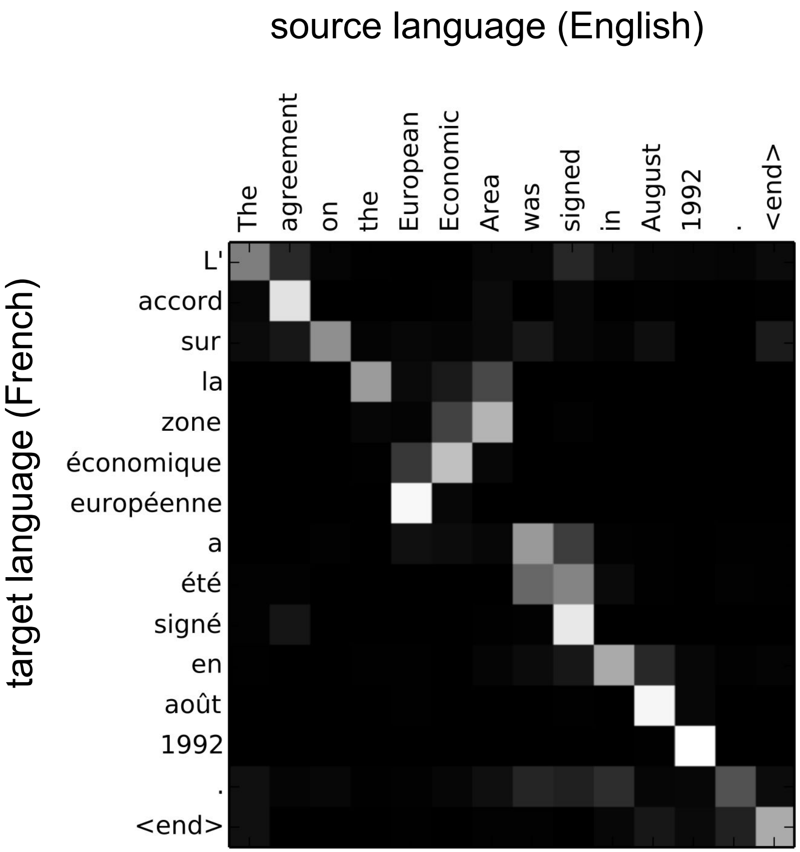
girafe ai

Lecture 03: Attention mechanism

Anastasia Yanina

Words alignment



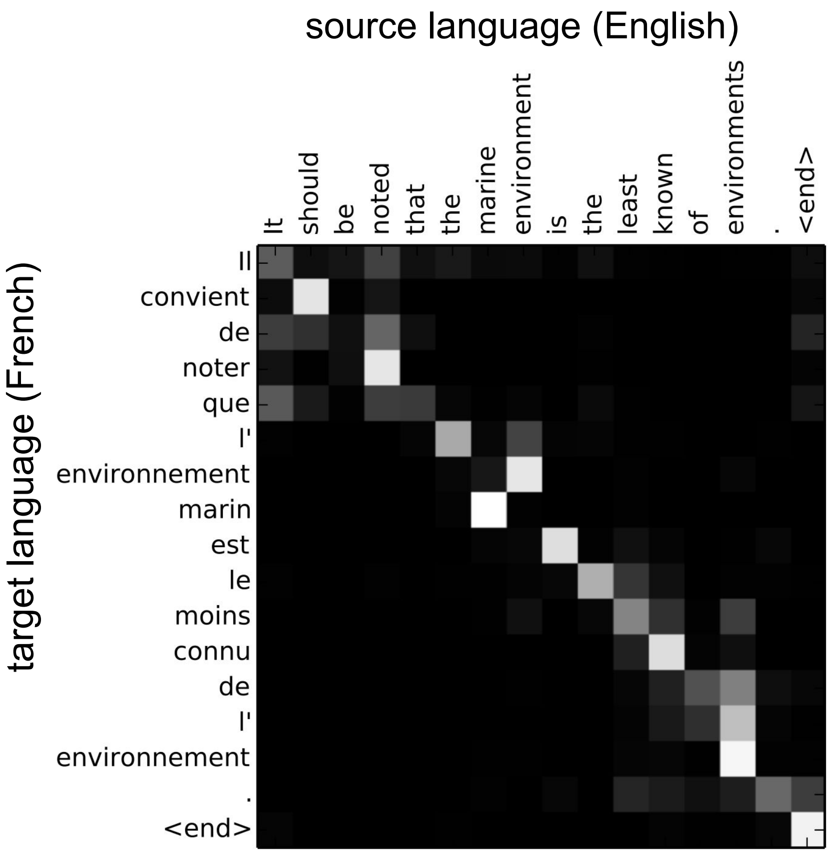


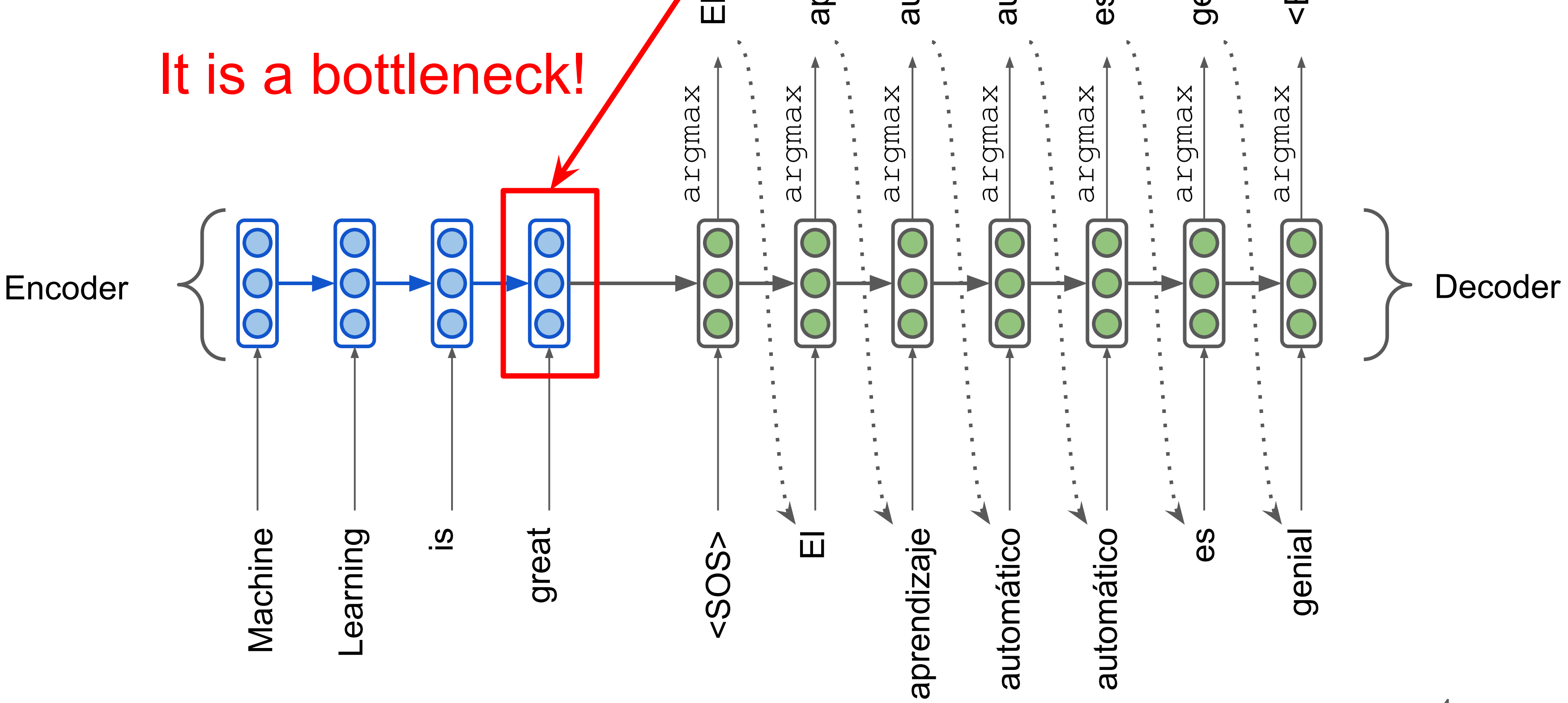
Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

Attention

Seq2seq NMT

This state encodes the whole sentence

It is a bottleneck!



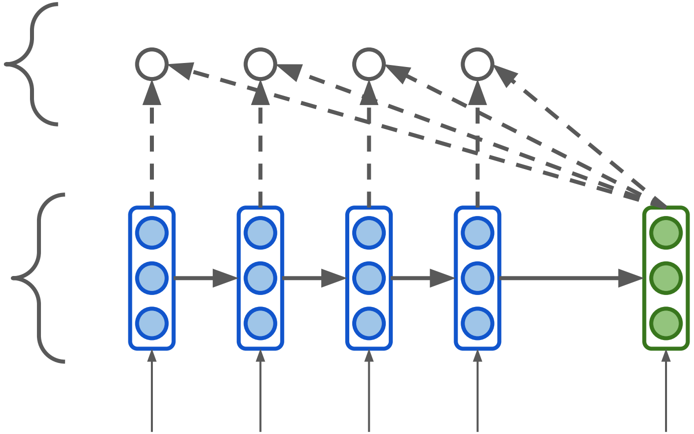
Main idea:

on each step of the **decoder**, use **direct connection to the encoder** to focus on a particular part of the source sequence

