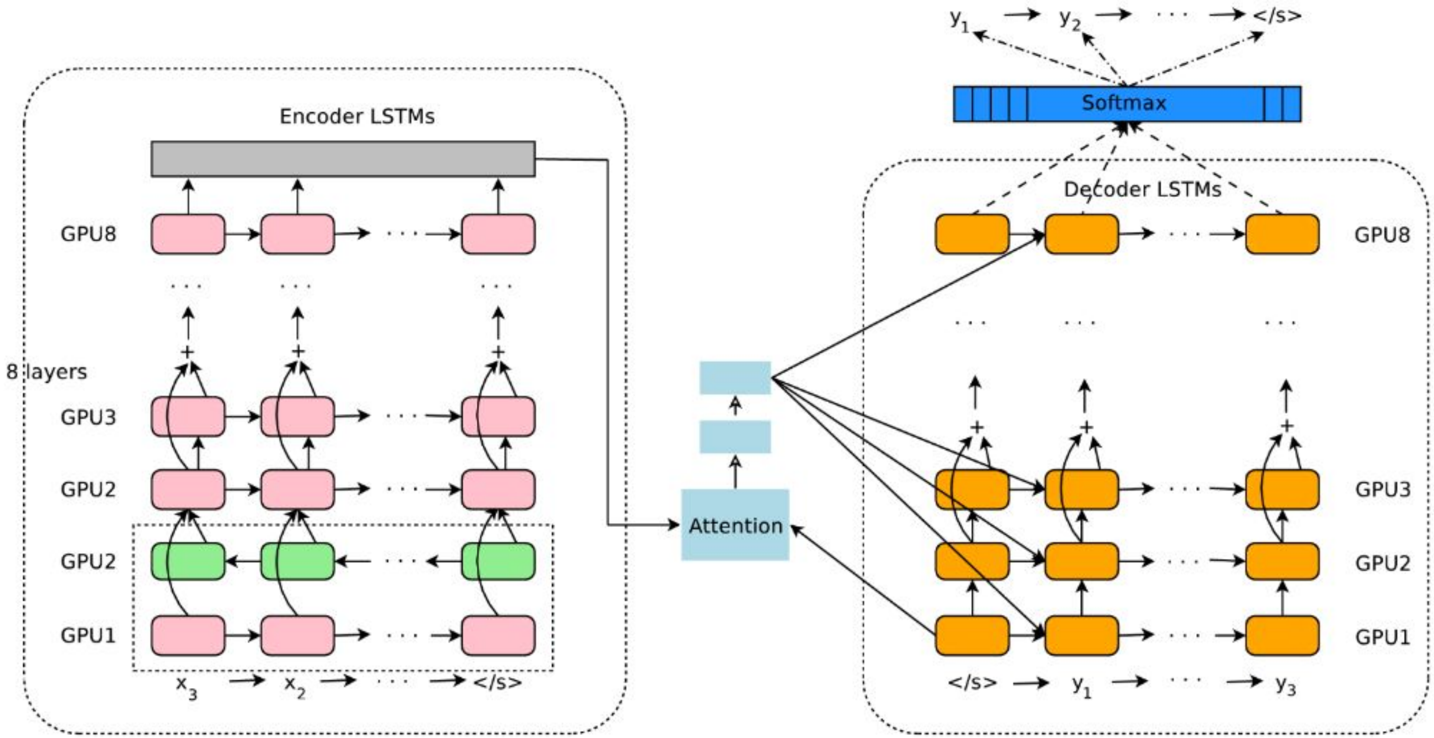
These slides are almost the exact copy of ML-MIPT course. Special thanks to ML-MIPT team.

Attention is All You Need

Based on:
https://github.com/girafe-ai/ml-mipt/blob/master/week1_04_Transformer/week04_Transformer.pdf
http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf
https://jalammar.github.io/illustrated-transformer/
https://github.com/yandexdataschool/nlp_course

Deep Encoder-Decoder Models (GNMT)

Wu et al. 2016



The Transformer

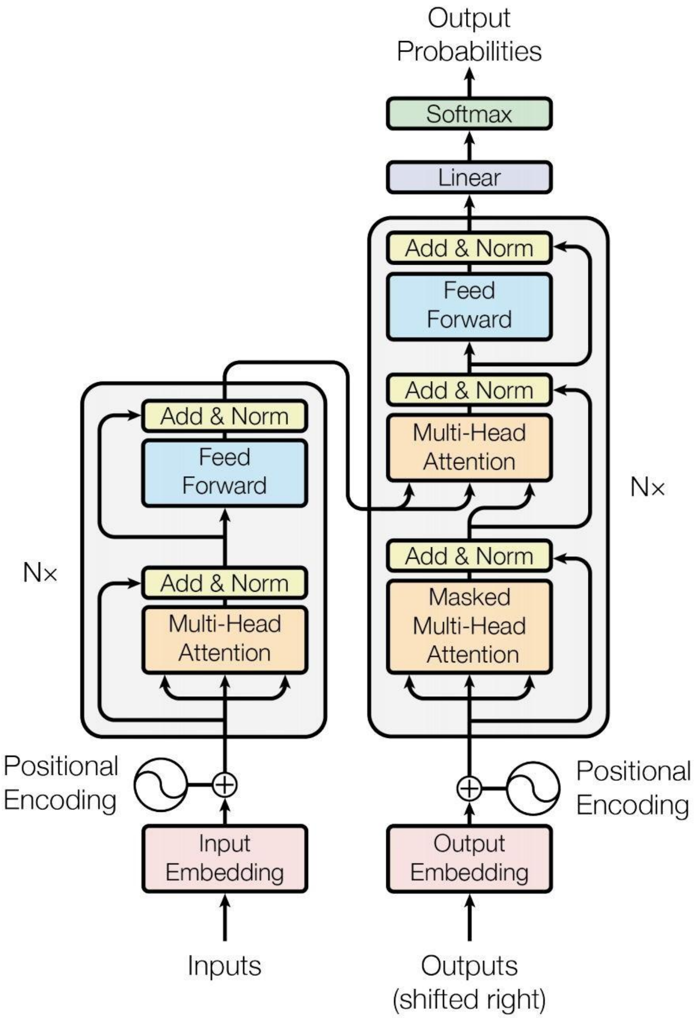


Image source: [Attention Is All You Need](), Neural Information Processing Systems 2017

