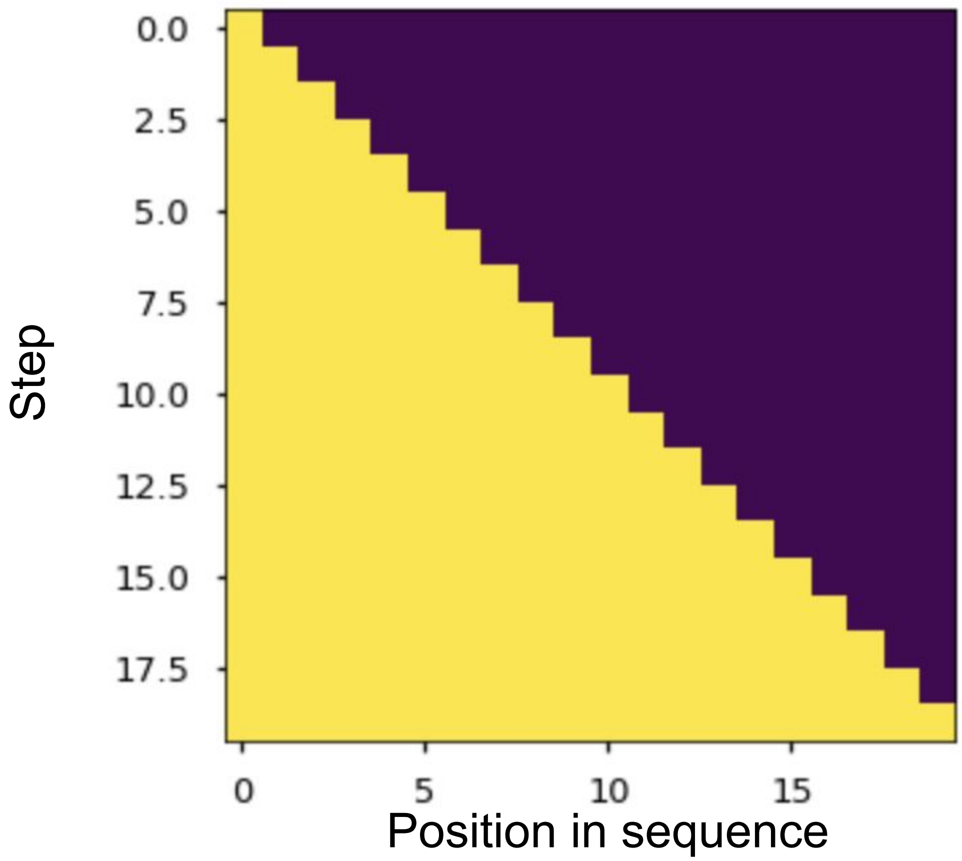
Here comes the mask

Image source: https://jalammar.github.io/illustrated-transformer/

The Decoder Side