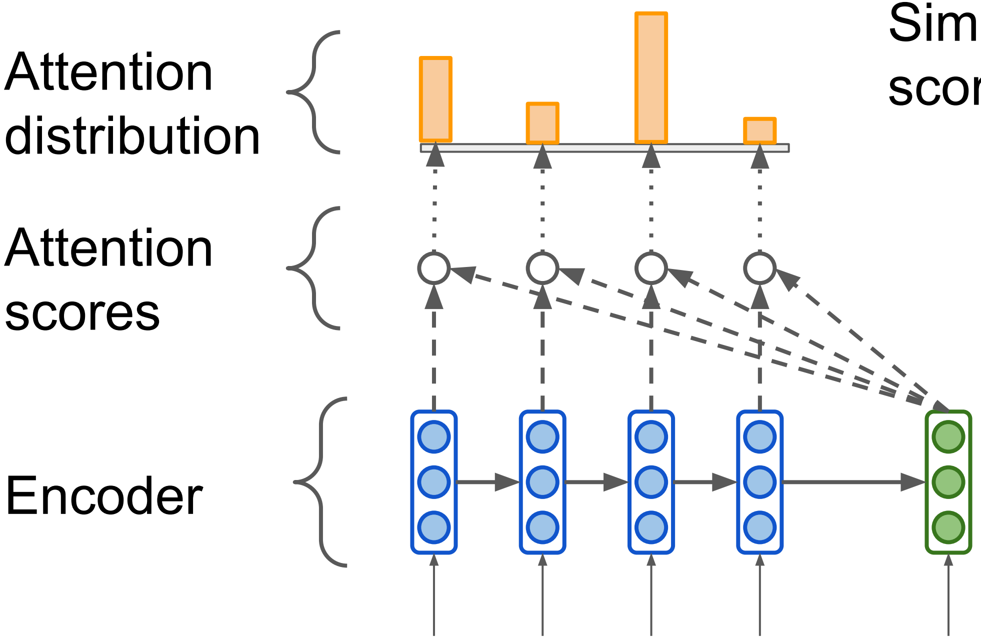
Seq2seq with attention

Attention scores

Encoder



Seq2seq with attention



Simply apply softmax to scores

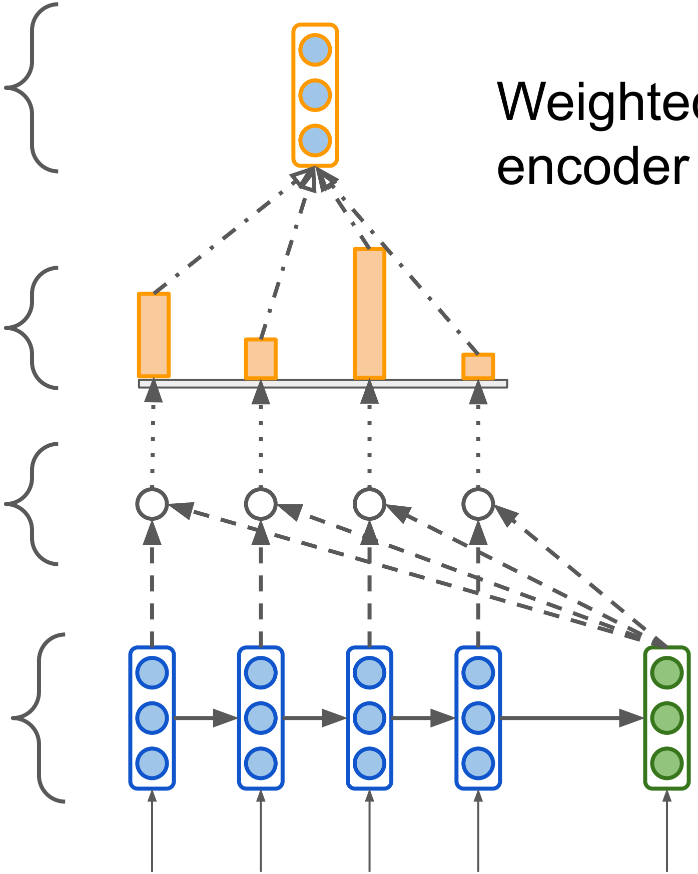
Seq2seq with attention

Attention output

Attention distribution

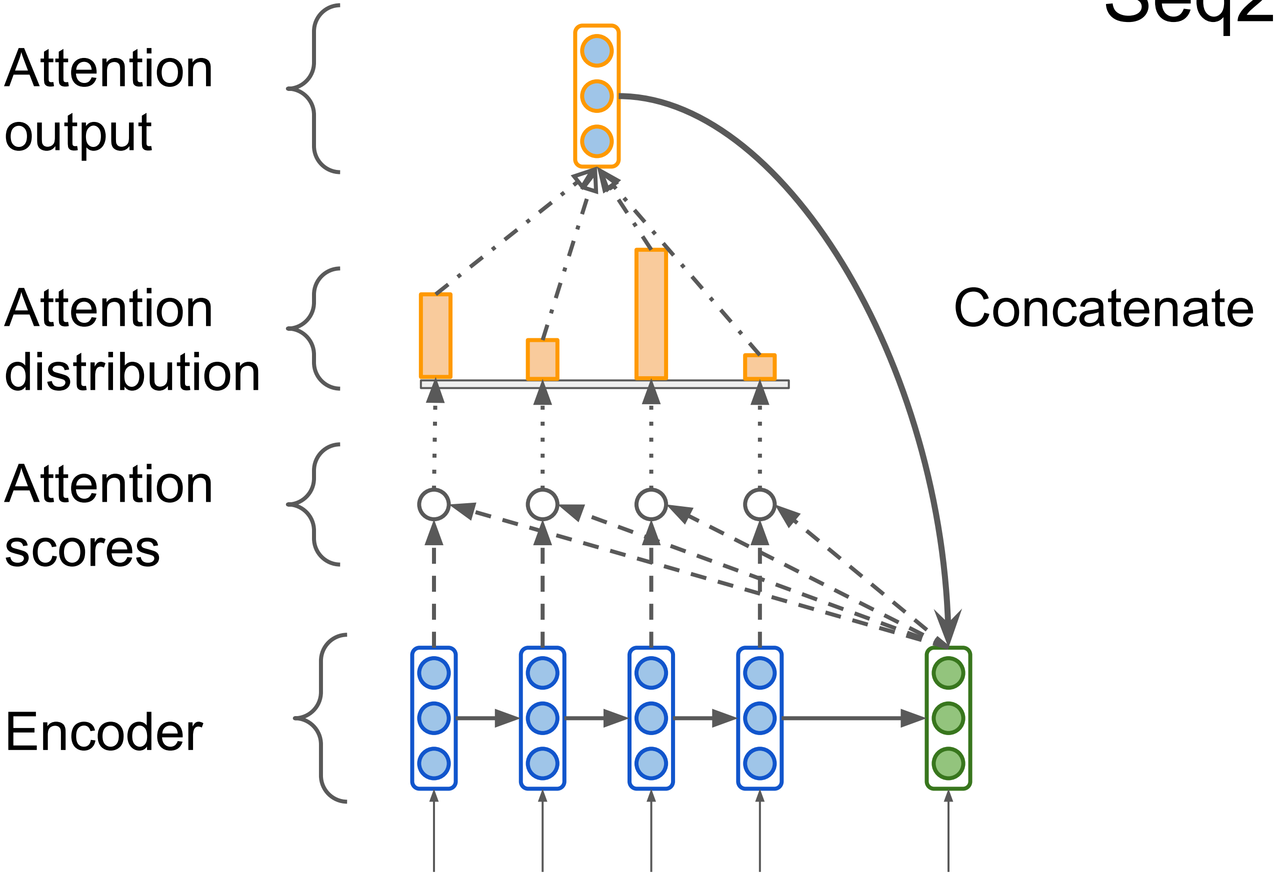
Attention scores

Encoder

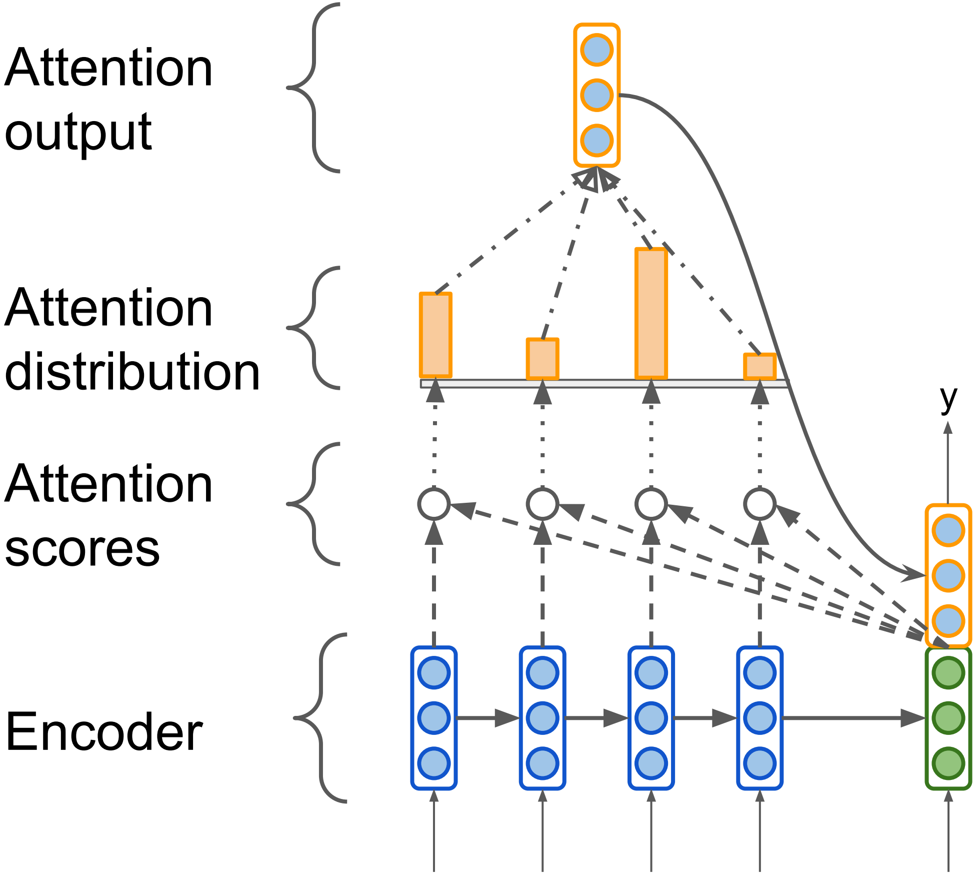


Weighted sum of all encoder states

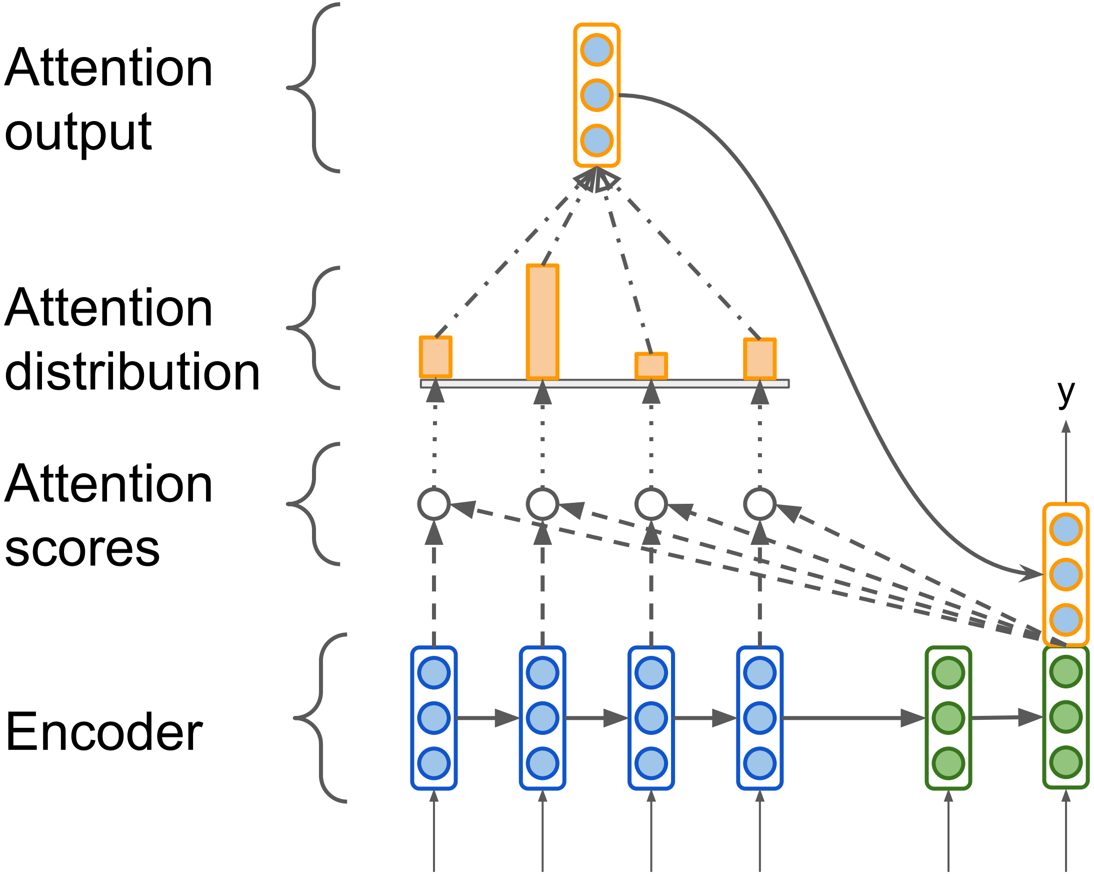
Seq2seq with attention



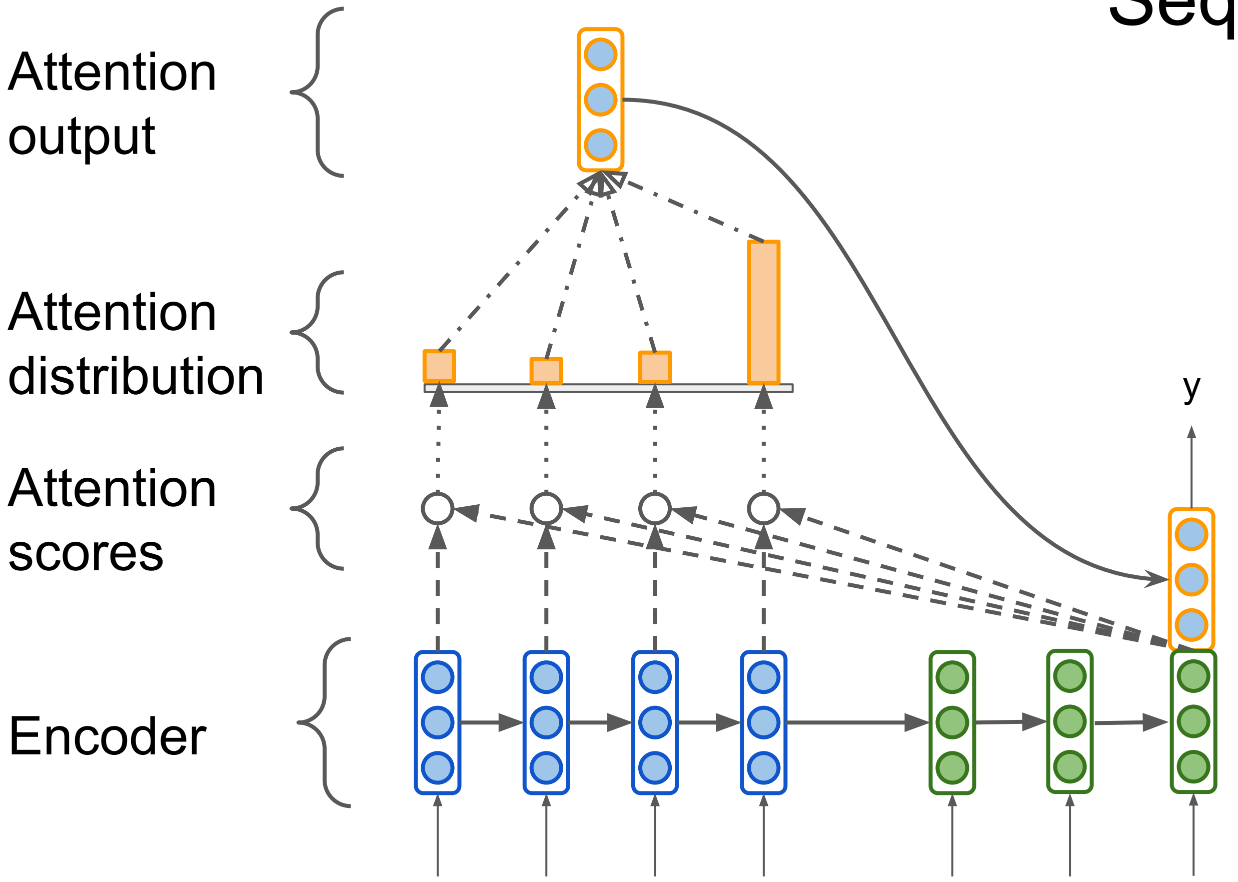
Seq2seq with attention



Seq2seq with attention



Seq2seq with attention



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 provides interpretability

- We may see what the decoder was focusing on
- We get word alignment for free!

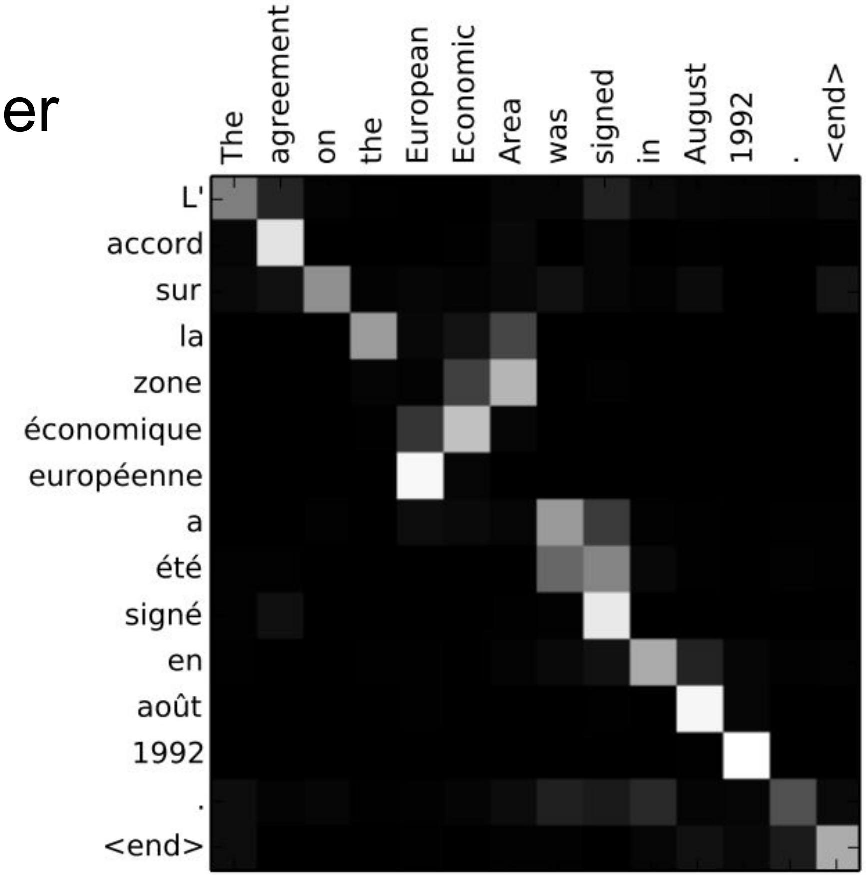
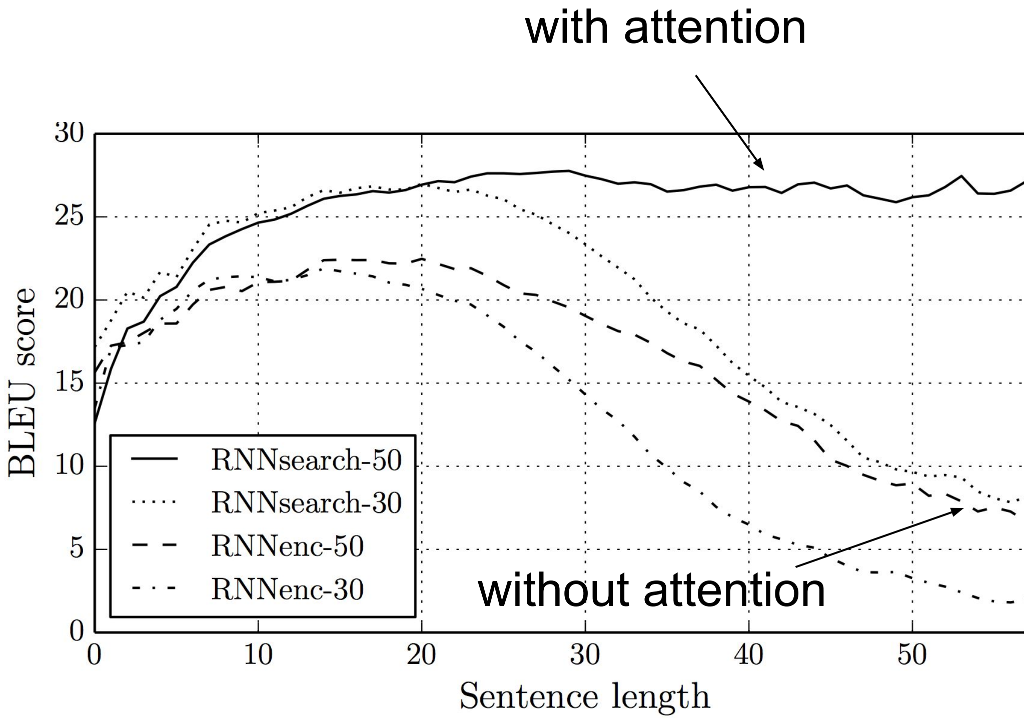


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

Attention advantages

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



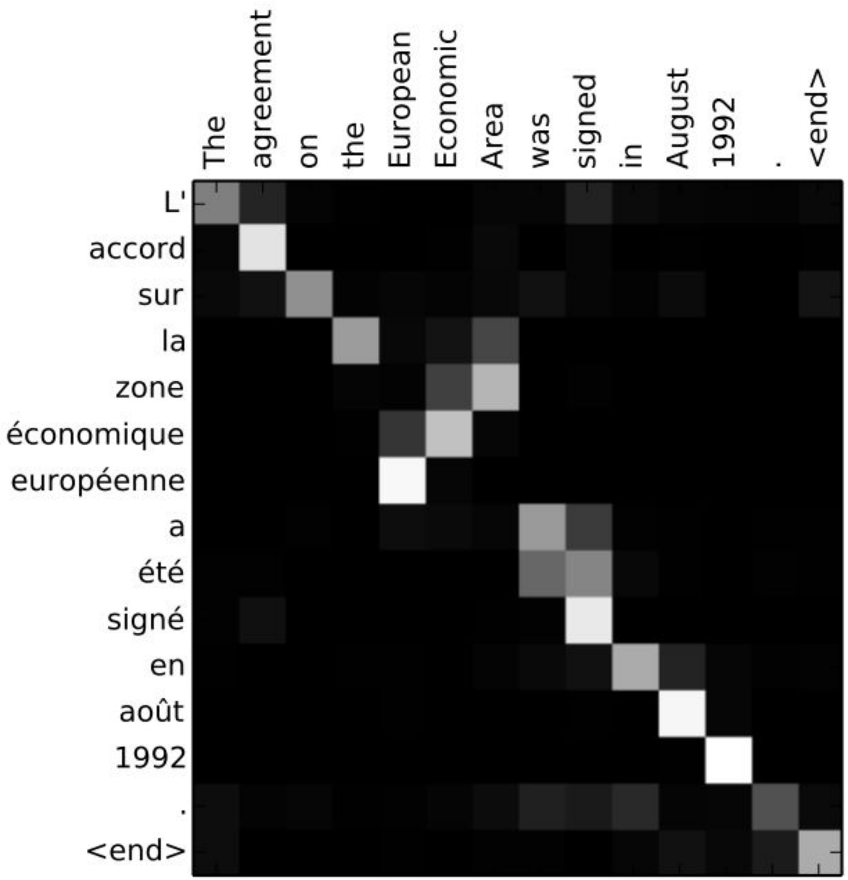


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate]