The Transformer

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

The Transformer

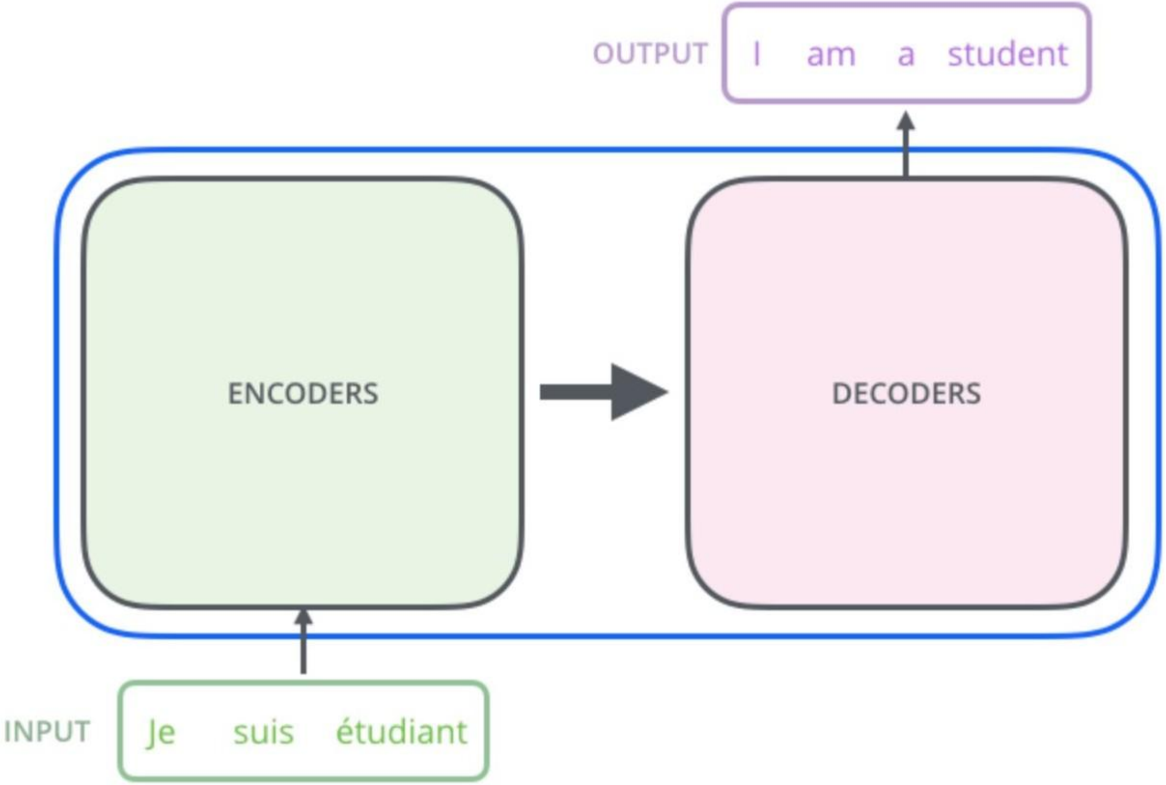


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

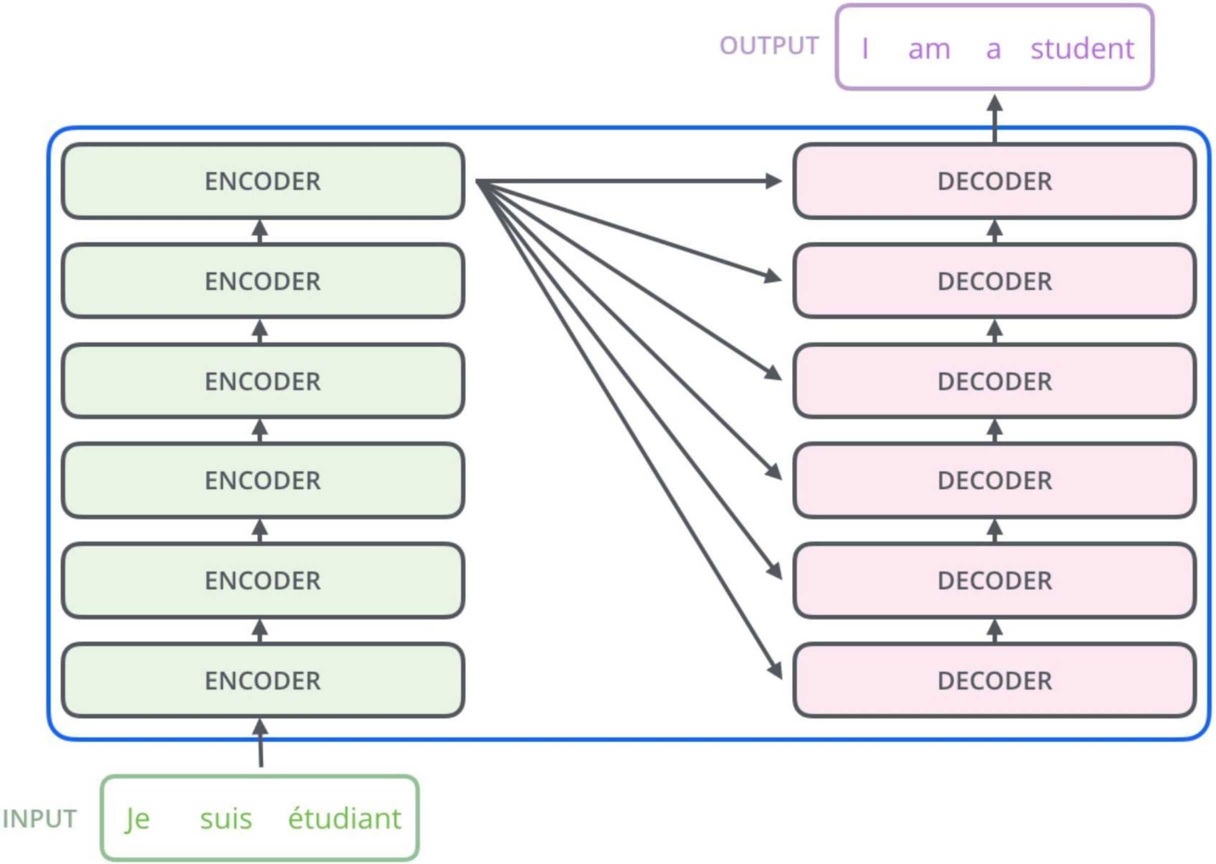
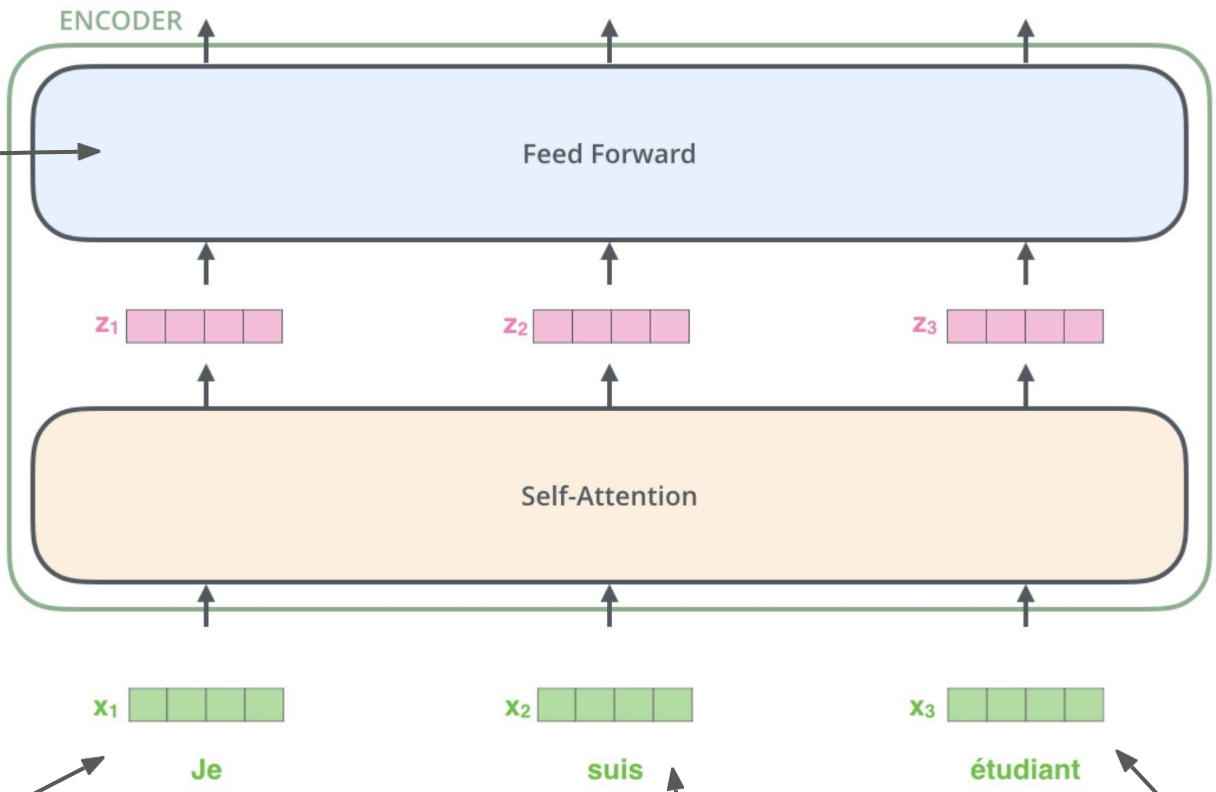


Image source: https://jalammar.github.io/illustrated-transformer/

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 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

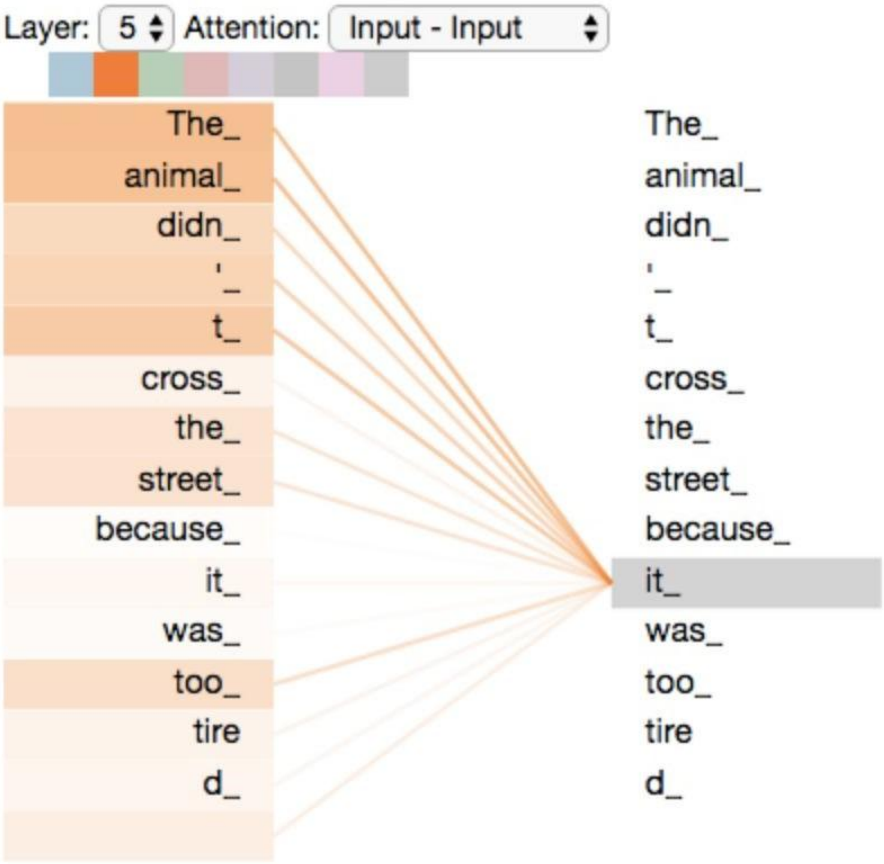
Self-Attention at a High Level

”`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

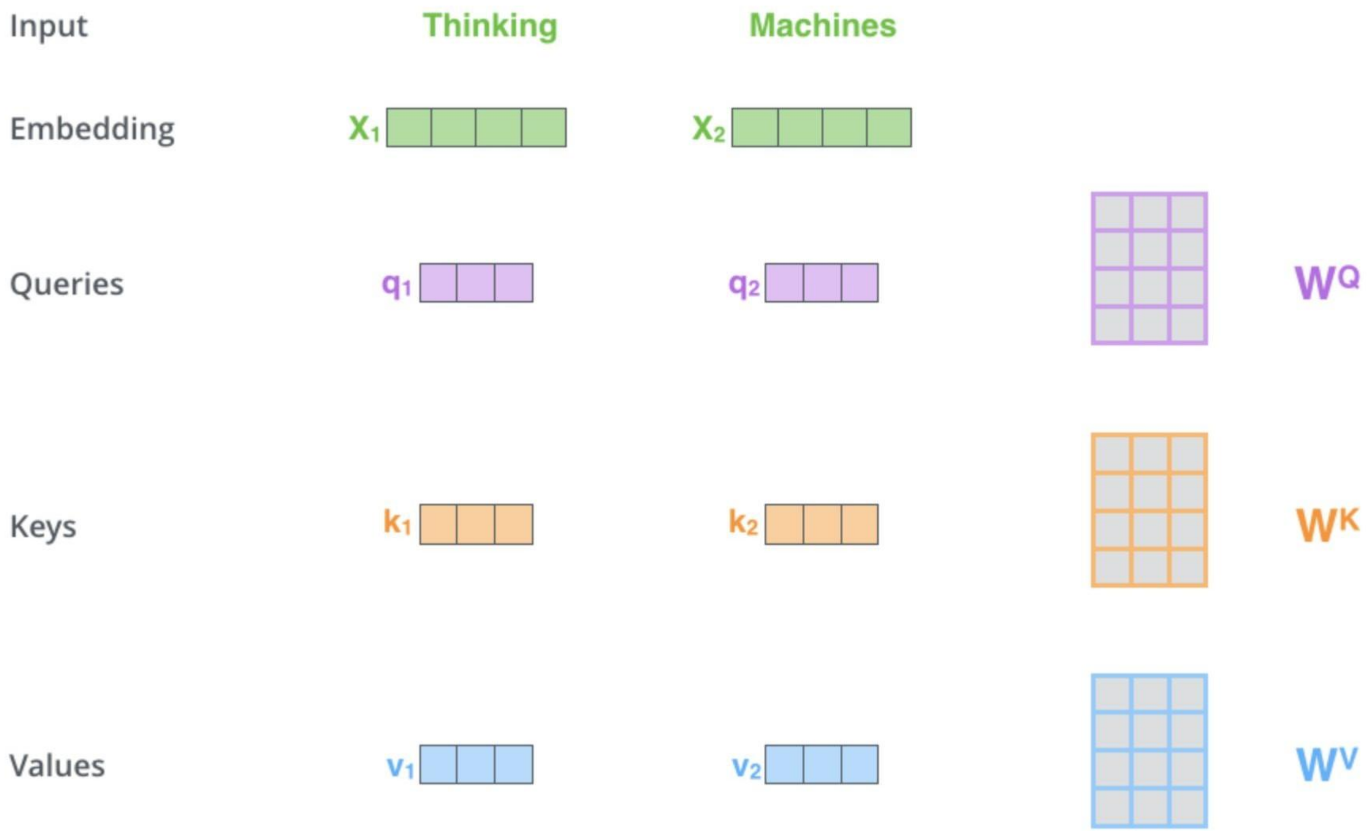


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 1:

create 3 vectors

(**query**, **key**, **value**)

from each of the encoder's input vectors

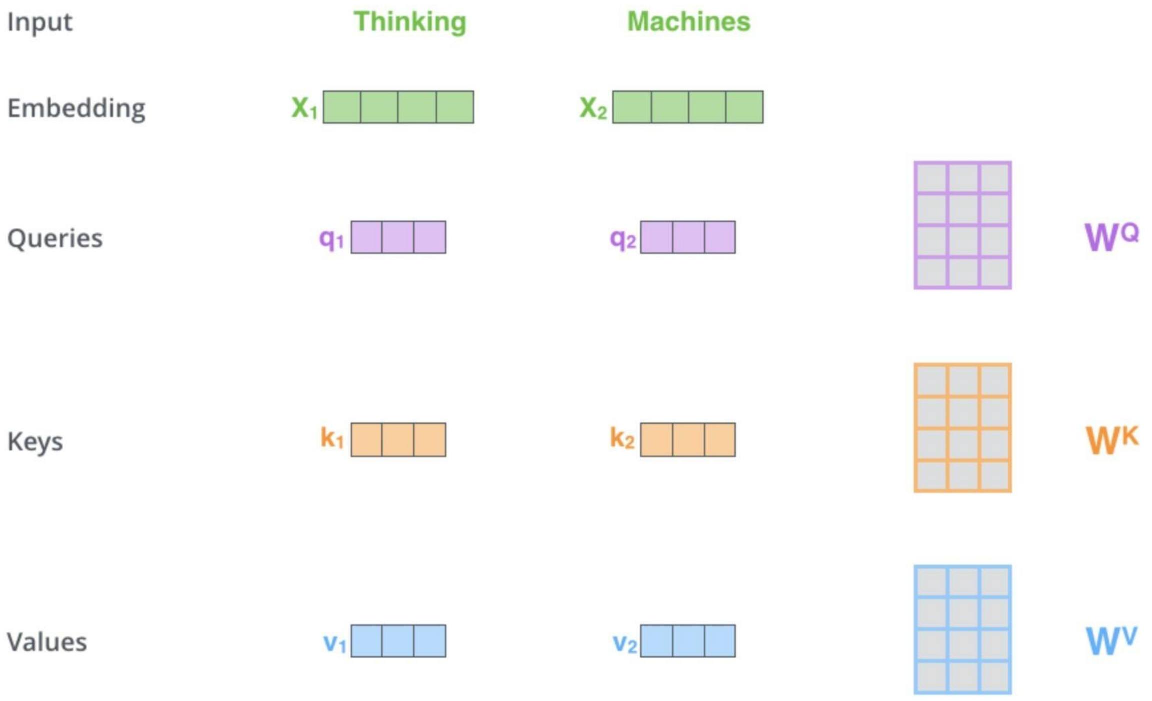


Image source: https://jalammar.github.io/illustrated-transformer/

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)

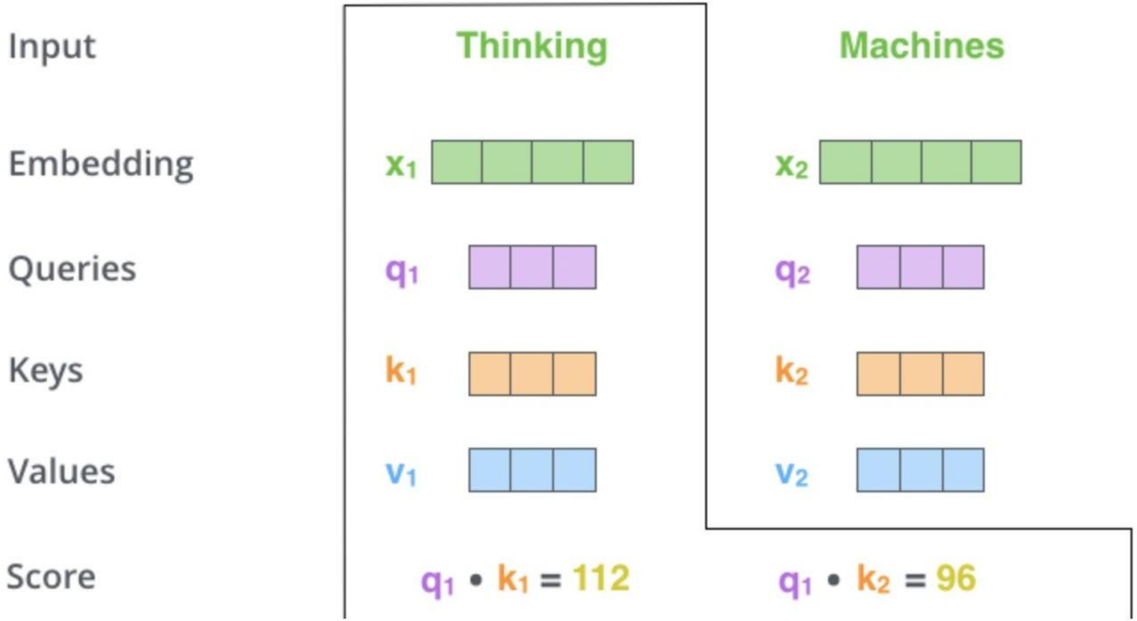


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 3:

divide the scores by 8

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

STEP 4:

softmax

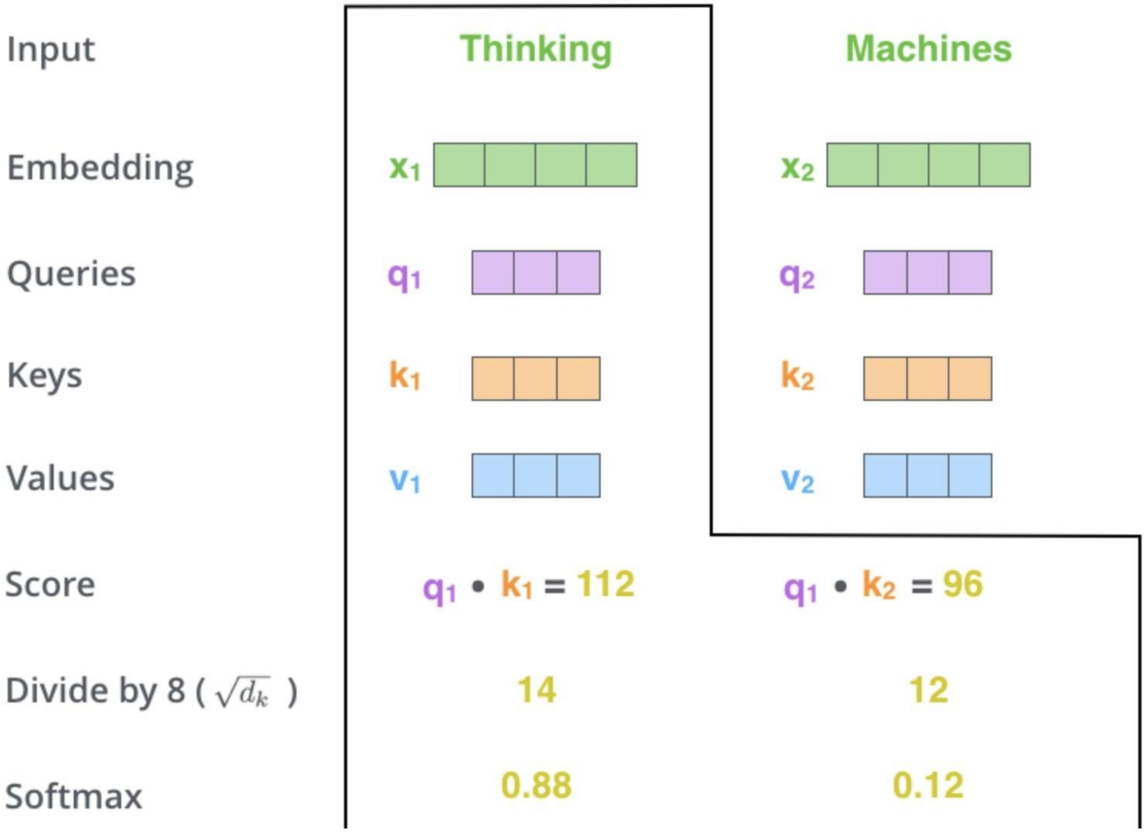


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors

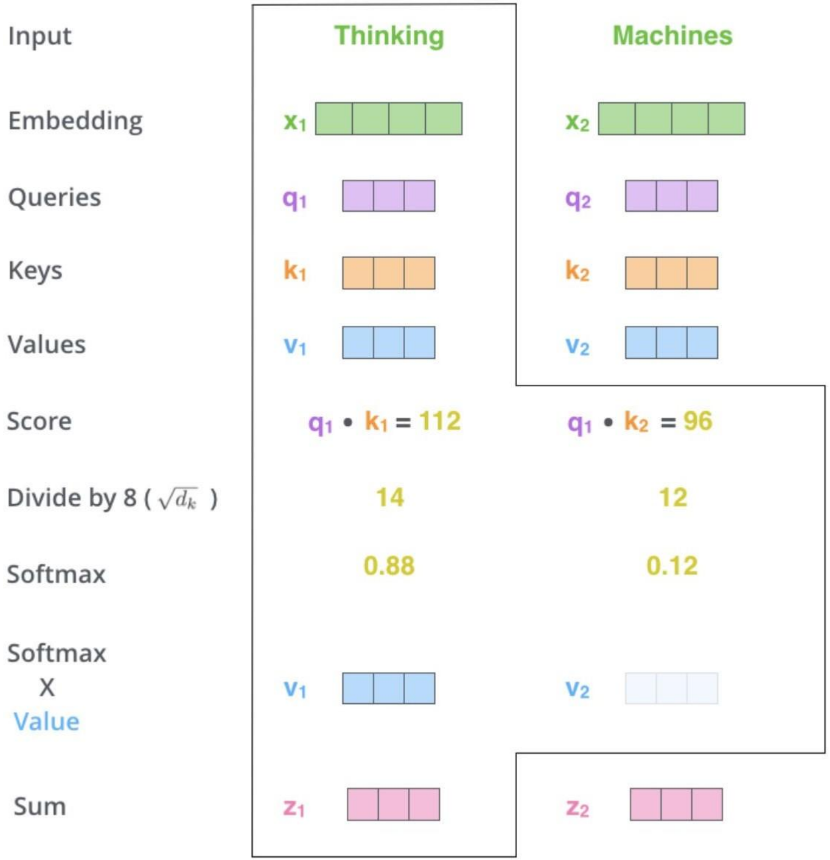
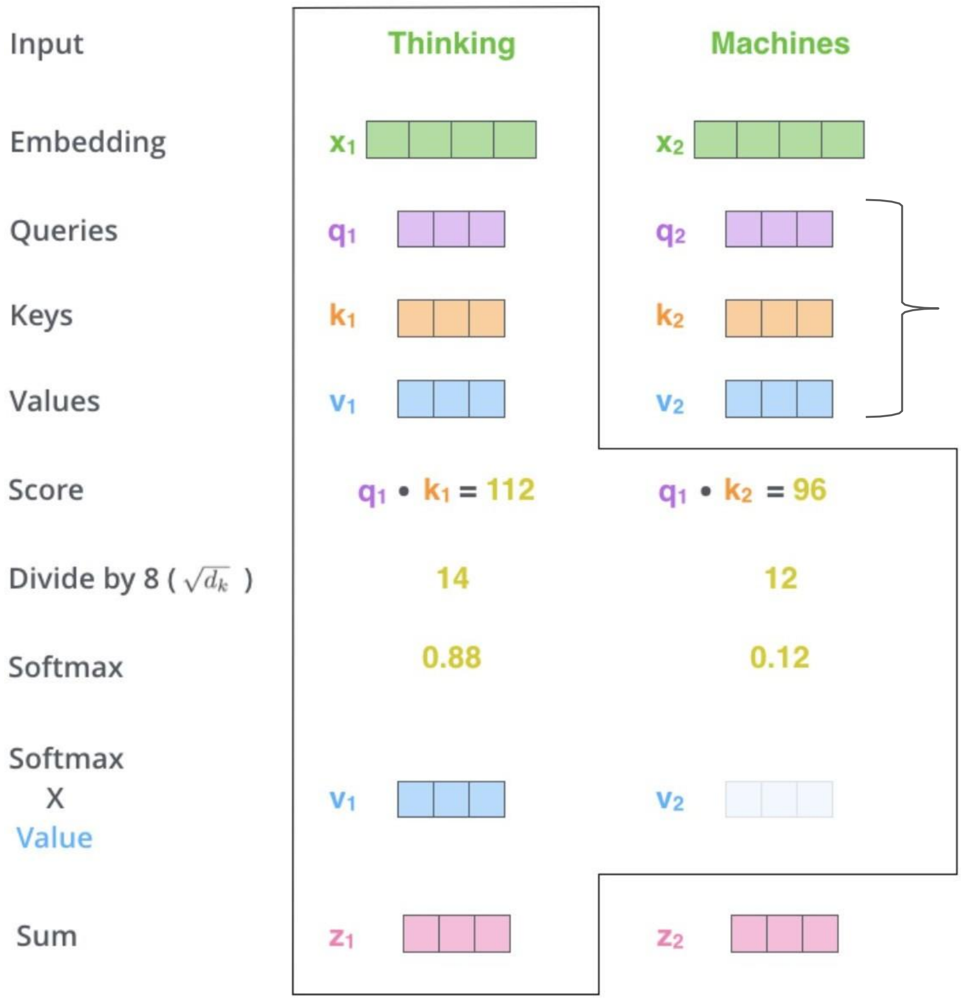


Image source: https://jalammar.github.io/illustrated-transformer/

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**)