The masked decoder input



The Decoder Side

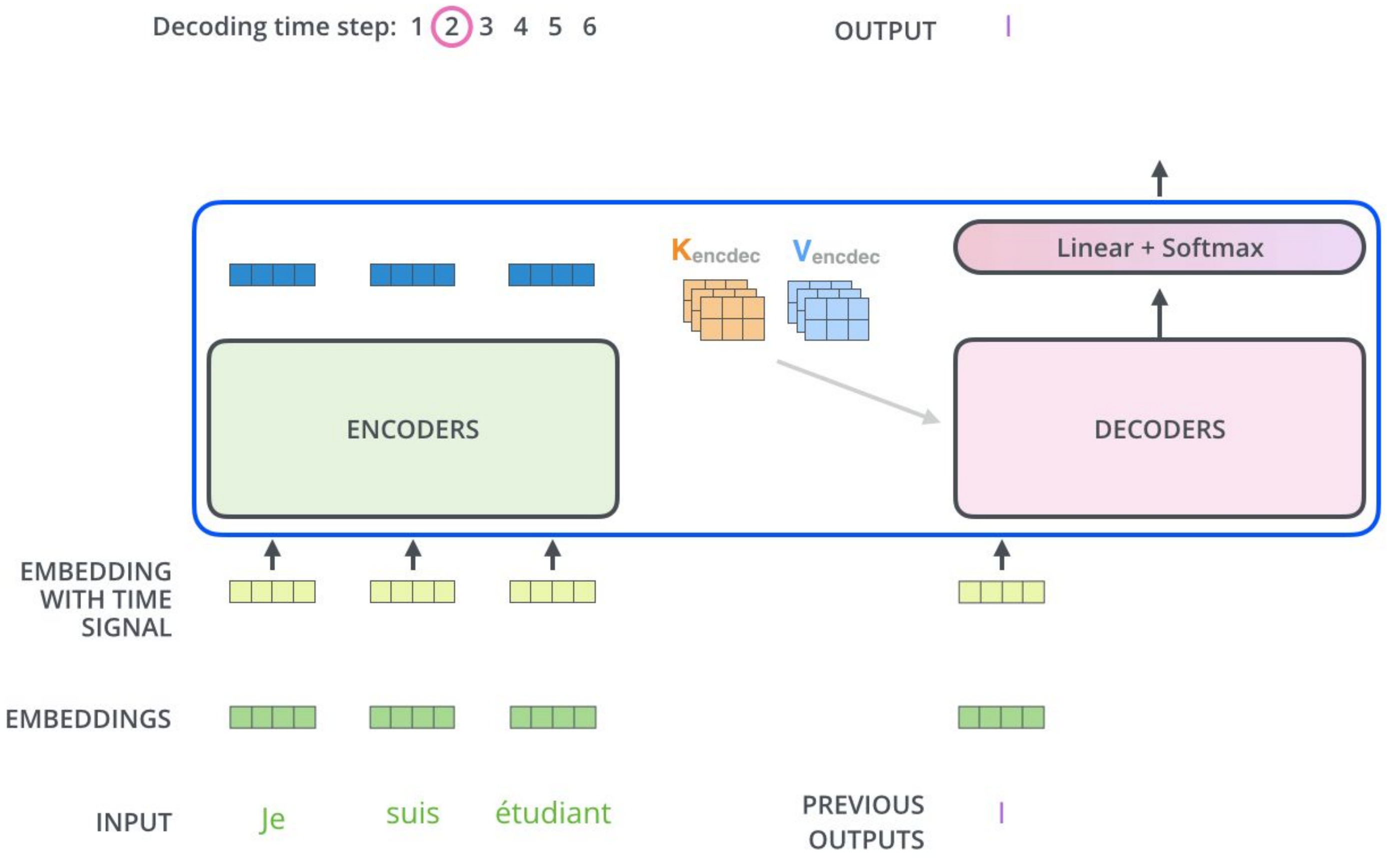
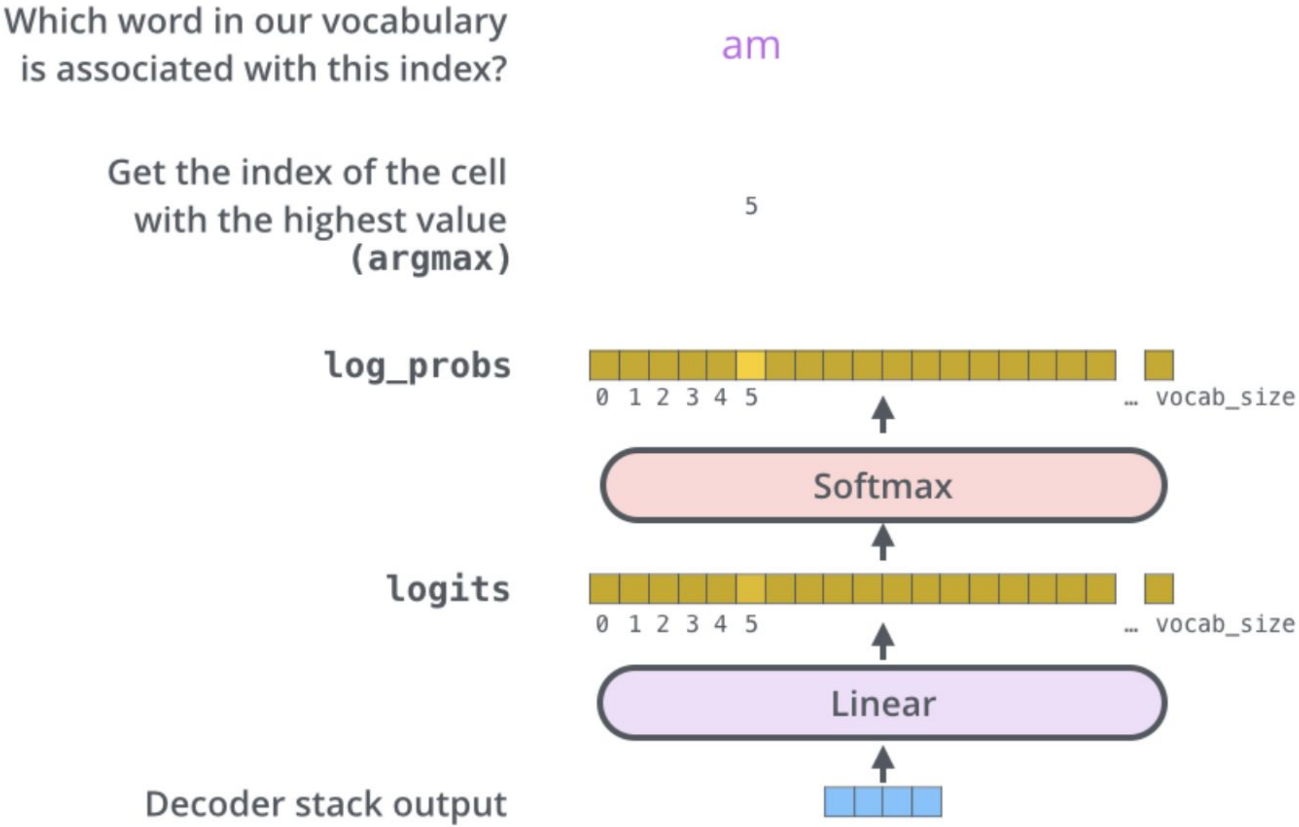