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

Self-Attention

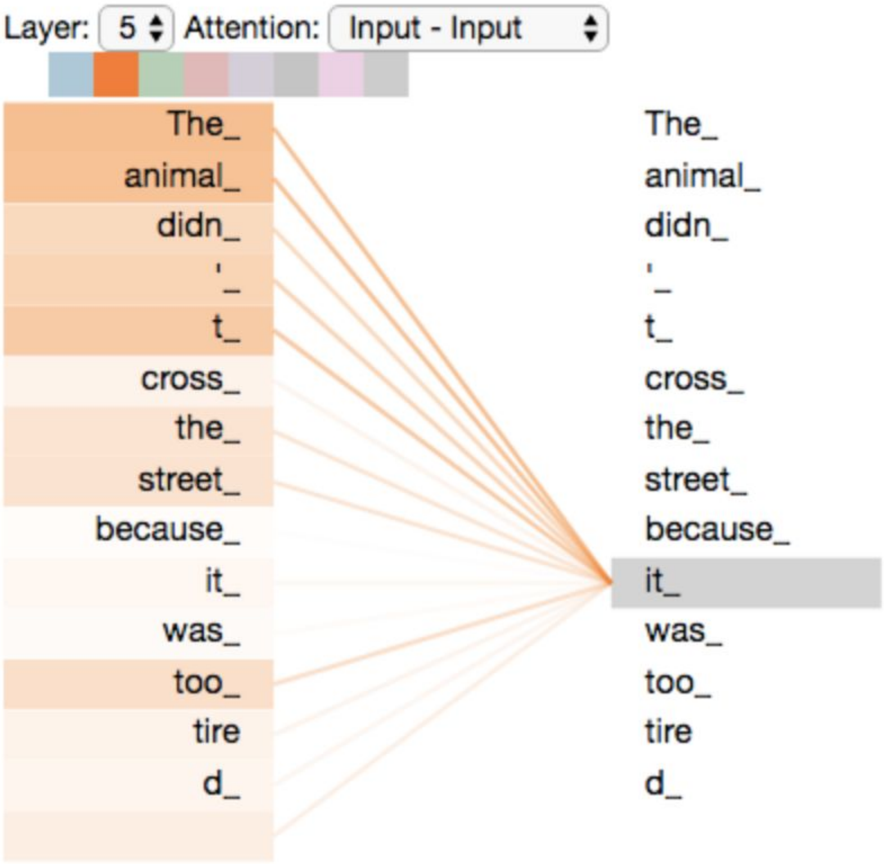
Self-Attention at a High Level

”`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

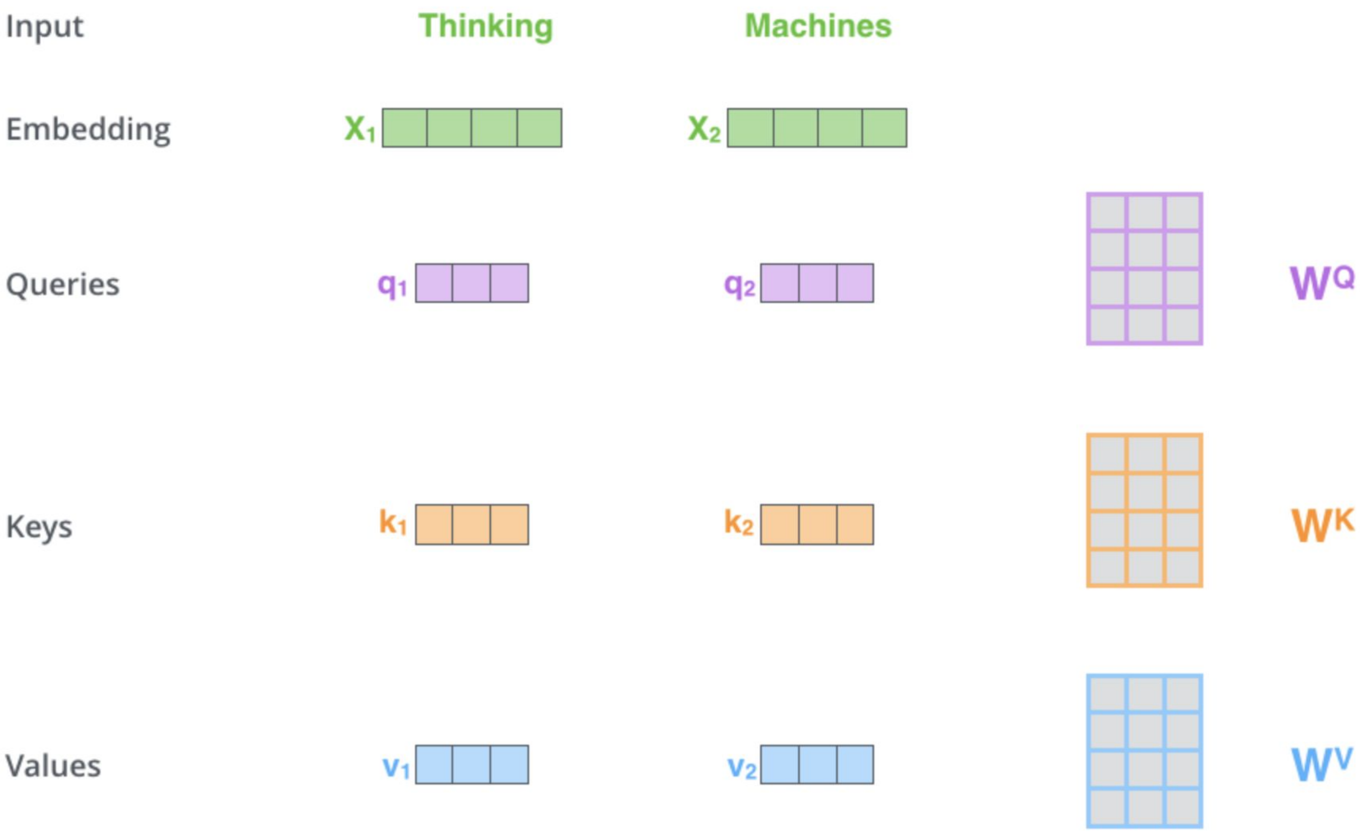


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

<u>STEP 1:</u>

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

from each of the encoder's input vectors

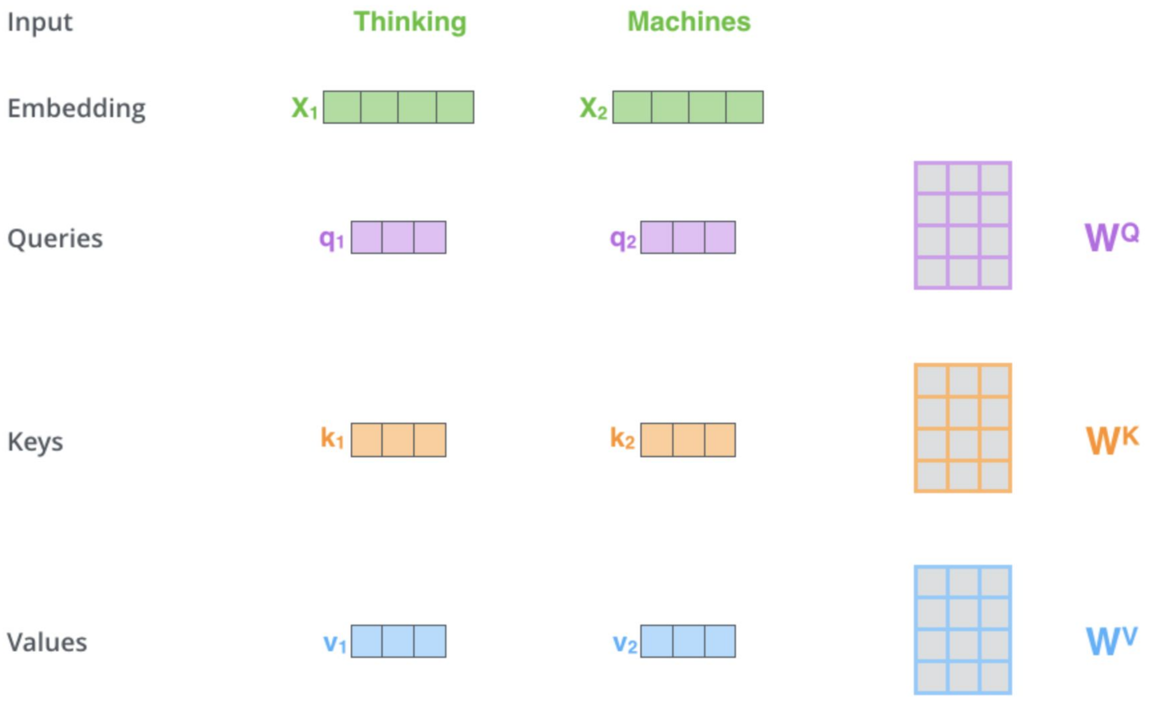


Image source: https://jalammar.github.io/illustrated-transformer/

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

<u>STEP 2:</u>

calculate a score

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

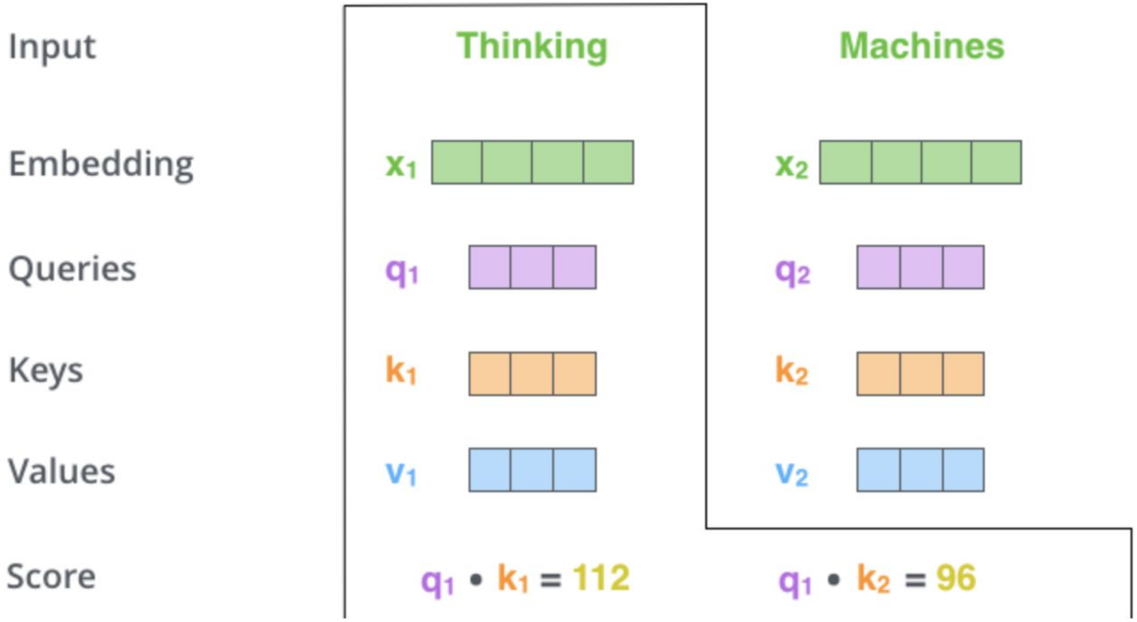


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 3:

divide the scores by 8

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

STEP 4:

softmax

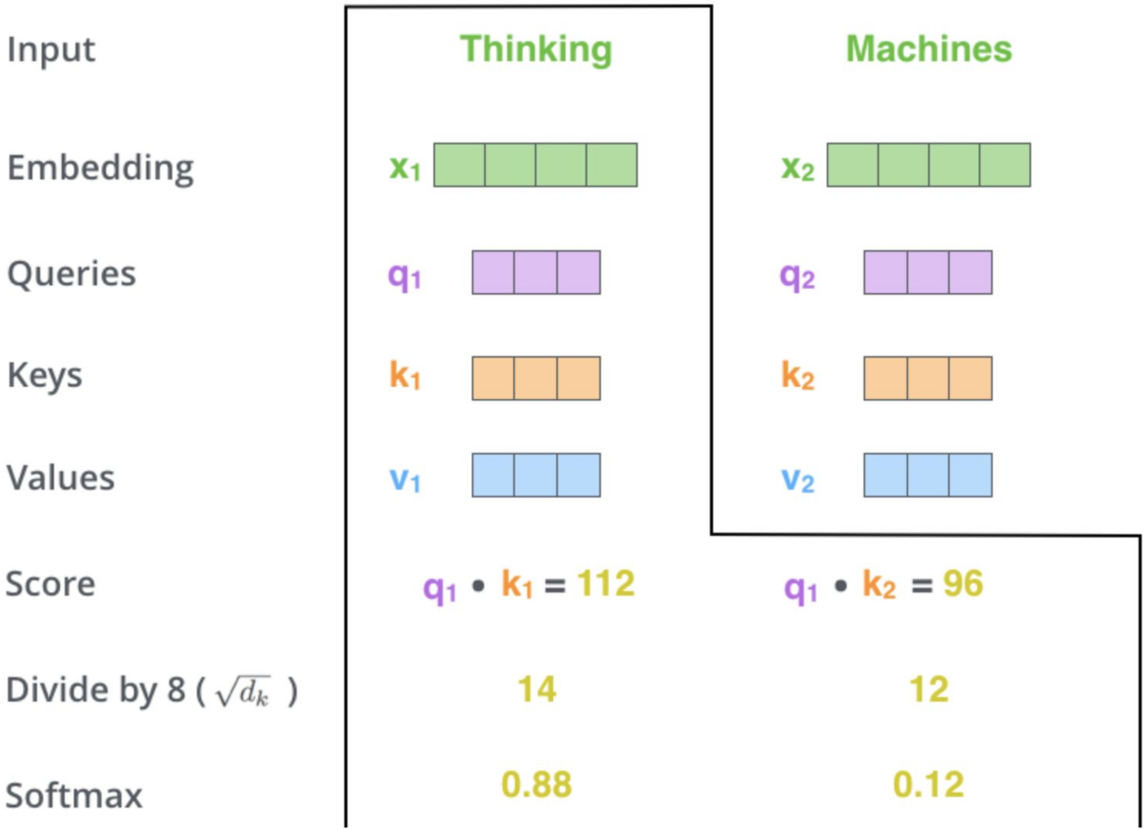


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors

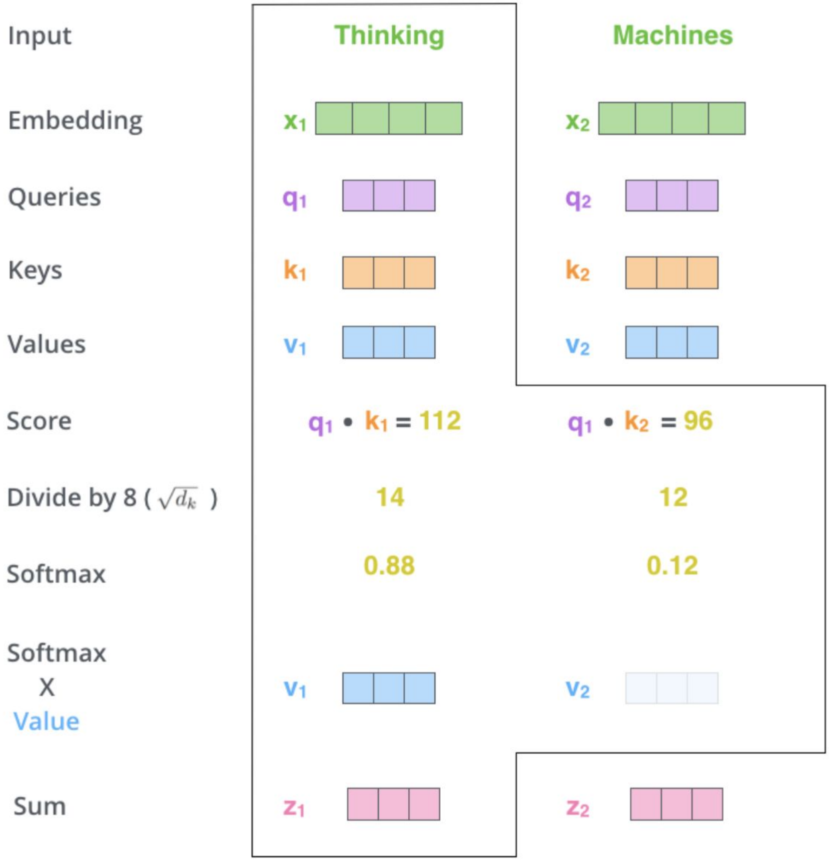
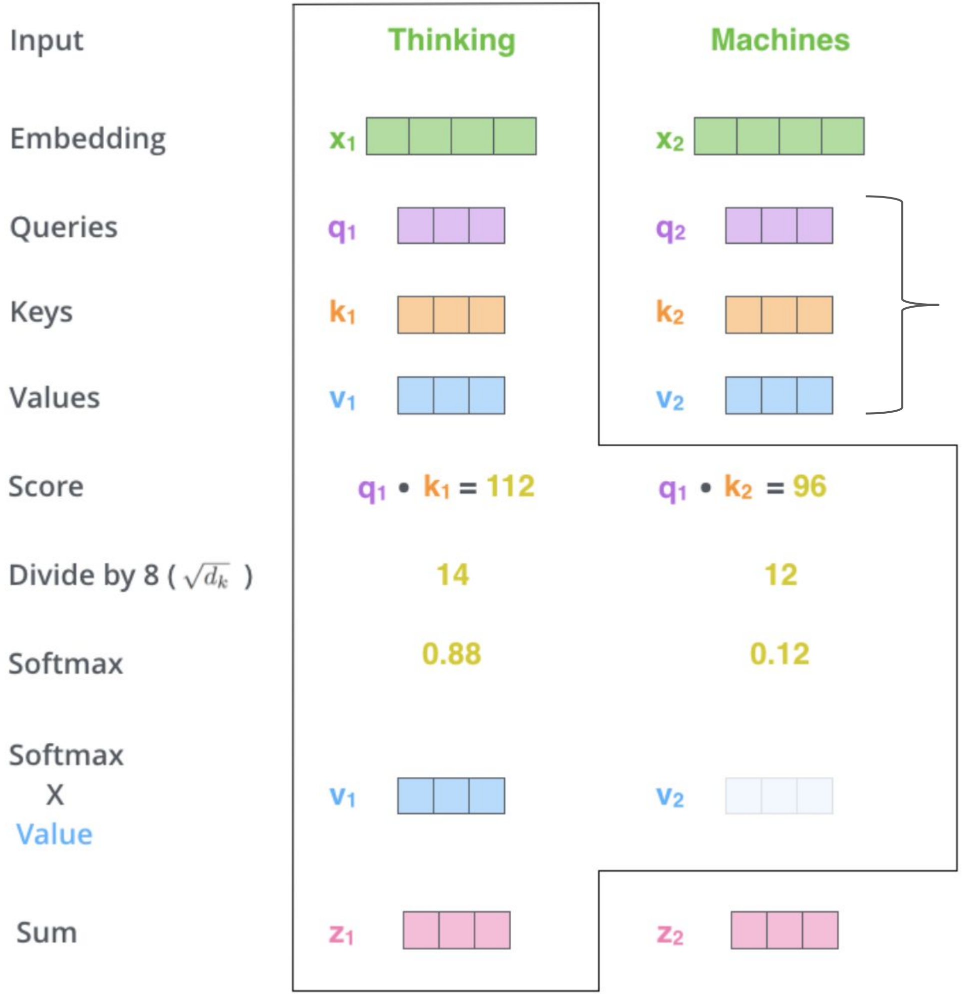


Image source: https://jalammar.github.io/illustrated-transformer/

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**)