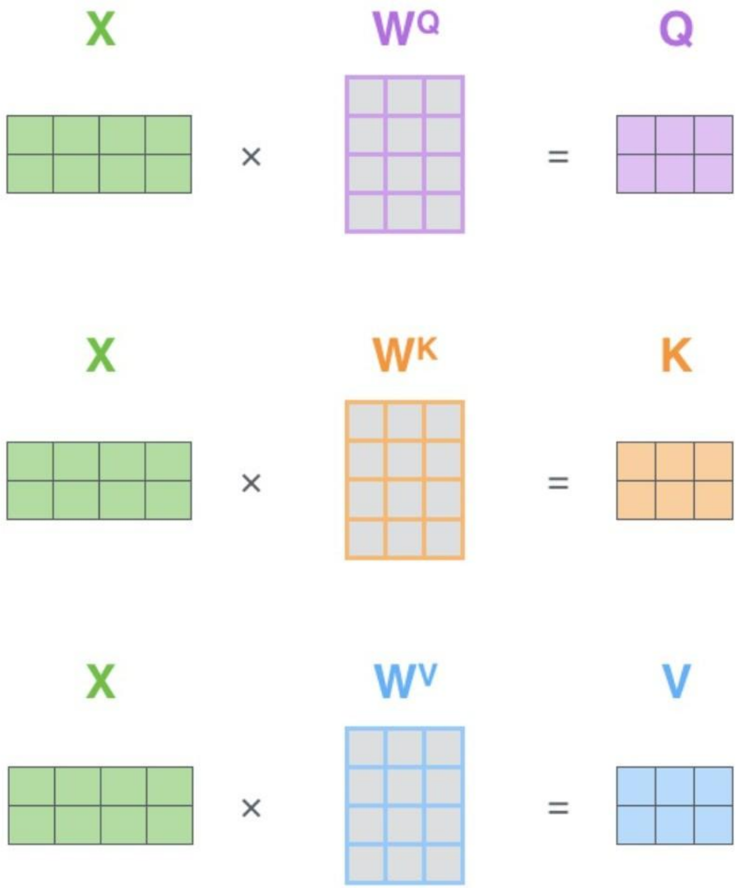


Image source: https://jalammar.github.io/illustrated-transformer/

Self-Attention: Matrix Calculation

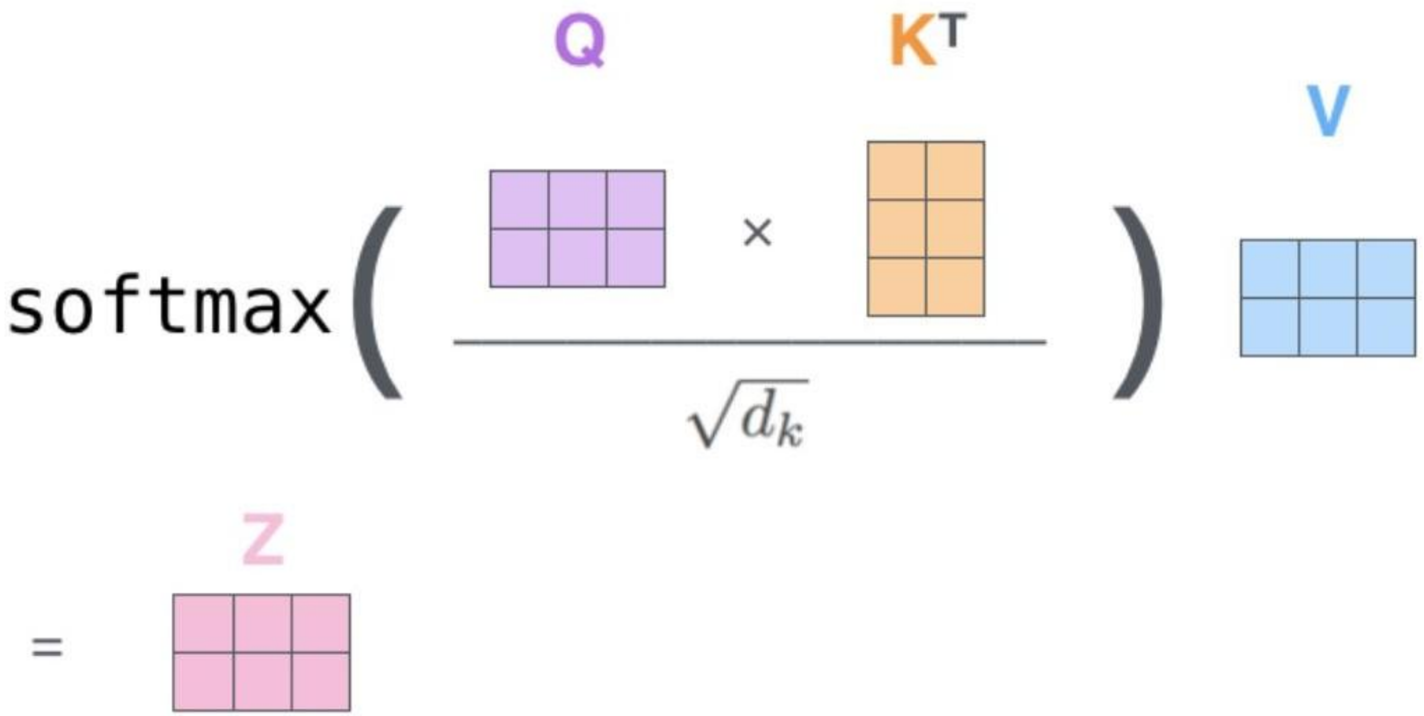


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

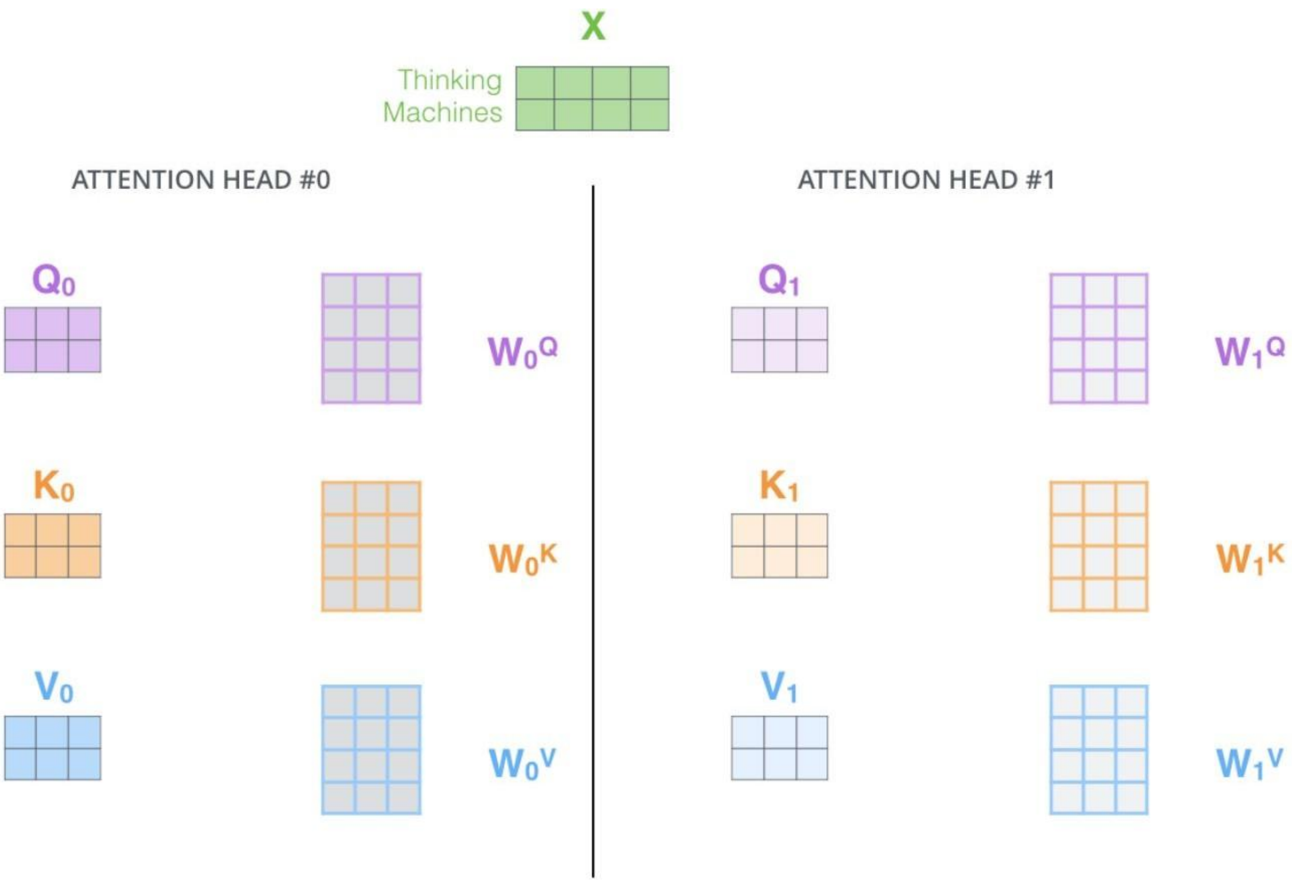


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