Image source: https://jalammar.github.io/illustrated-transformer/

Final Linear and Softmax Layer



Which word in our vocabulary is associated with this index?

am

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

5

log_probs

0 1 2 3 4 5 … vocab_size

Softmax

logits

0 1 2 3 4 5 … vocab_size

Linear

Decoder stack output

Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

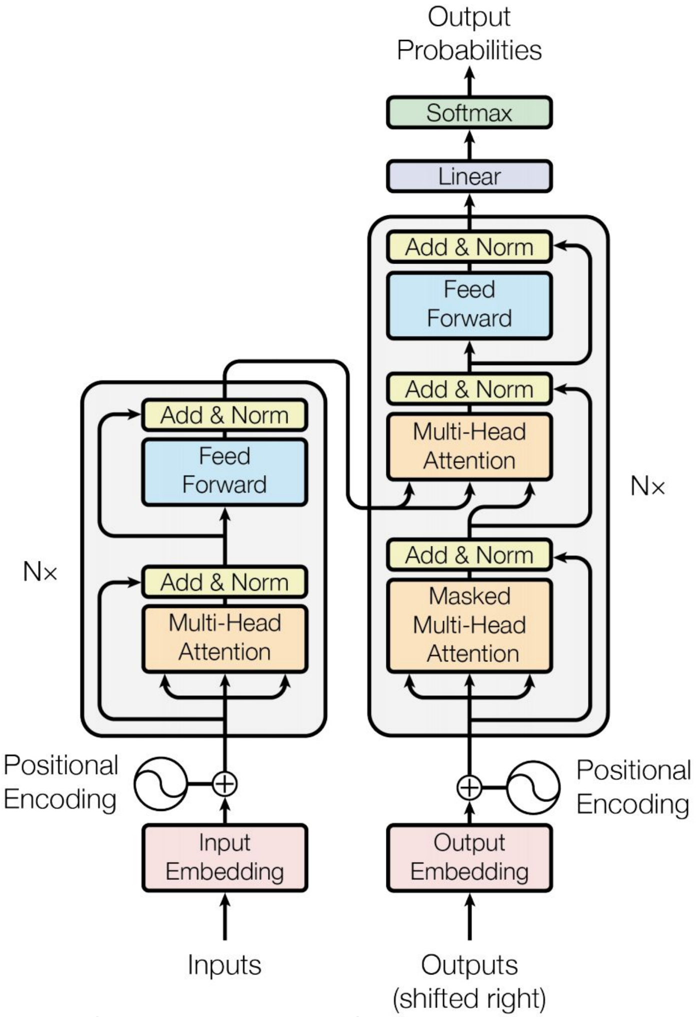


Image source: Attention Is All You Need, Neural Information Processing Systems 2017

Outro and Q&A

- 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