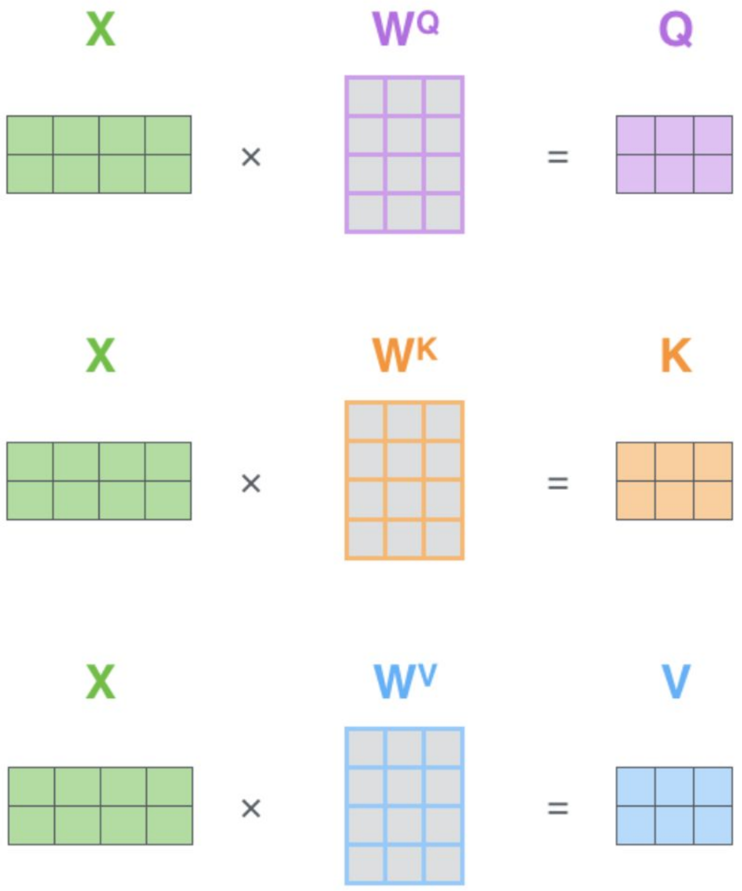


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

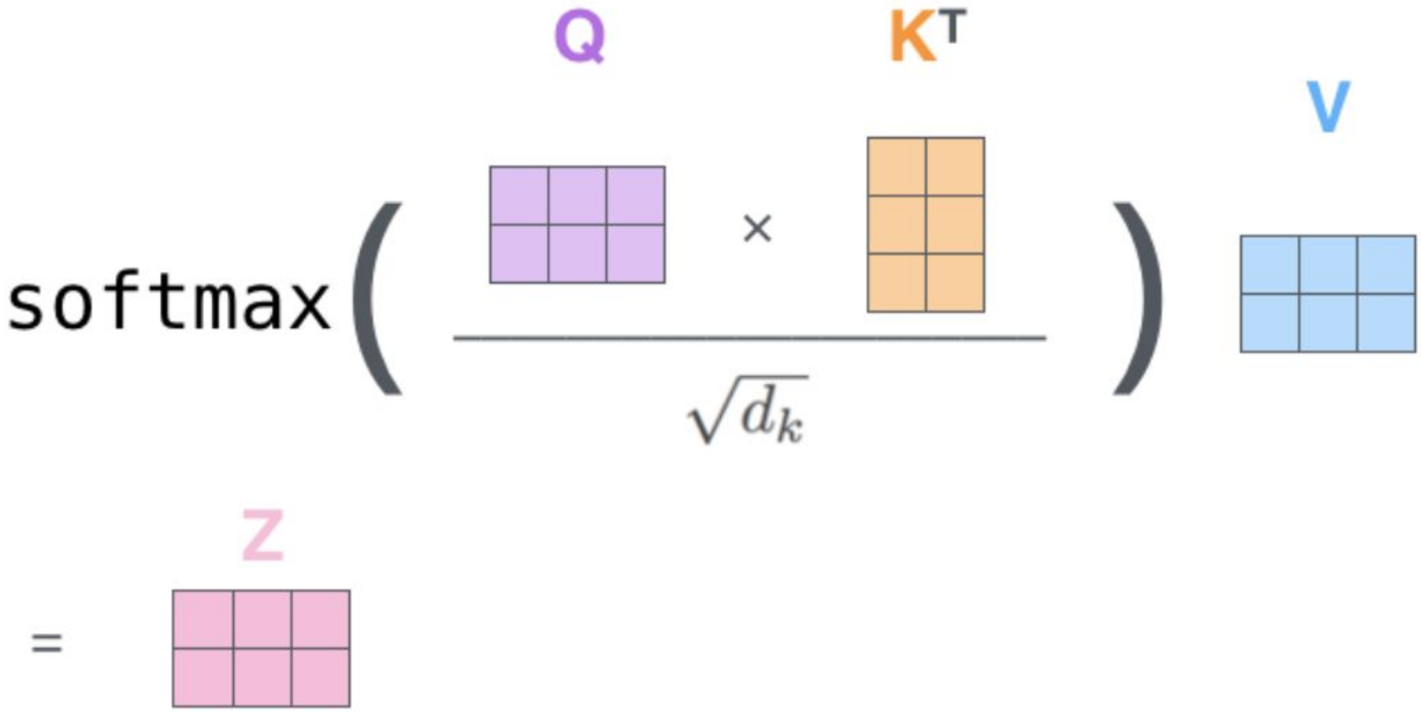


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

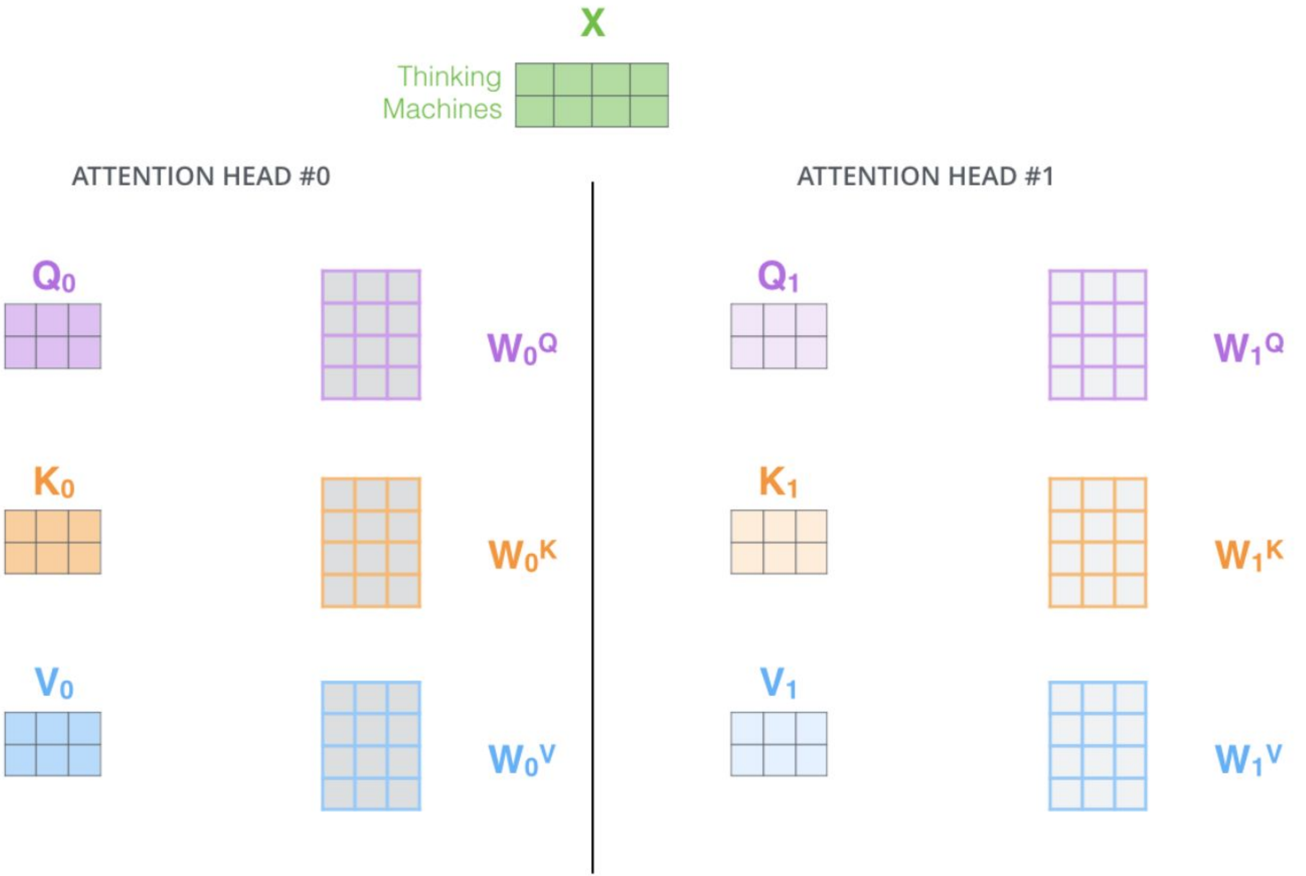


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

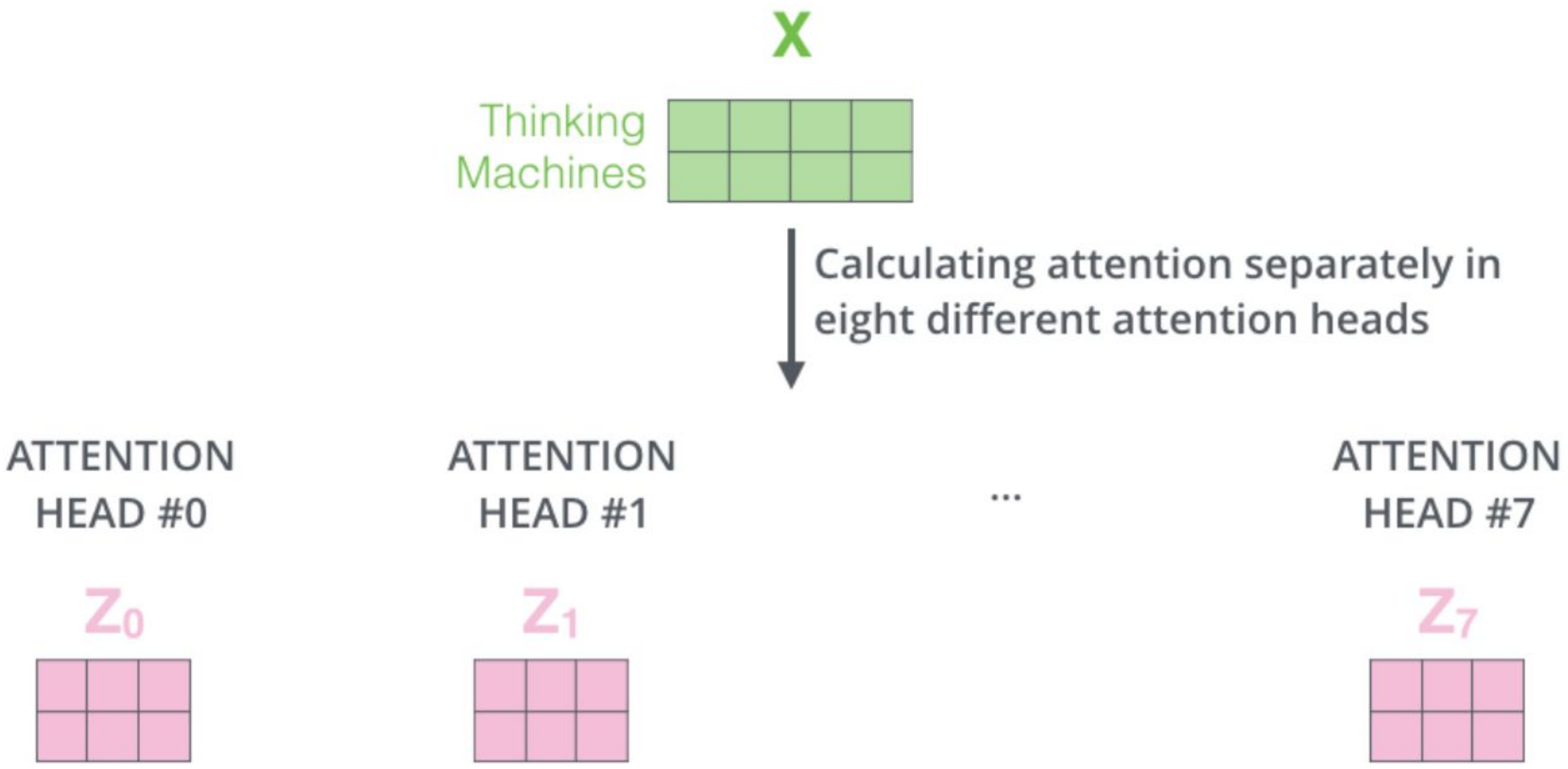


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

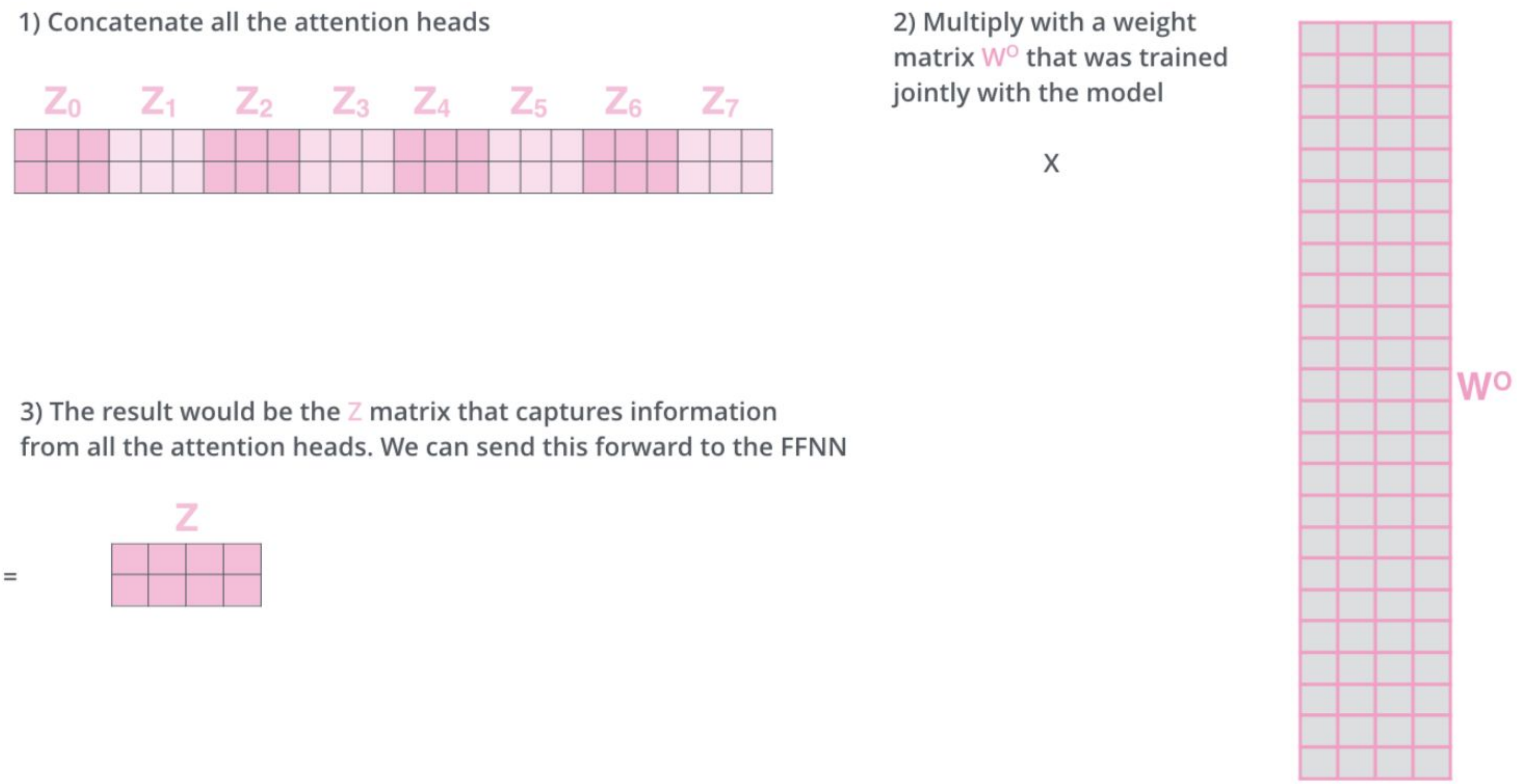


Image source: https://jalammar.github.io/illustrated-transformer/