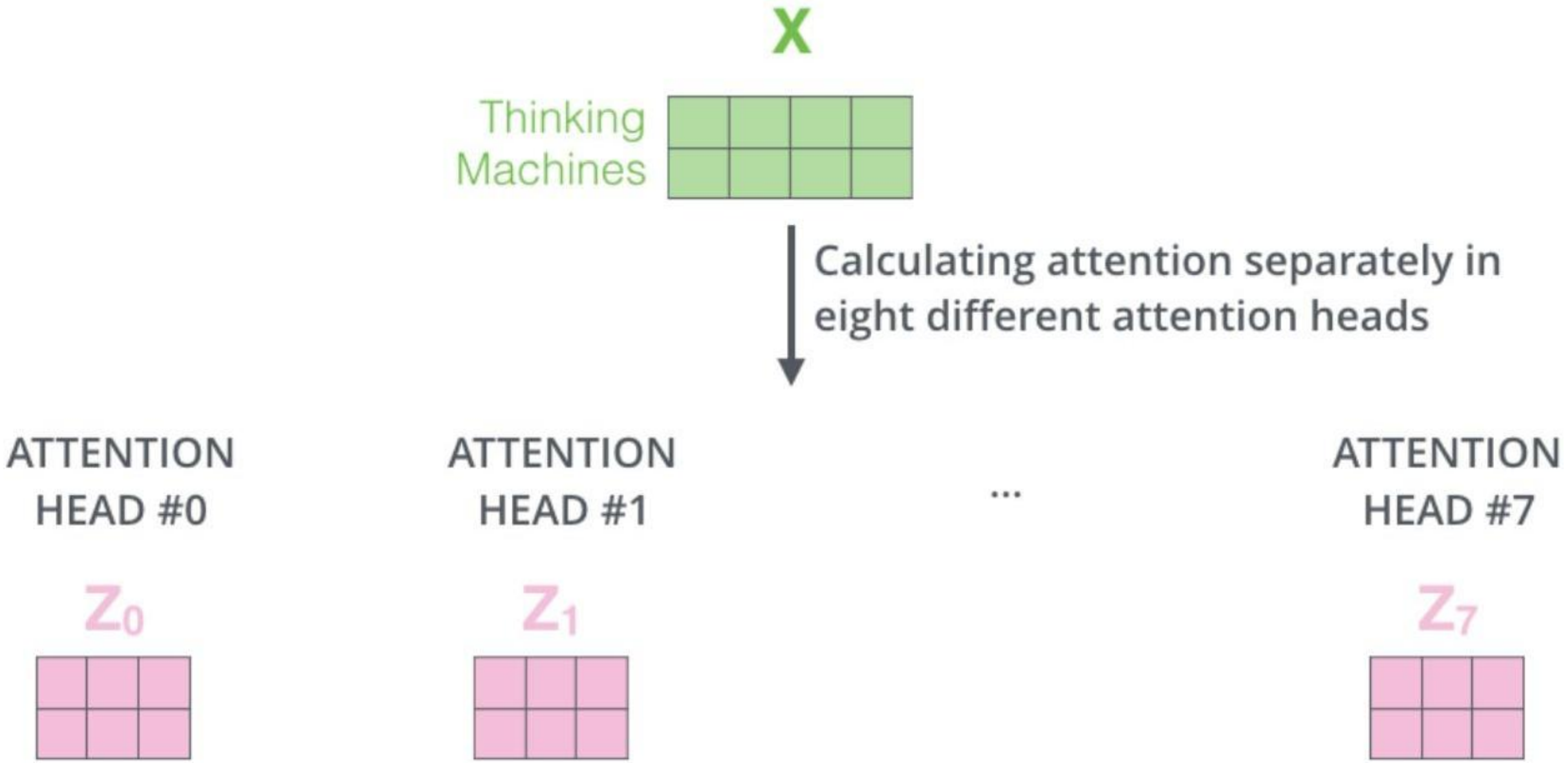


Image source: https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention

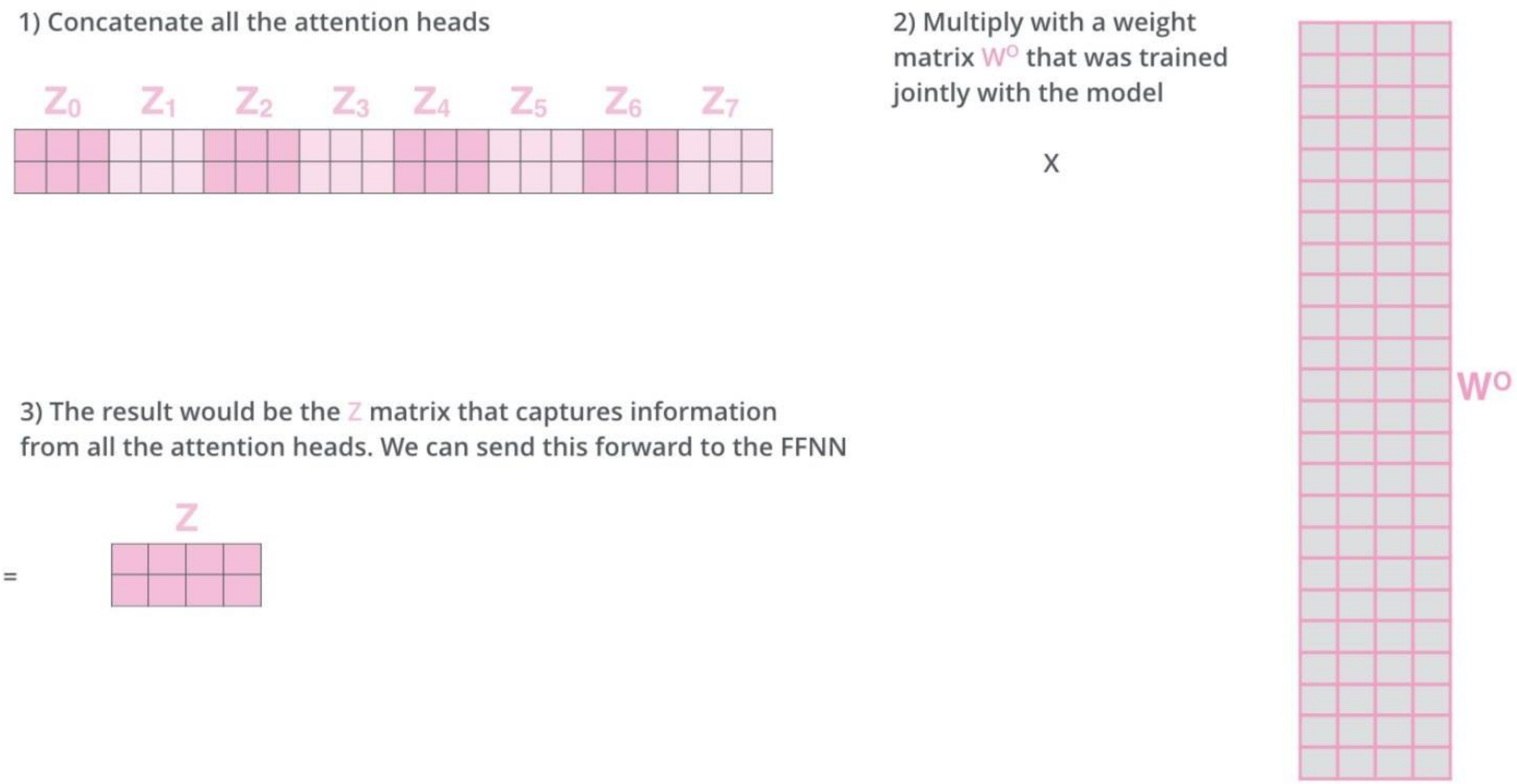


Image source: https://jalammar.github.io/illustrated-transformer/

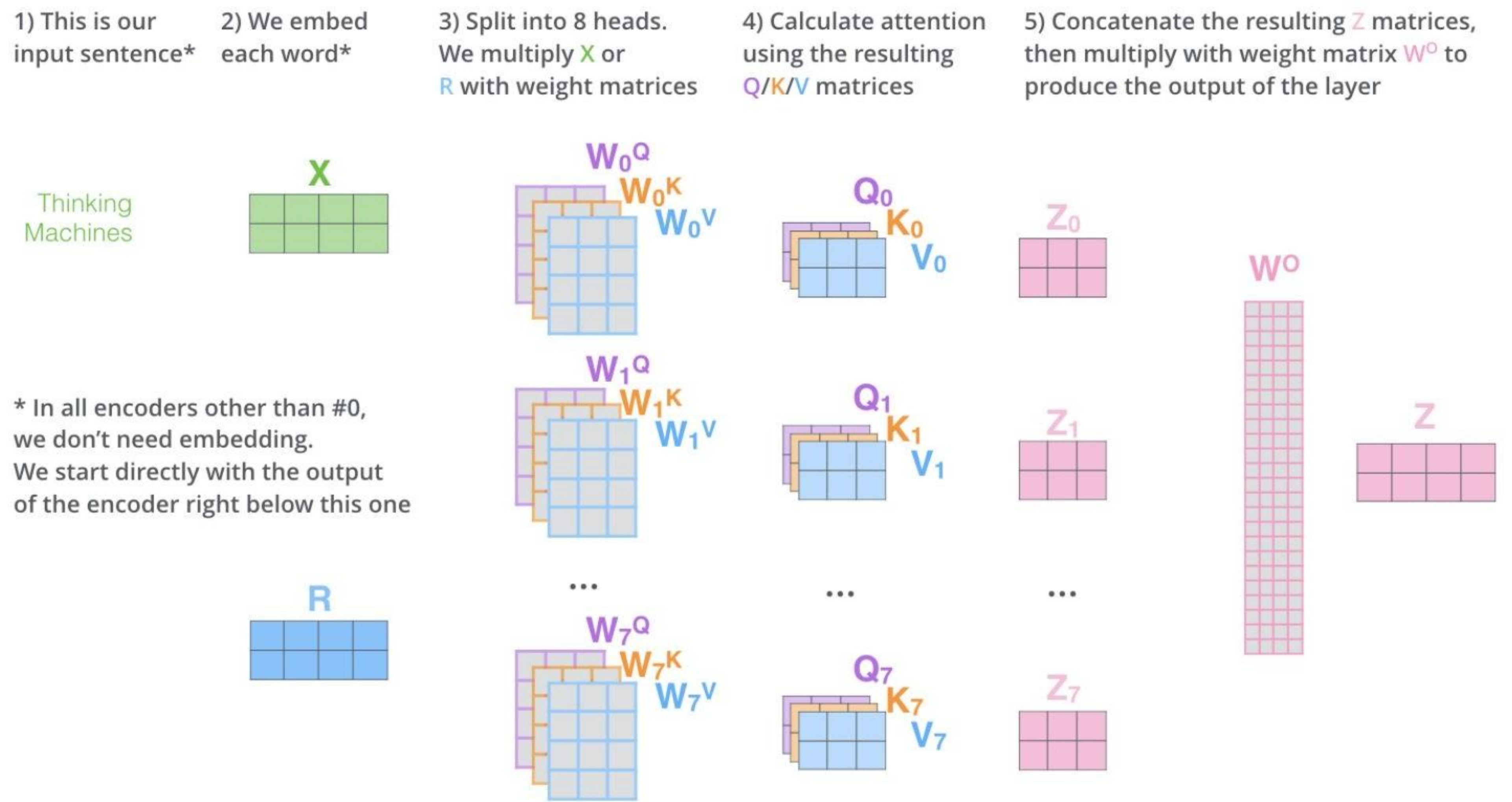
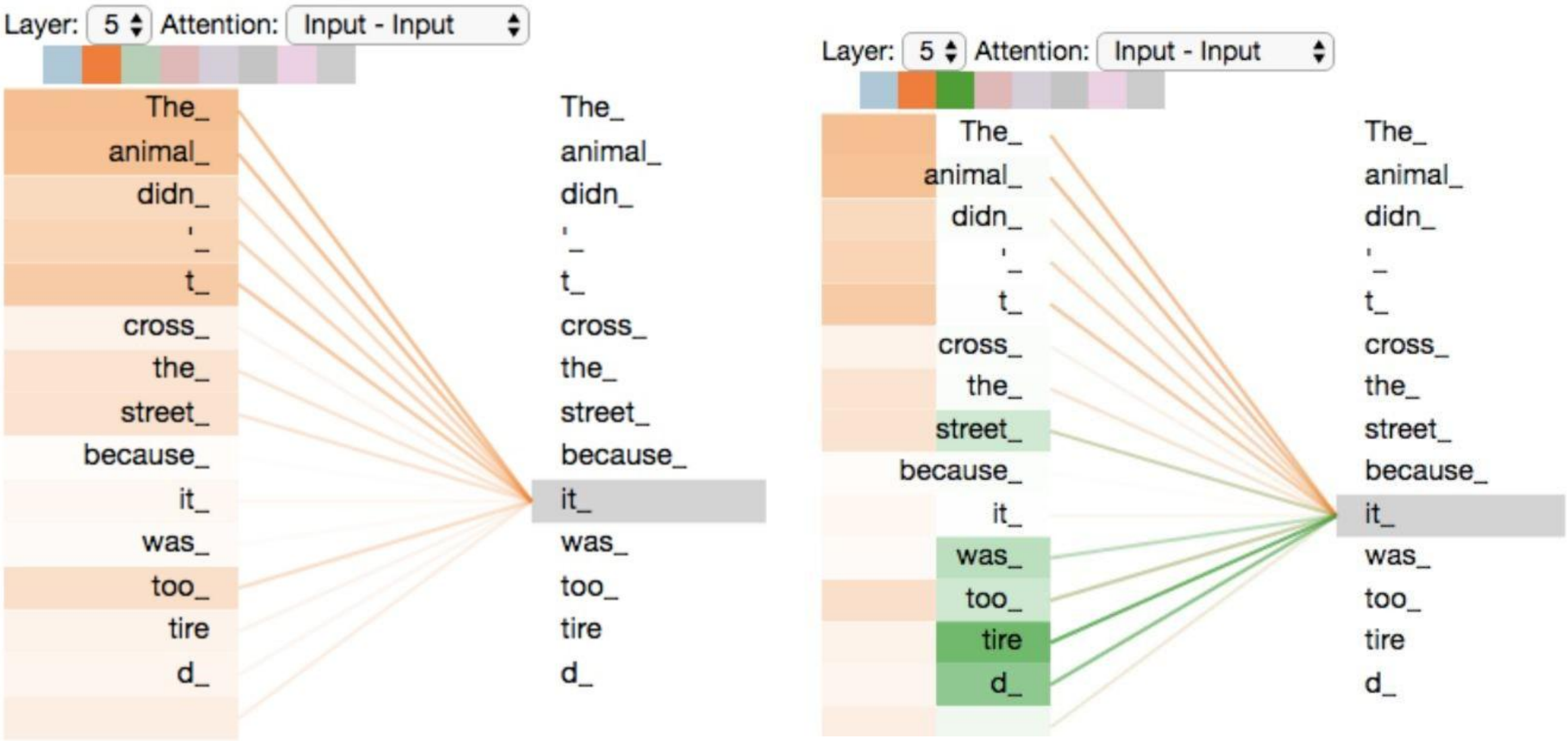
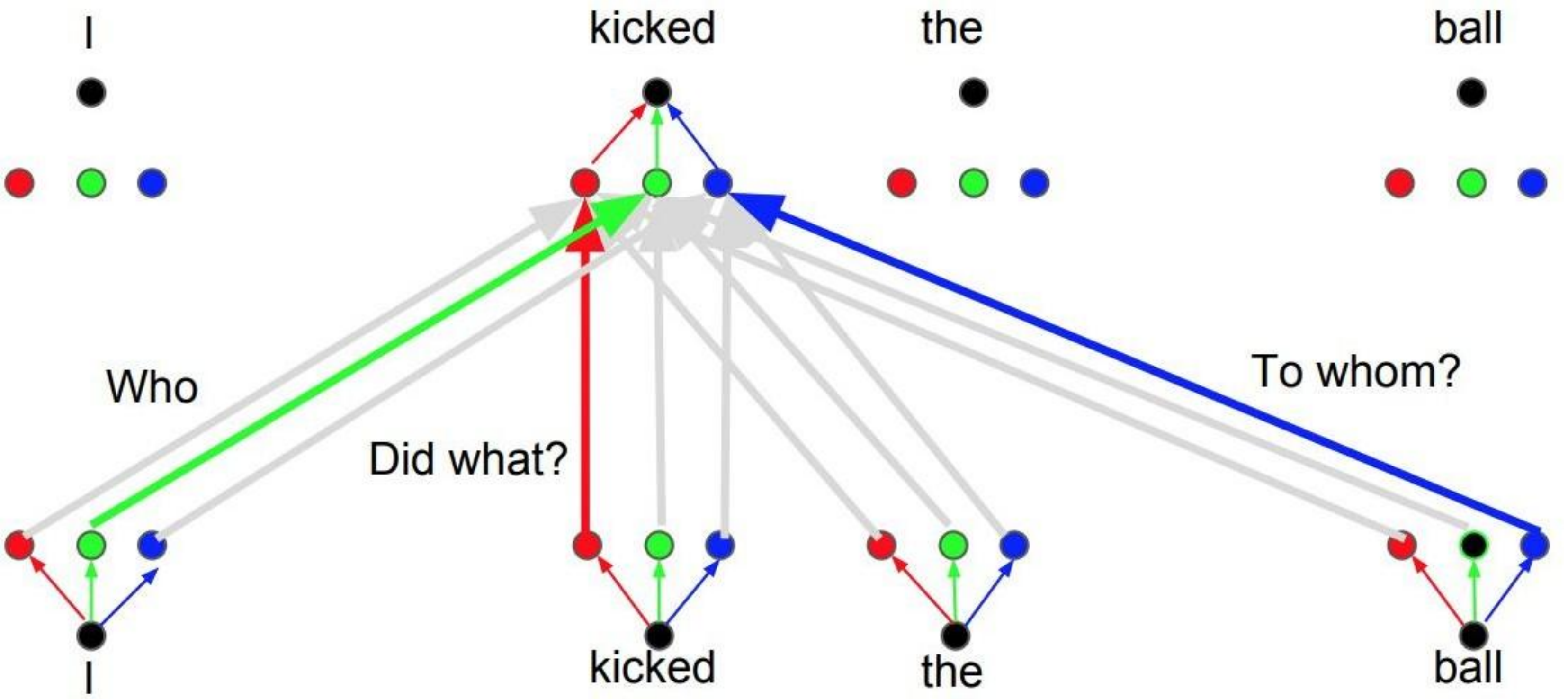