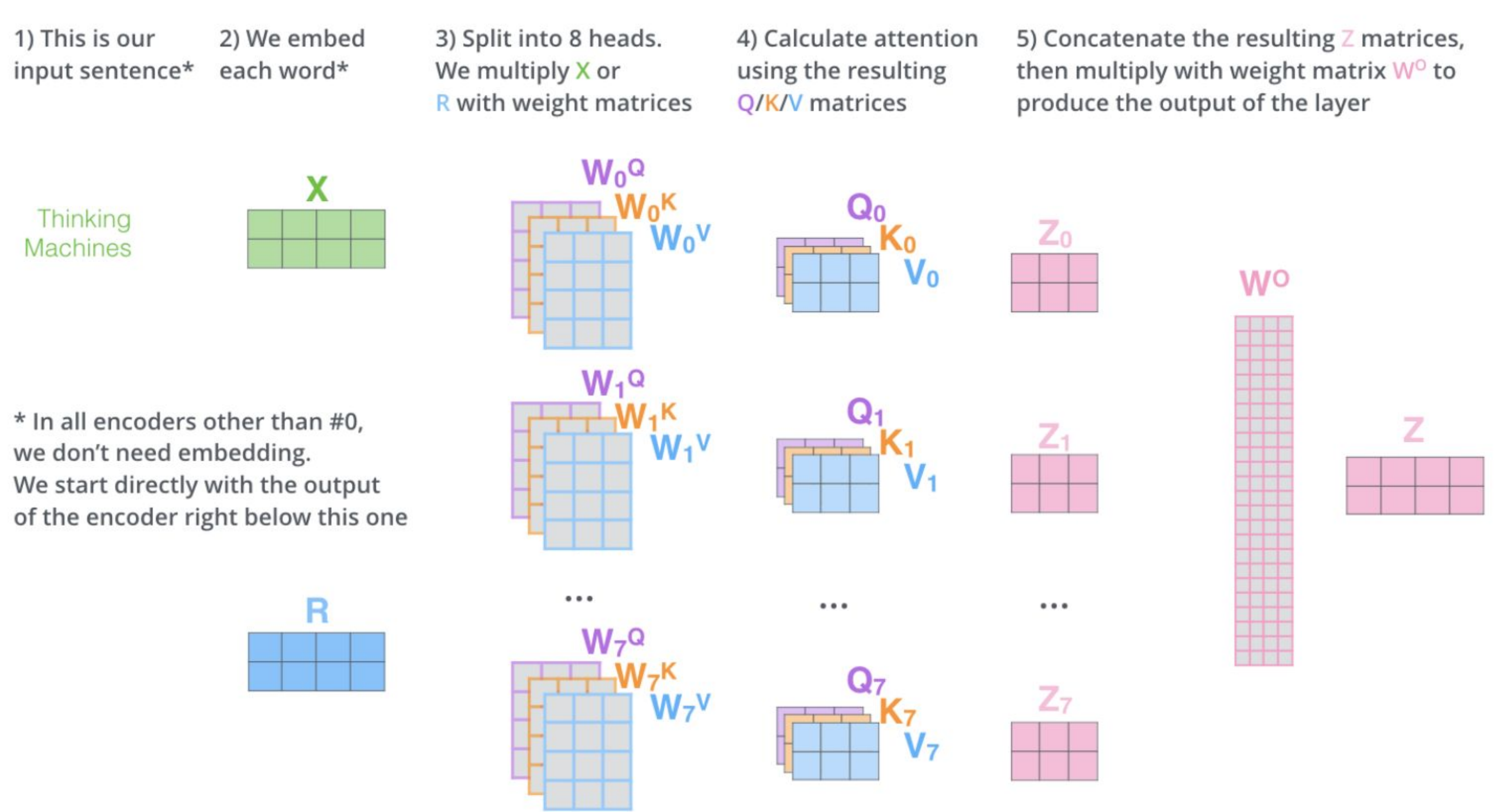
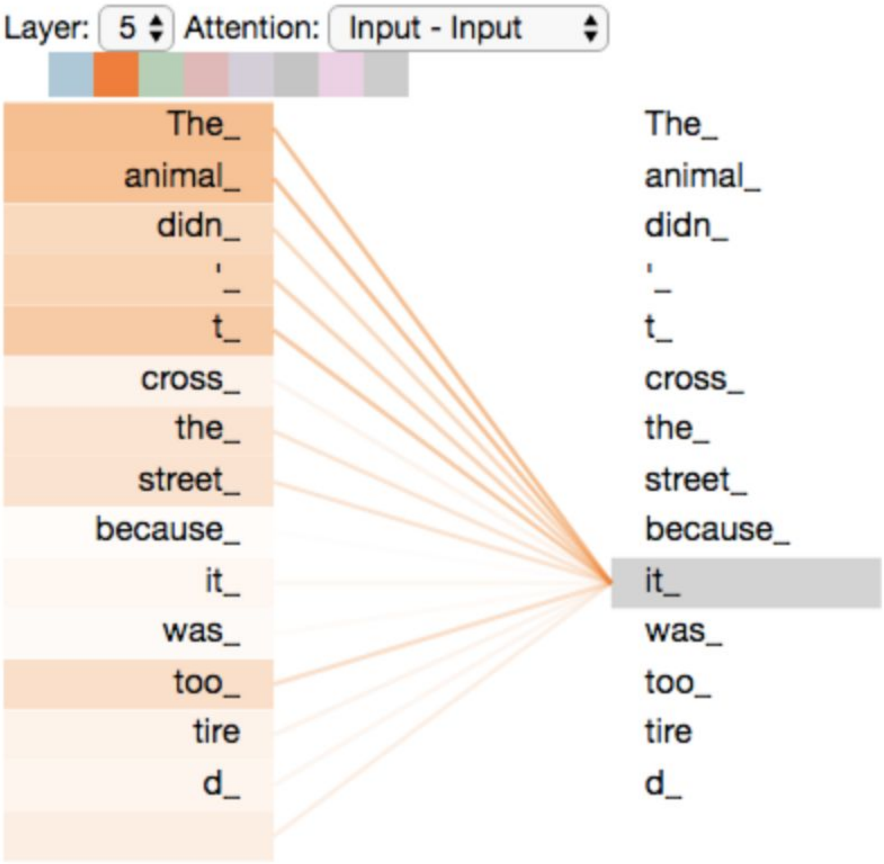
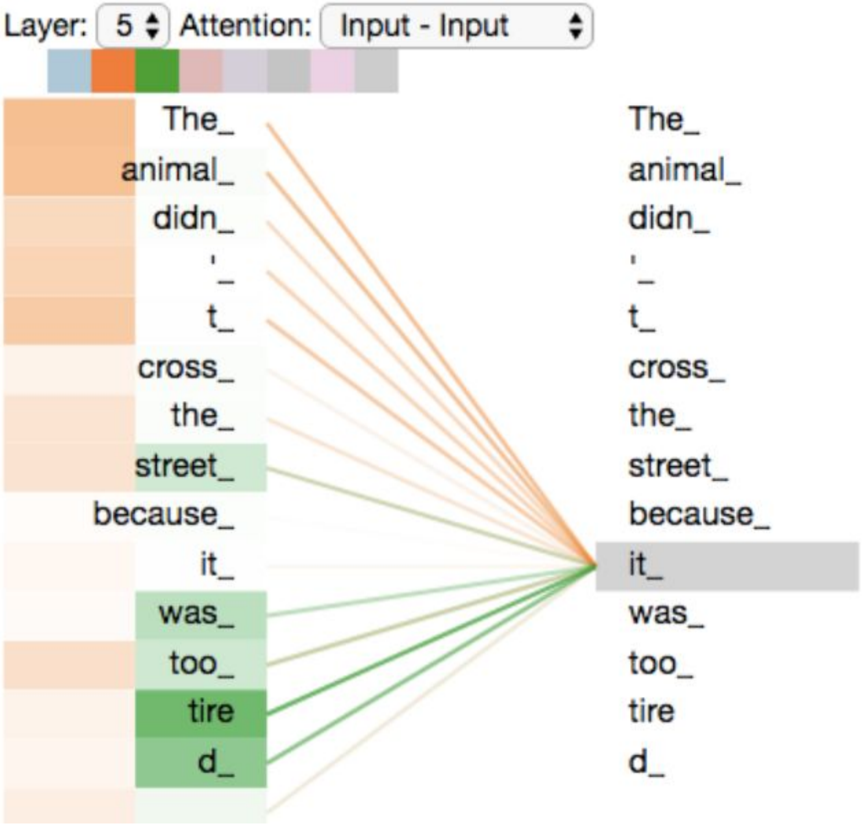


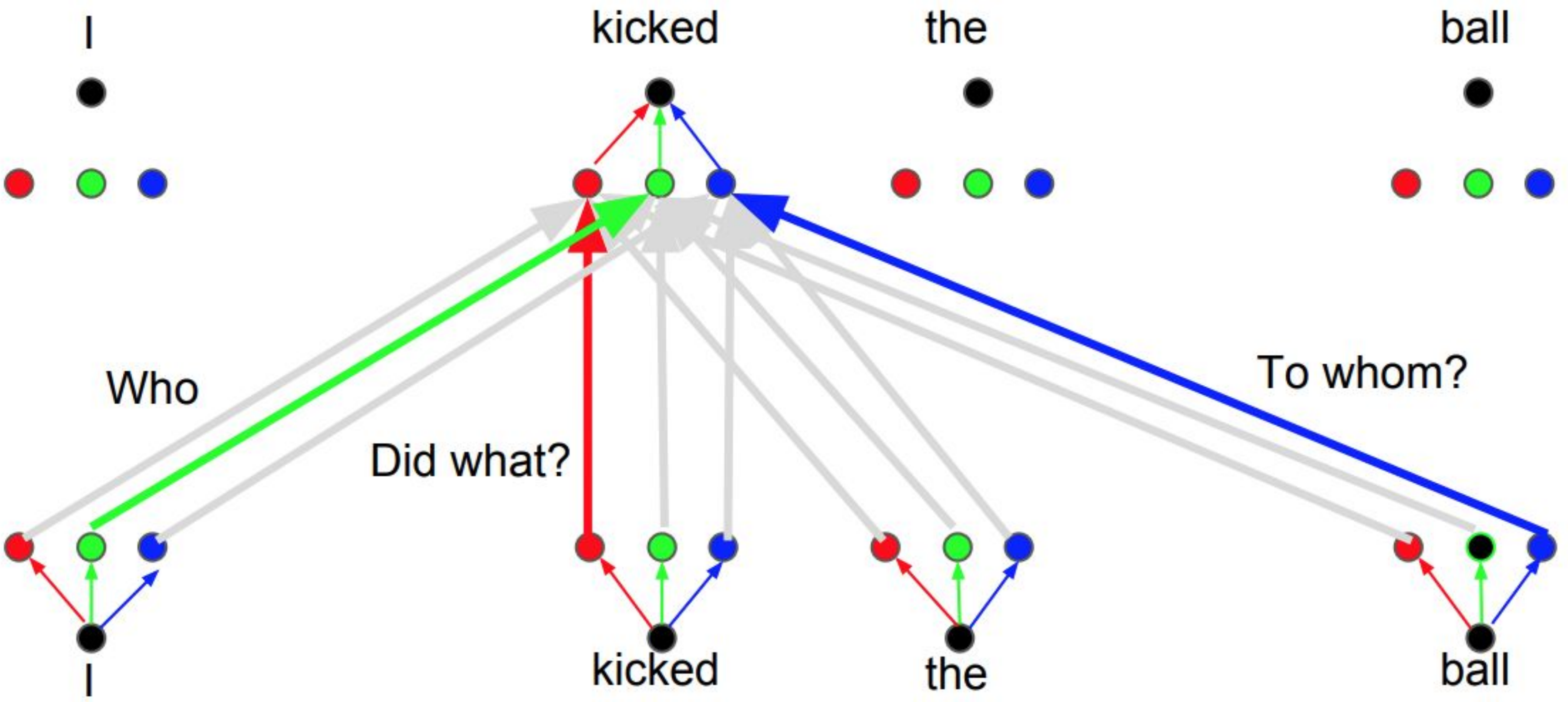
Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

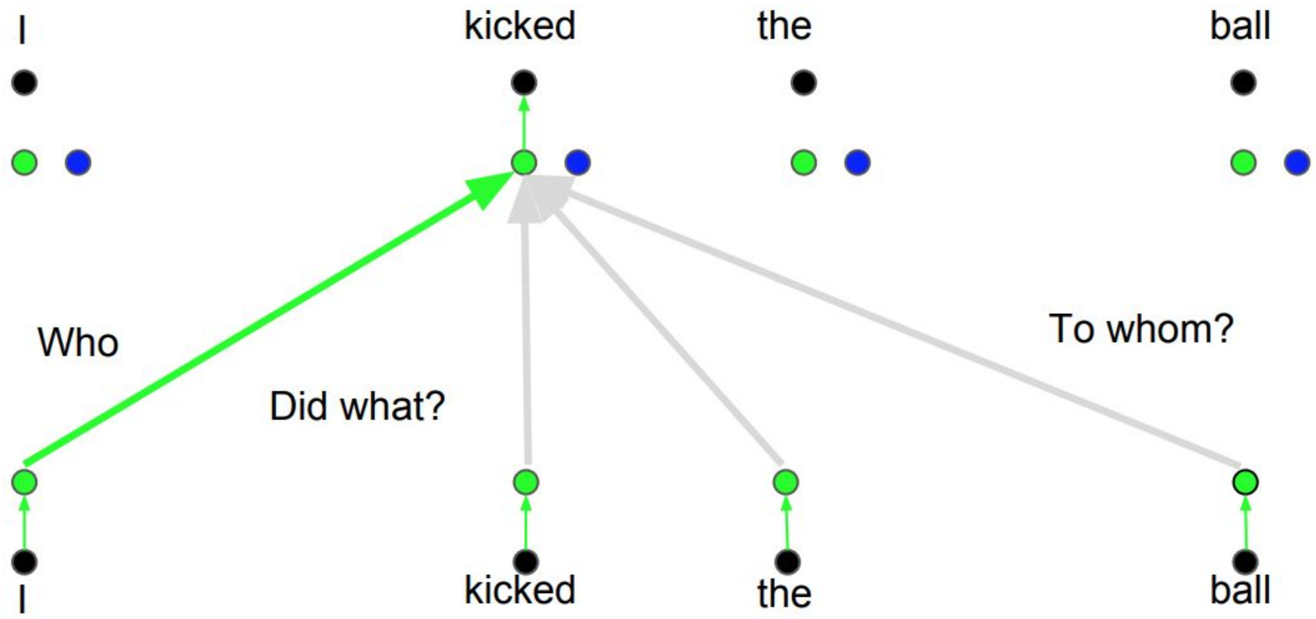




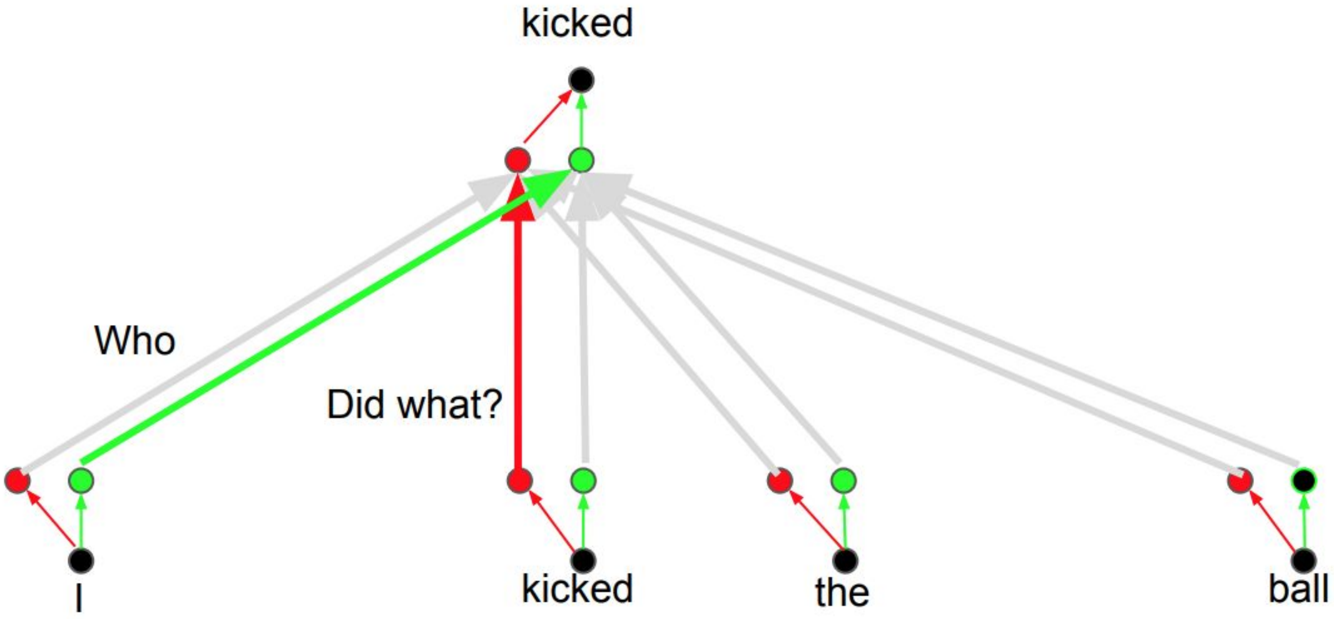
Why Multi-Head Attention?



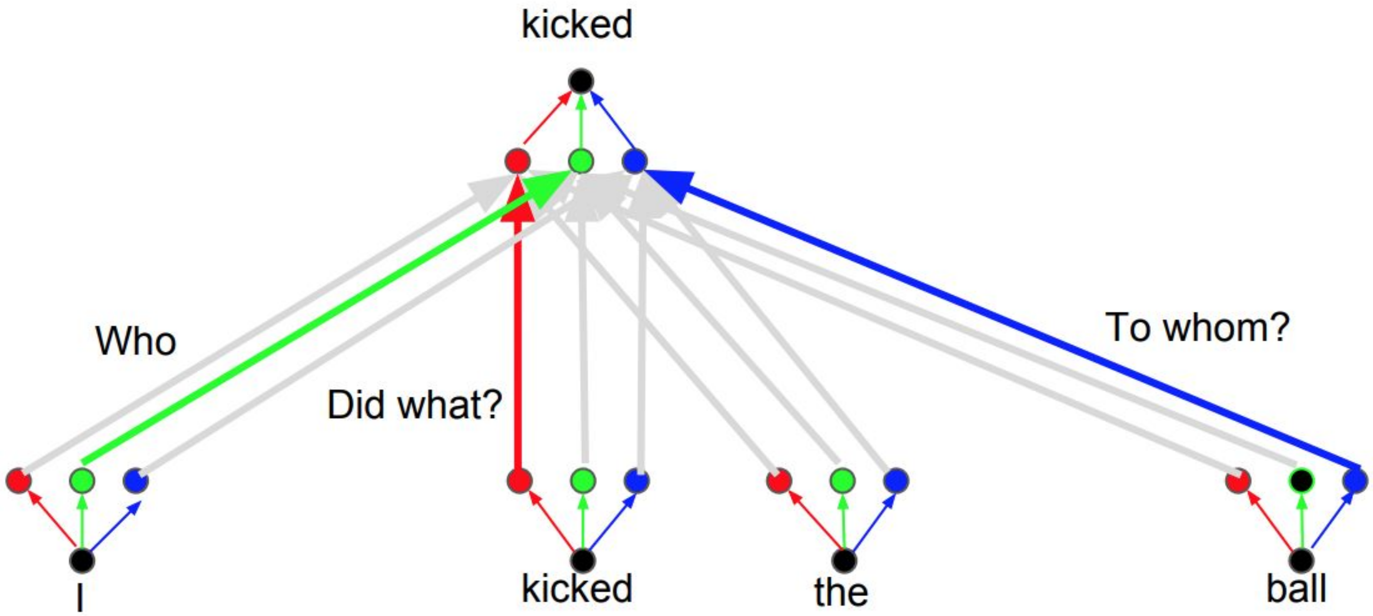
Attention head: Who



Attention head: Did What?

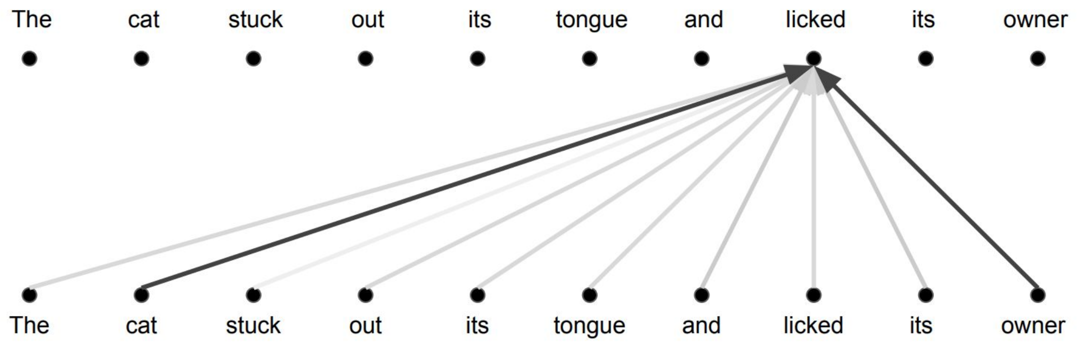


Attention head: To Whom?



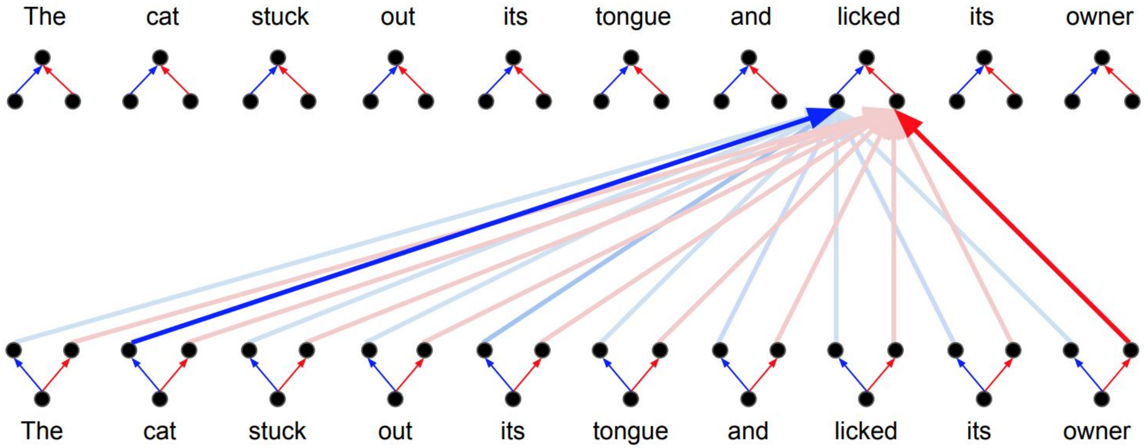
Attention vs. Multi-Head Attention

Attention: a weighted average



Multi-Head Attention:

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



Outro

- Attention mechanism allows to “attend all positions” in the original sequence (or any other input with internal structure)
- Attention mechanism requires more computational resources than original seq2seq models
- Change of the model architecture affects the training procedure, so be careful with intuitive explanations

Transformer outro

The Transformer

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

The Transformer

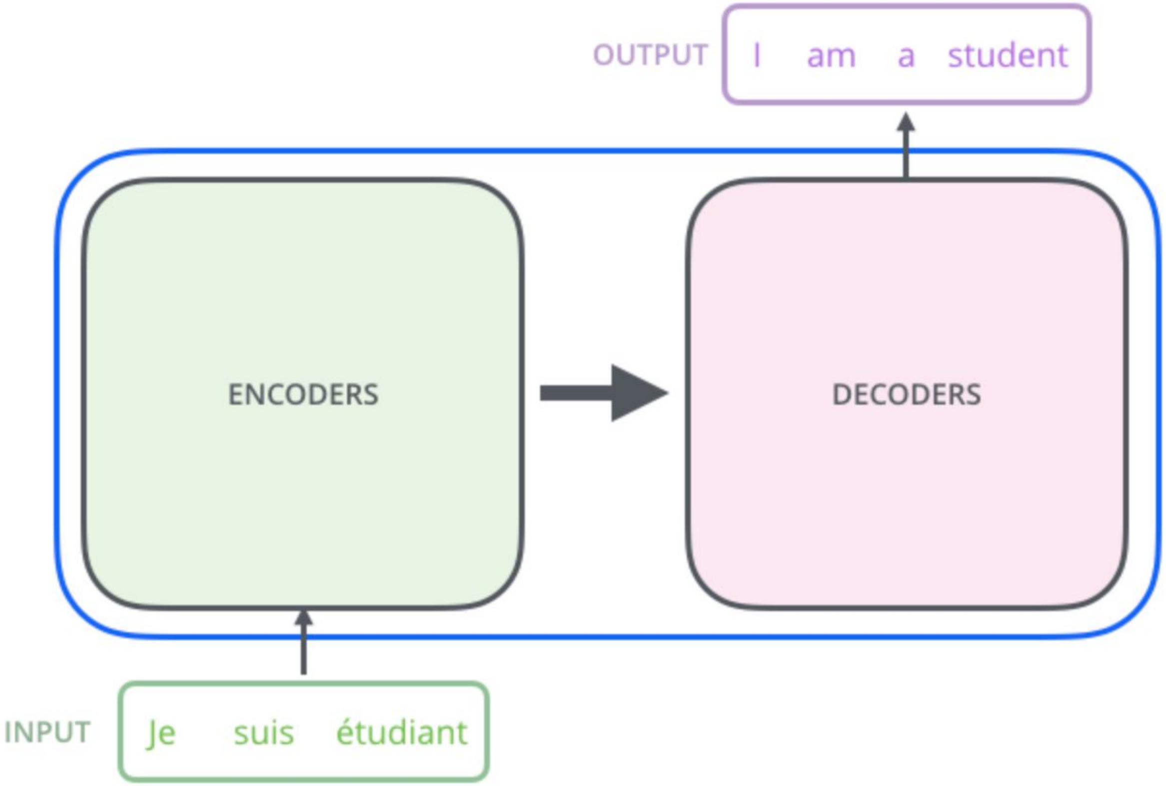


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

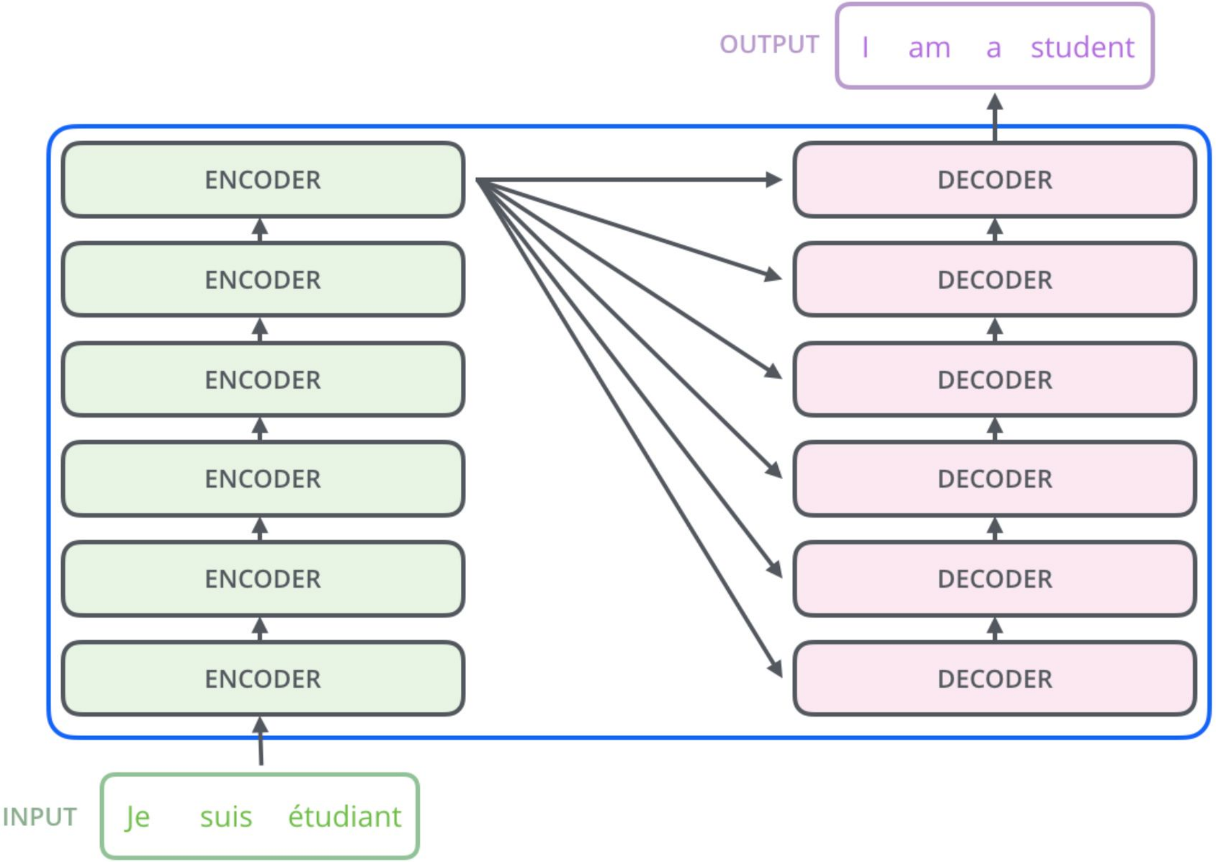
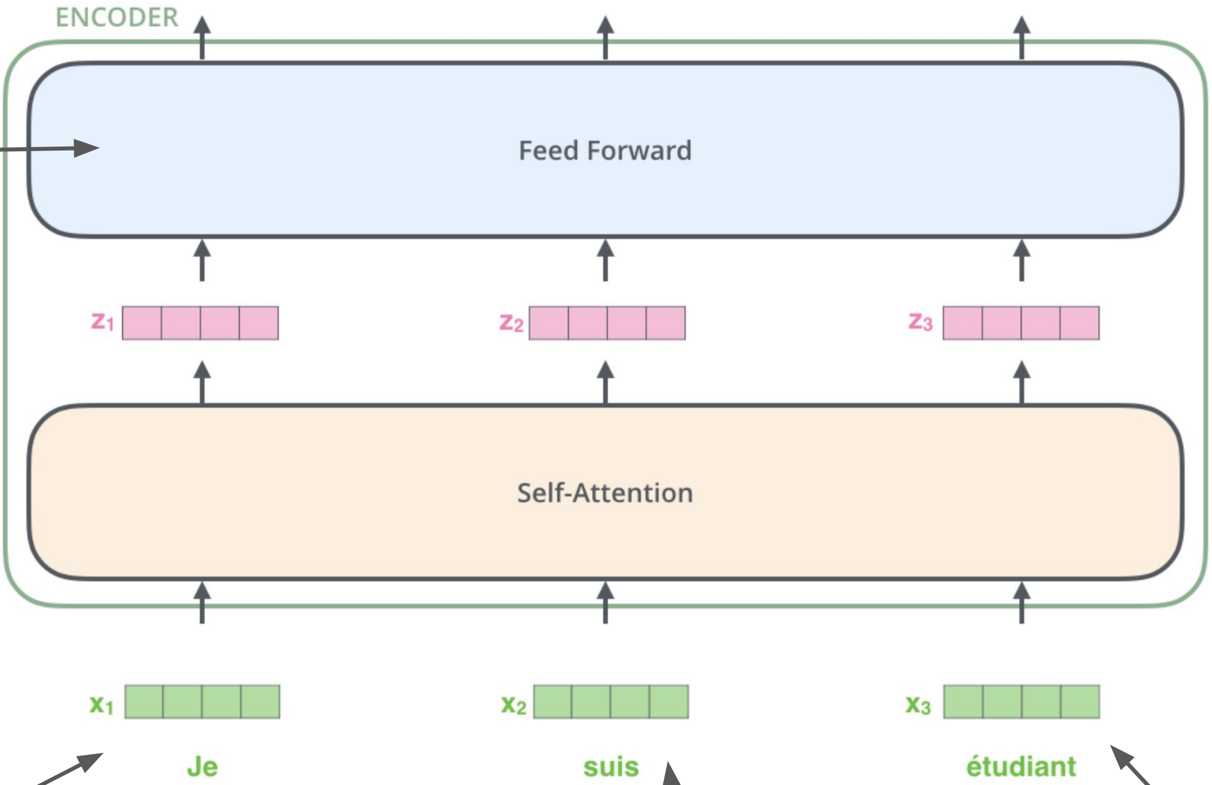


Image source: https://jalammar.github.io/illustrated-transformer/

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/

Layer Normalization

The Transformer: recap

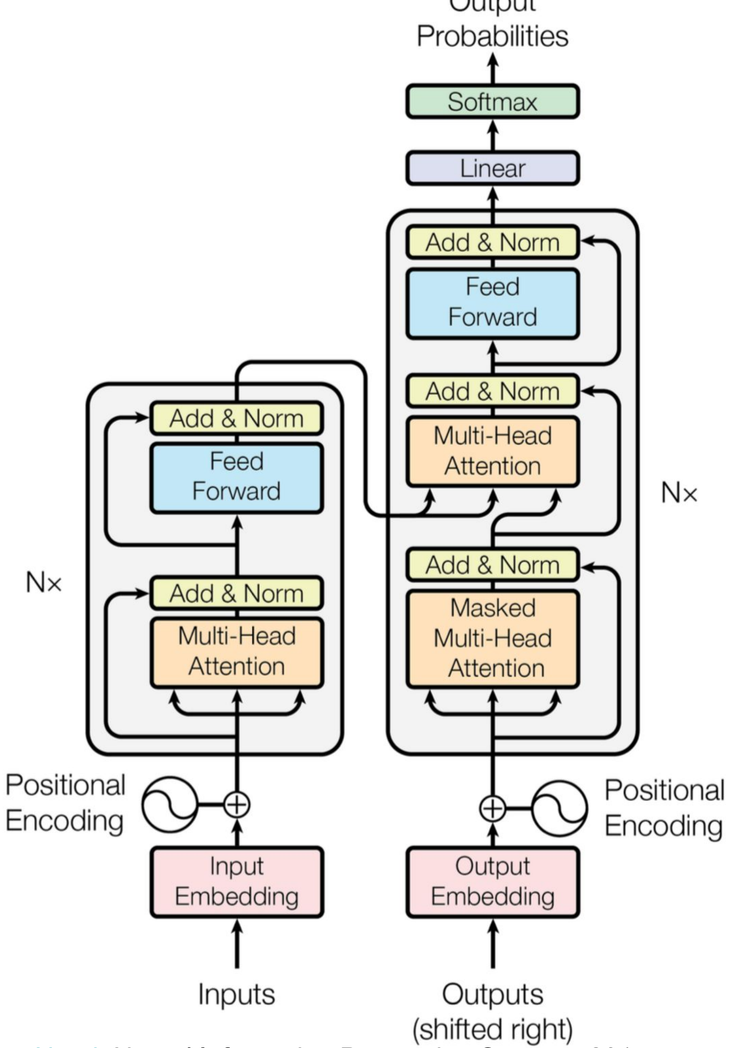


Image source: [Attention Is All You Need](), Neural Information Processing Systems 2017

Layer Normalization

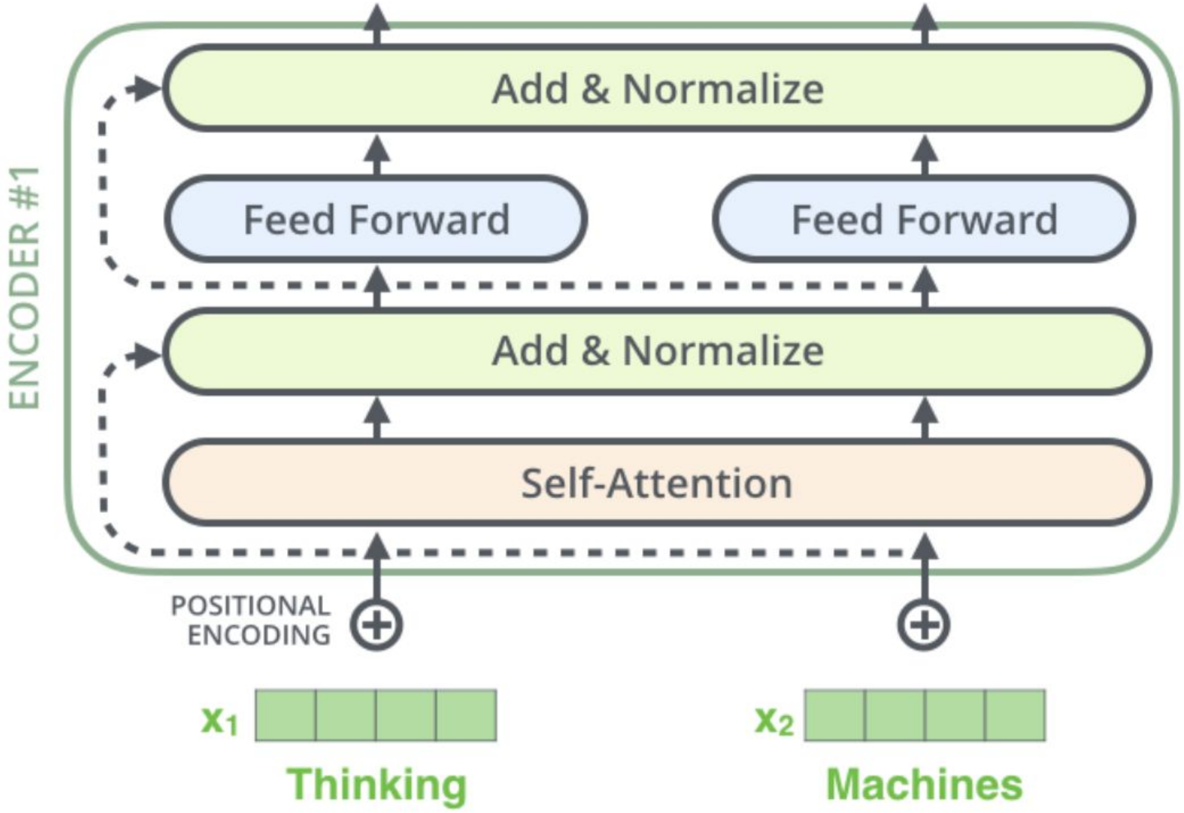
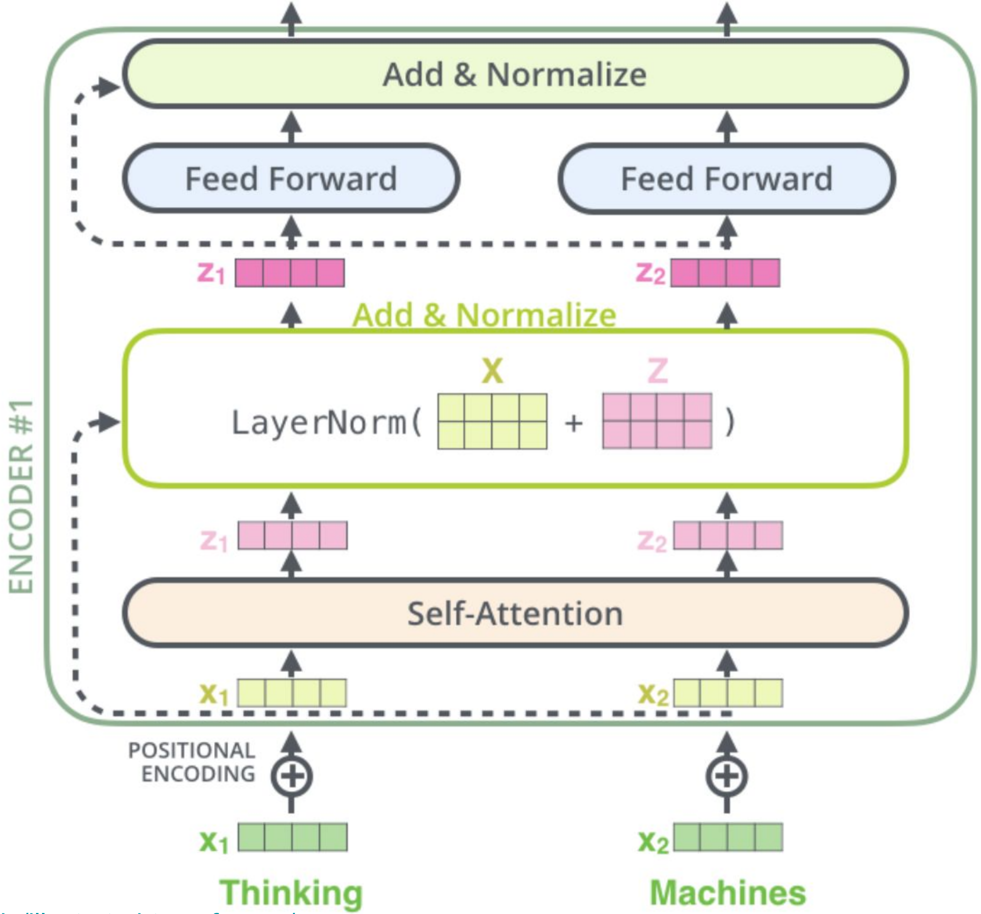