Image source: https://jalammar.github.io/illustrated-transformer/

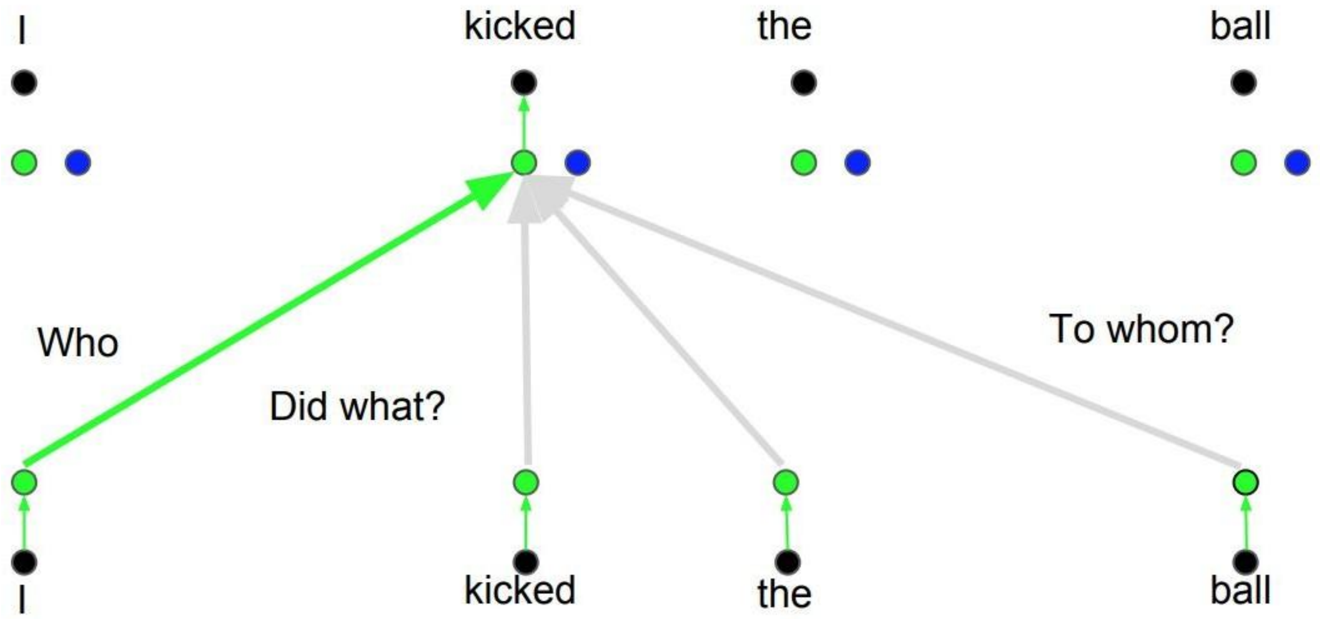
Multi-Head Attention



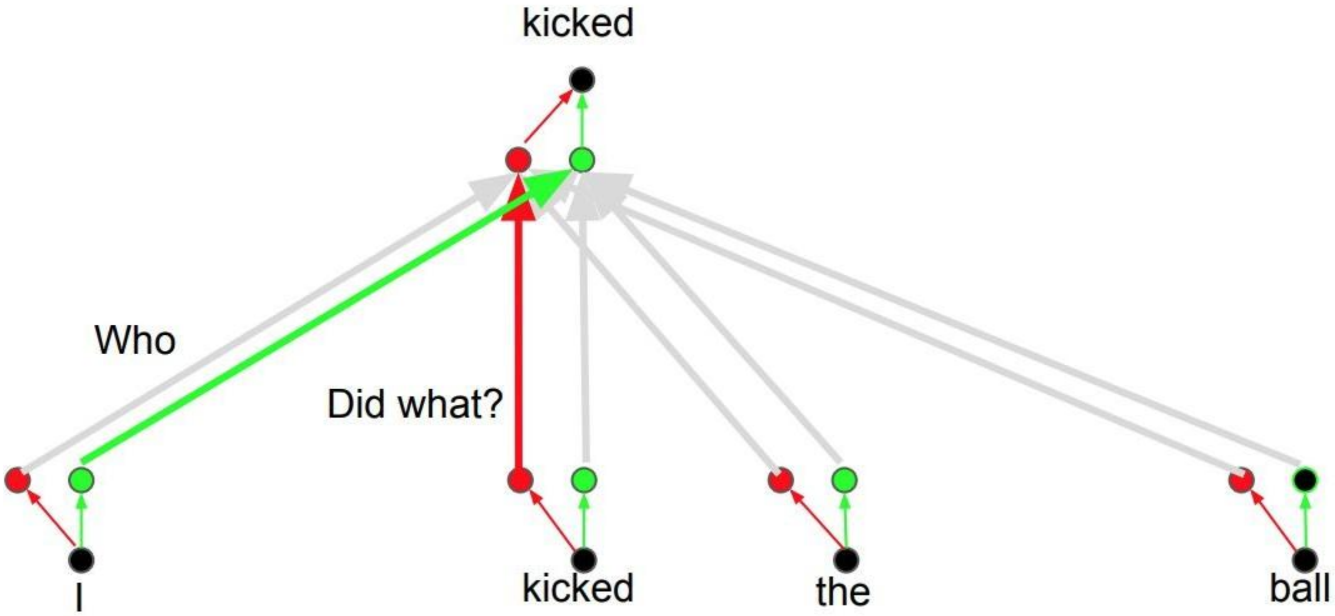
Why Multi-Head Attention?



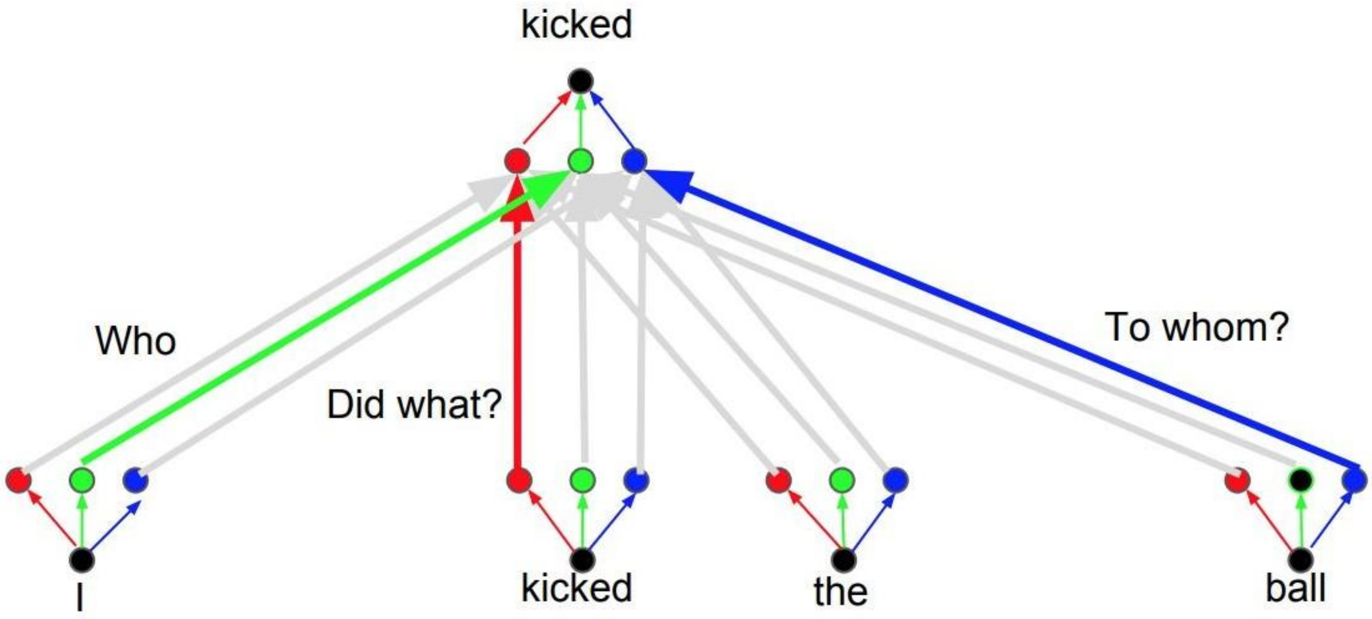
Attention head: Who



Attention head: Did What?