Image source: https://jalammar.github.io/illustrated-transformer/

Layer Normalization

Like BatchNorm

but normalize along all features representing latent vector



More info: Layer Normalization

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

Layer Normalization

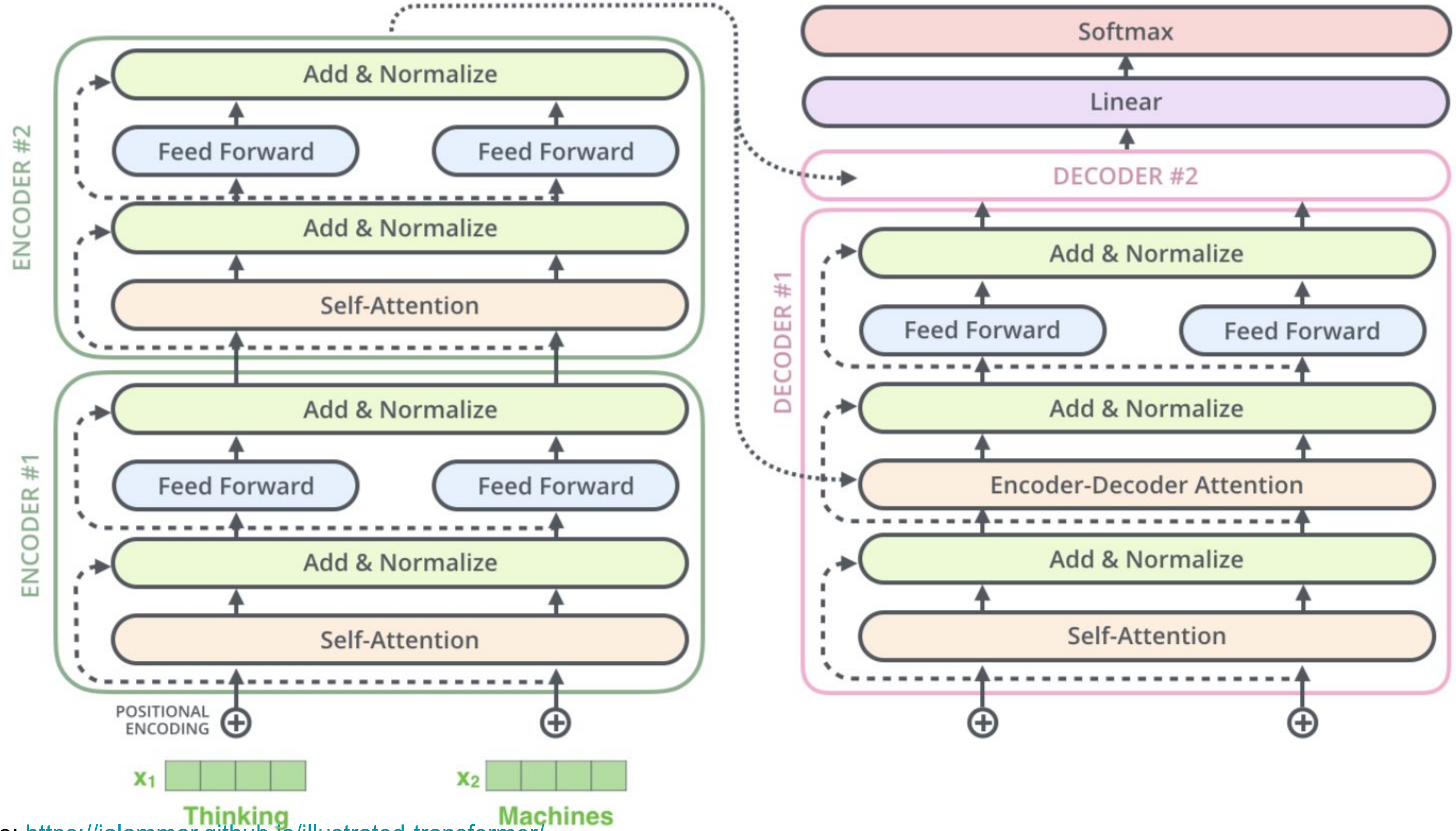


Image source: https://jalammar.github.io/illustrated-transformer/

The Decoder

The Decoder Side

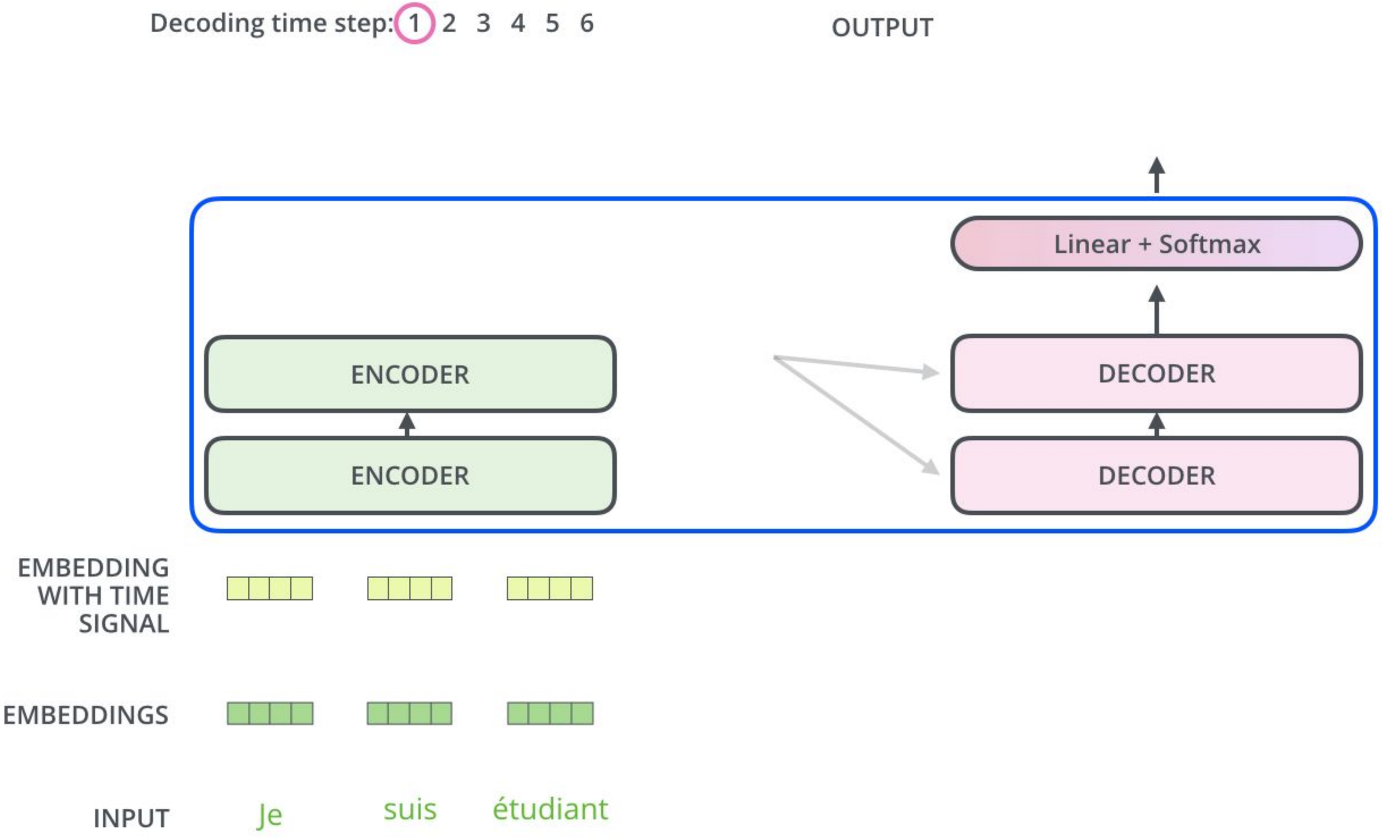
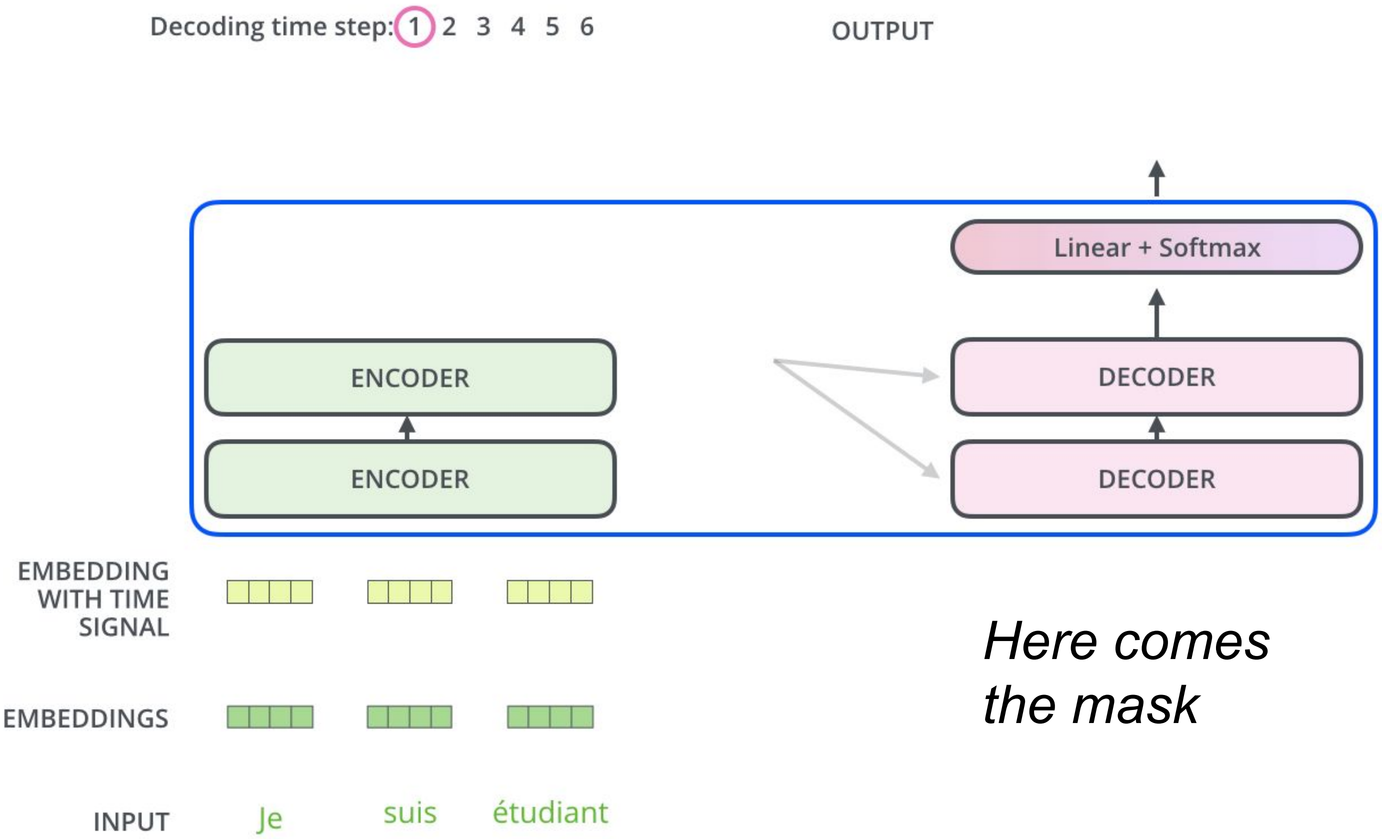


Image source: https://jalammar.github.io/illustrated-transformer/

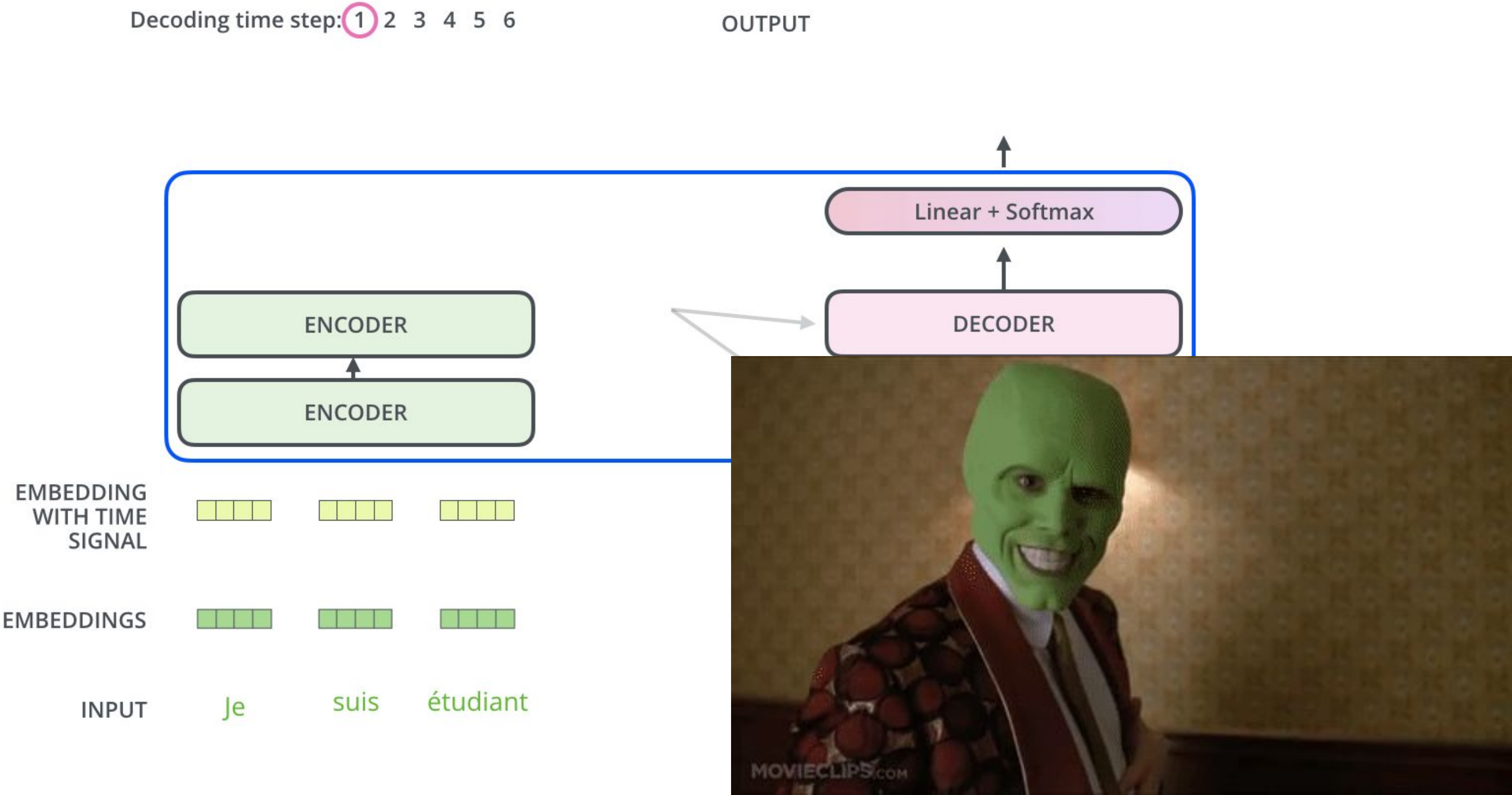
The Decoder Side



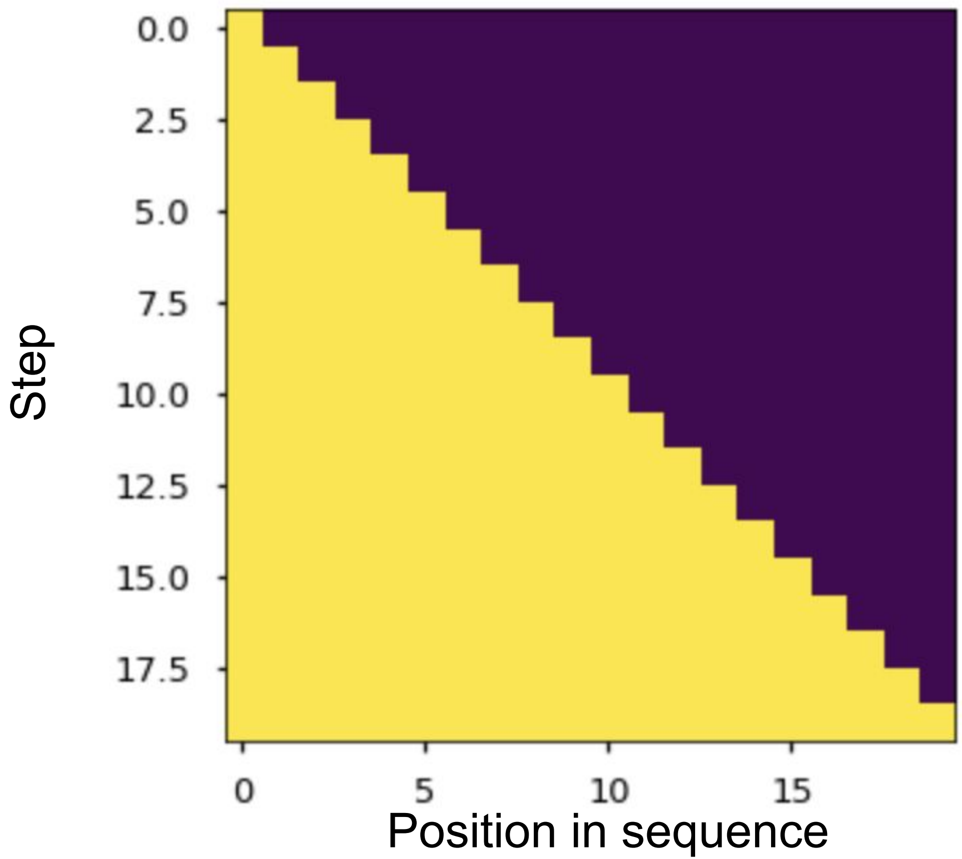
Here comes the mask

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

The Decoder Side



The masked decoder input



The Decoder Side

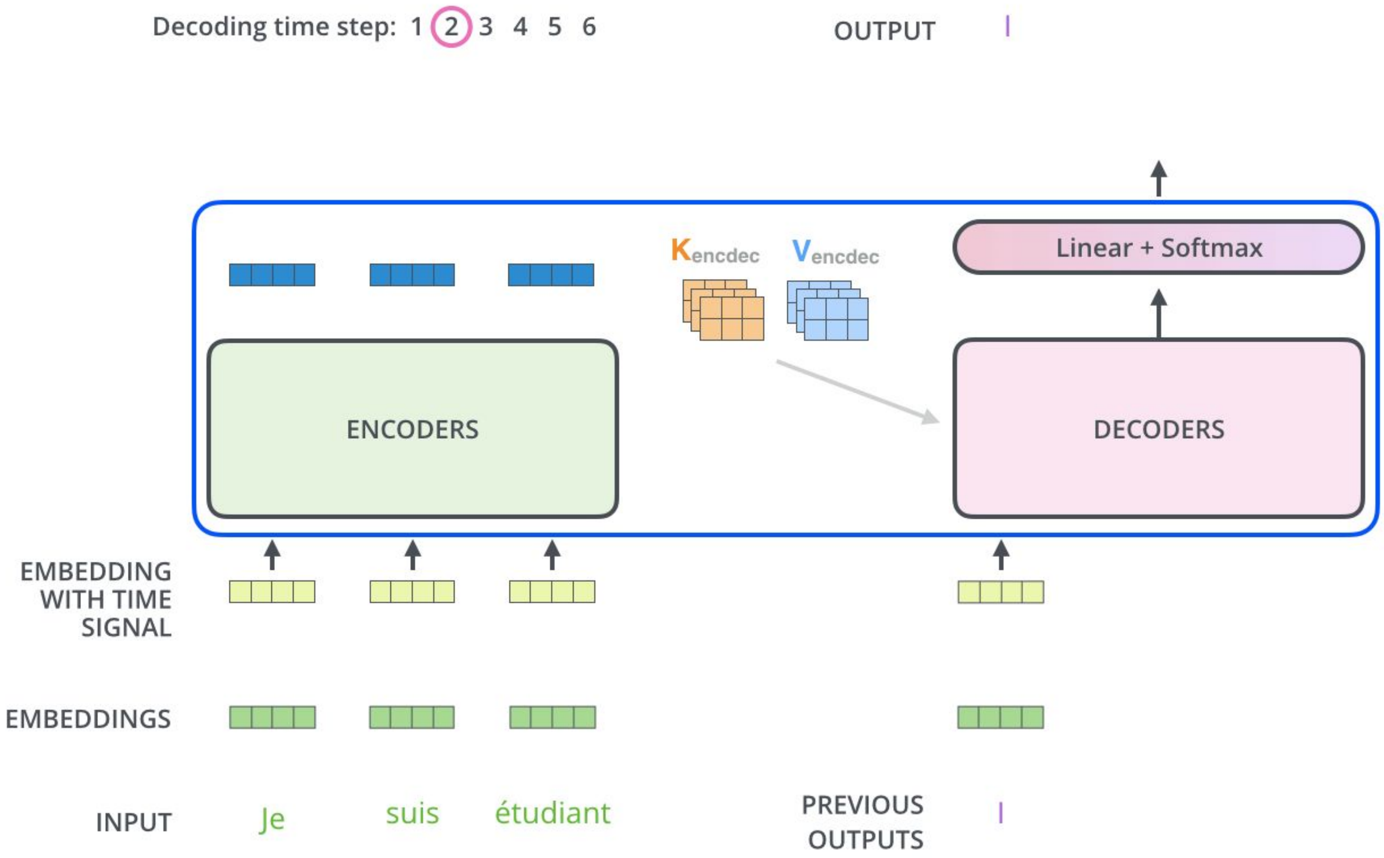


Image source: https://jalammar.github.io/illustrated-transformer/

Final Linear and Softmax Layer

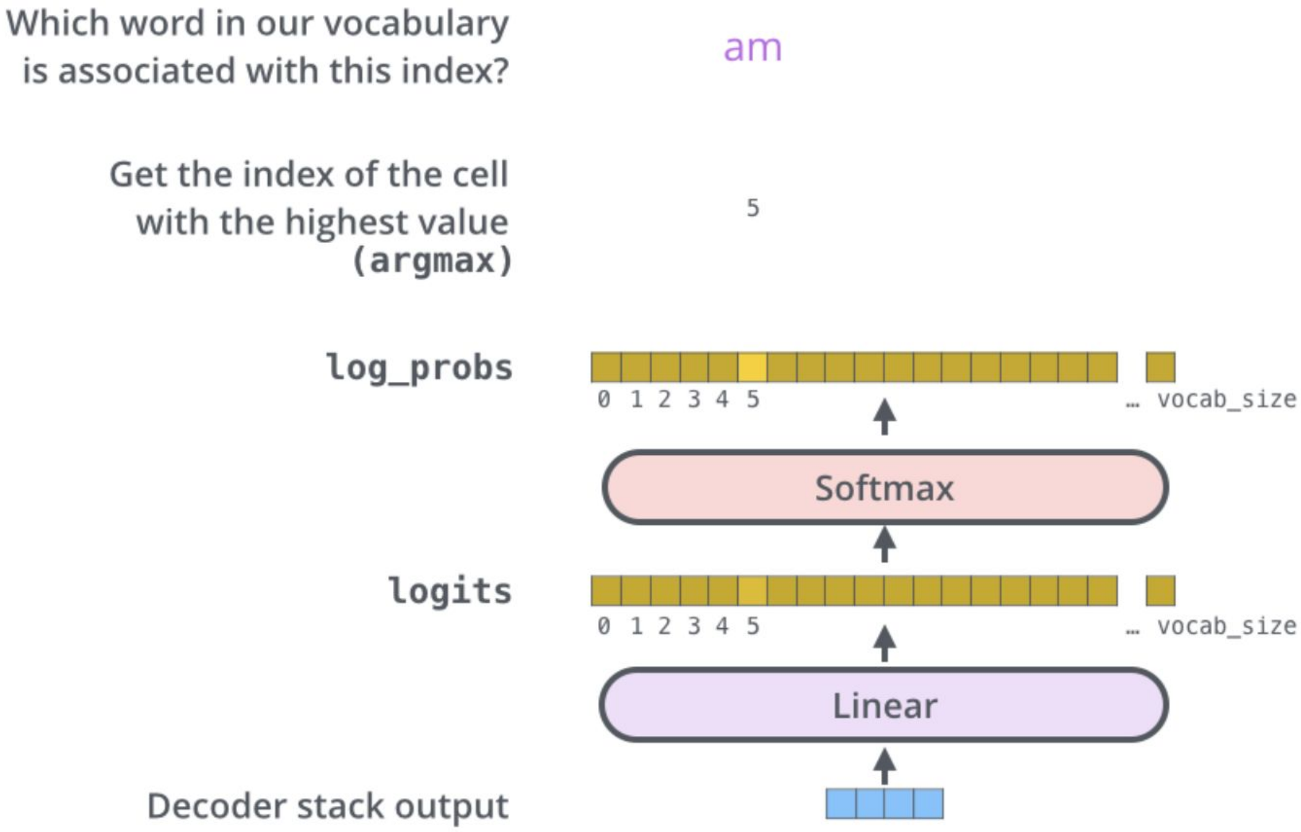


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

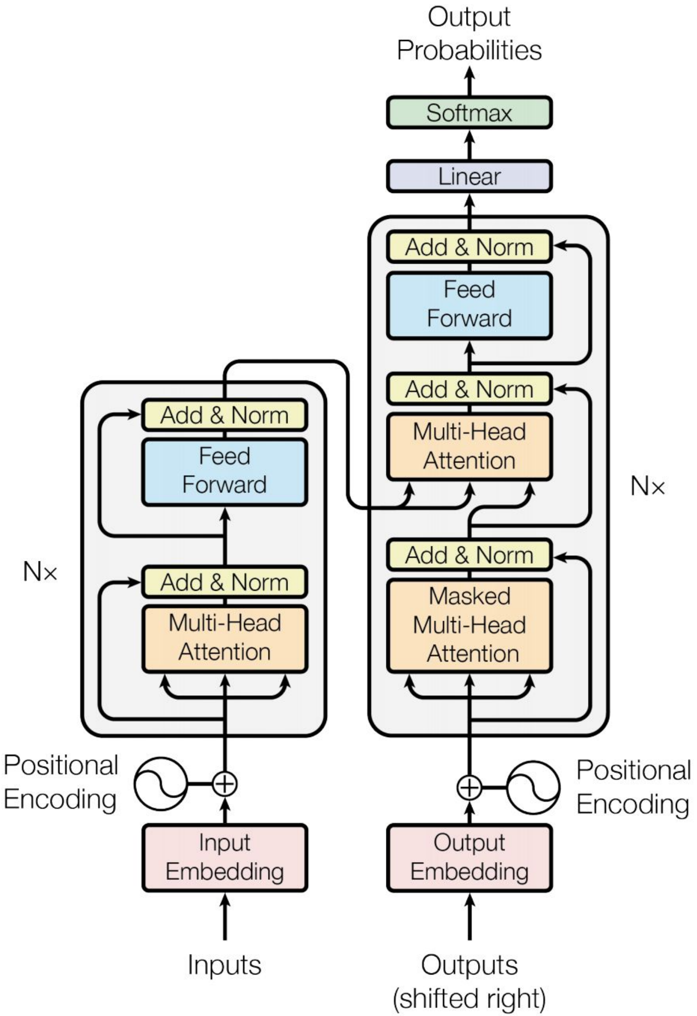


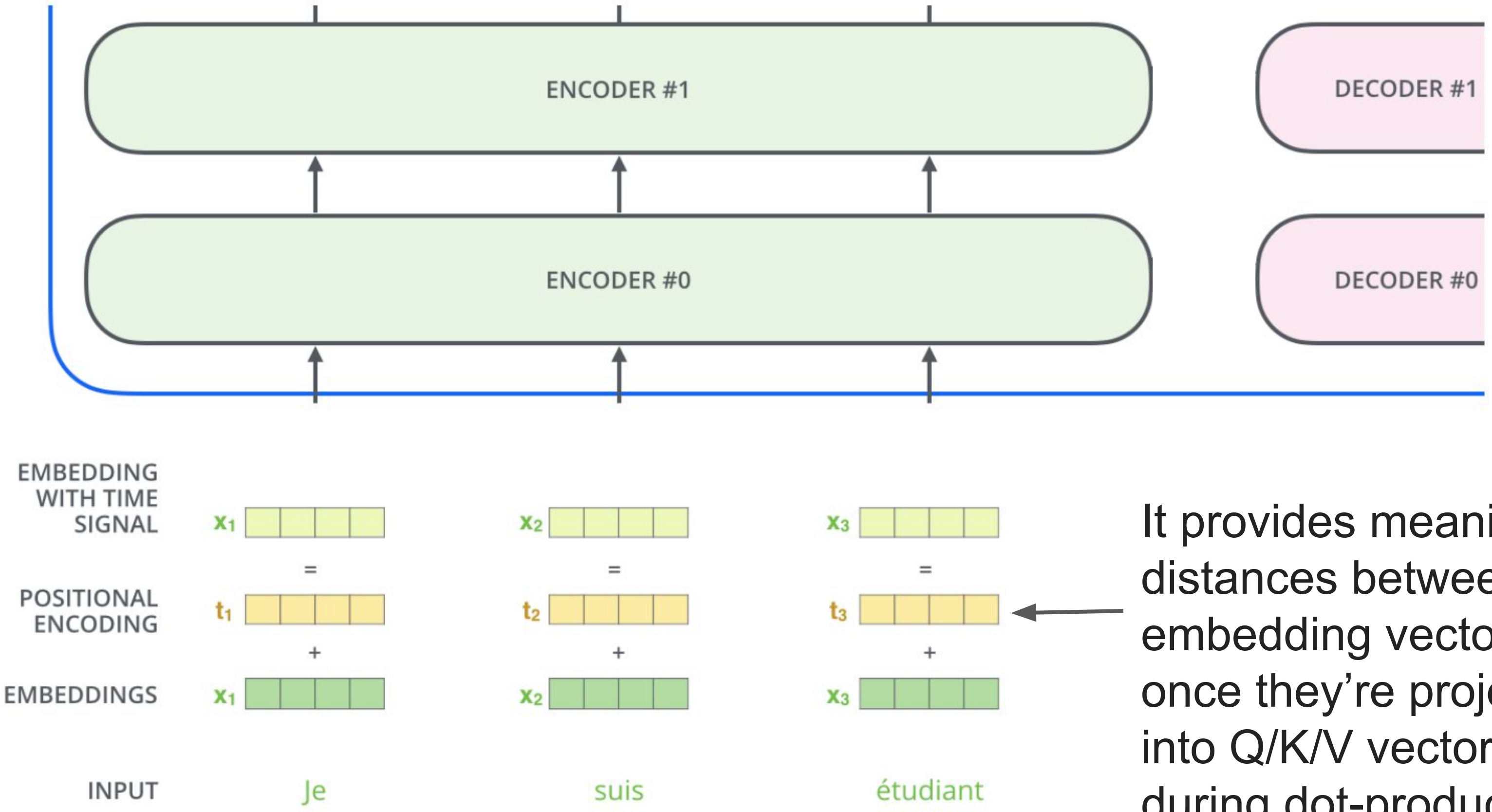
Image source: [Attention Is All You Need](), Neural Information Processing Systems 2017

Positional Encoding

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	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

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

Positional Encoding

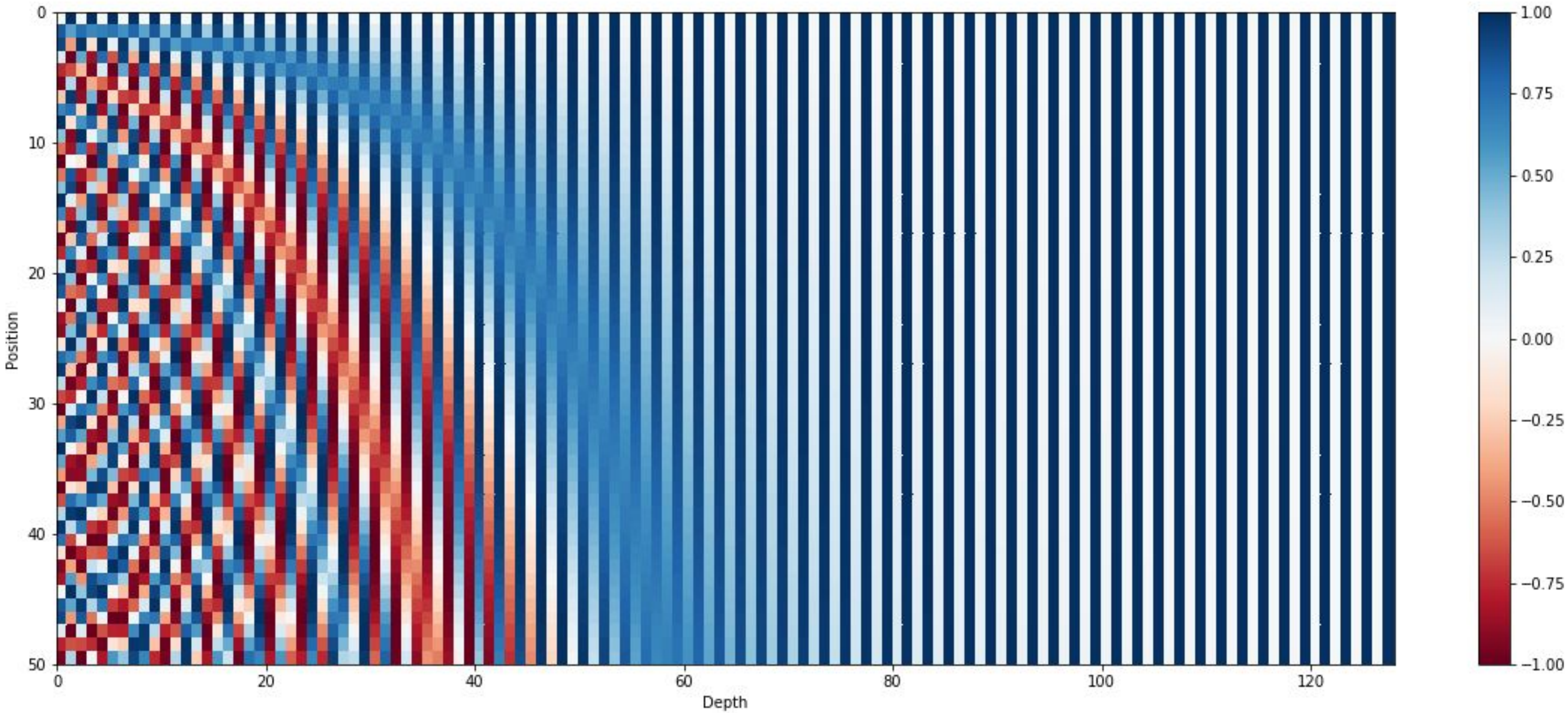


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

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/

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