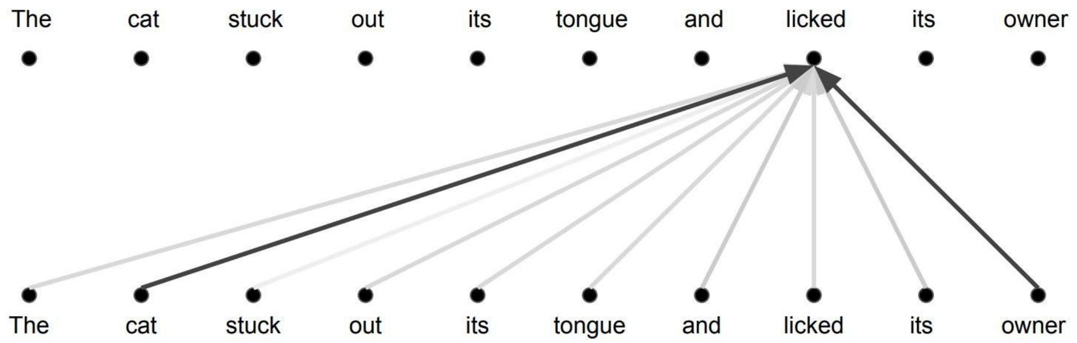


Attention head: To Whom?



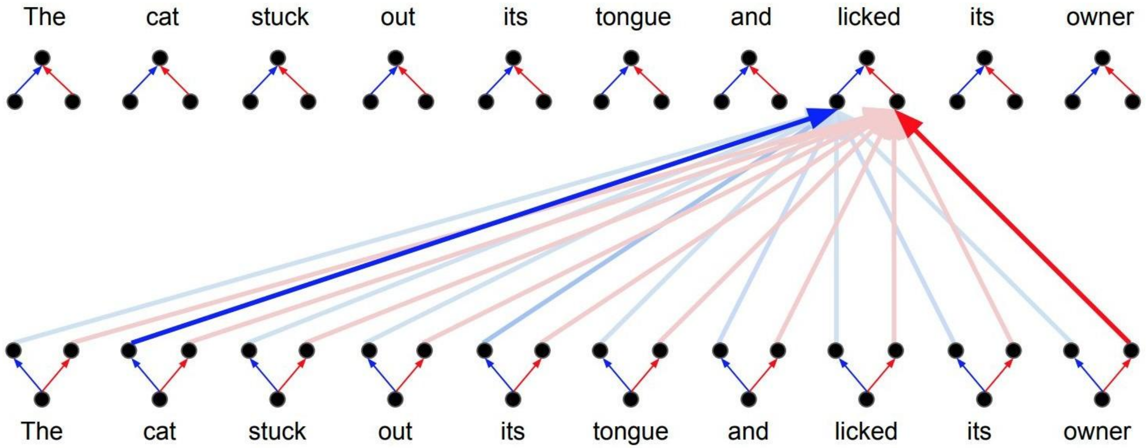
Attention vs. Multi-Head Attention

Attention: a weighted average



Multi-Head Attention:

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



Performance: WMT 2014 BLEU

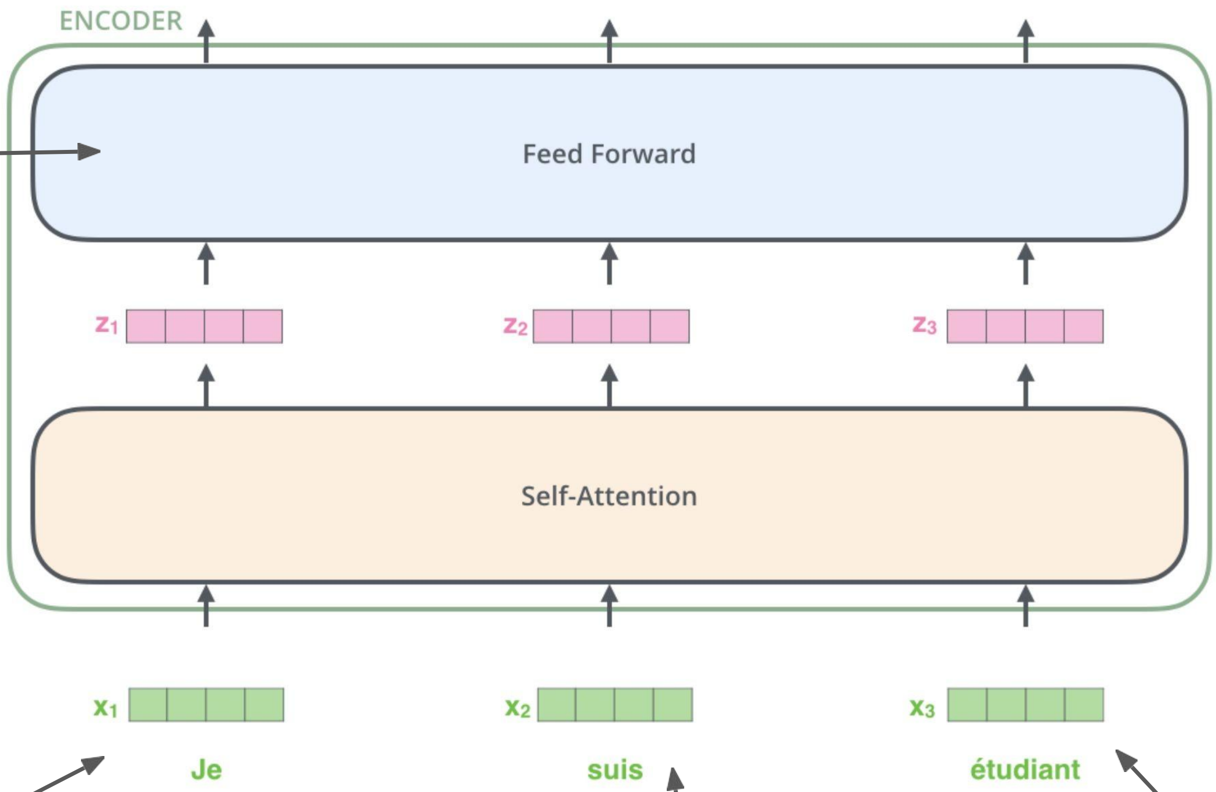
	EN-DE	EN-FR
GNMT (orig)	24.6	39.9
ConvSeq2Seq	25.2	40.5
Transformer*	**28.4**	**41.8**

*Transformer models trained $\boxed{>3\text{x faster}}$ than the others.

Positional Encoding

The Encoder Side

Can be parallelized



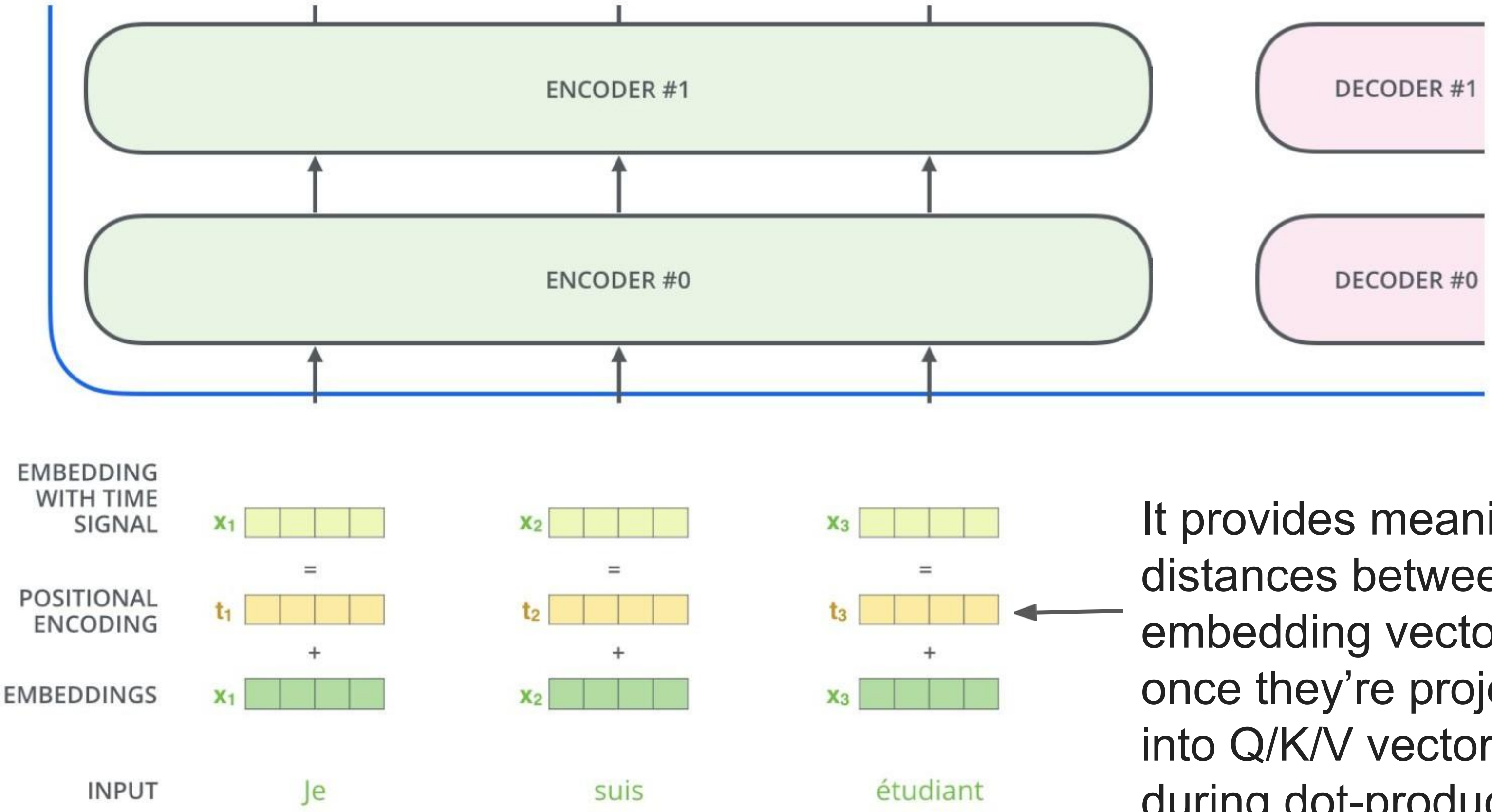
the word in each position flows through its own path in the encoder

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?

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

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

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

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

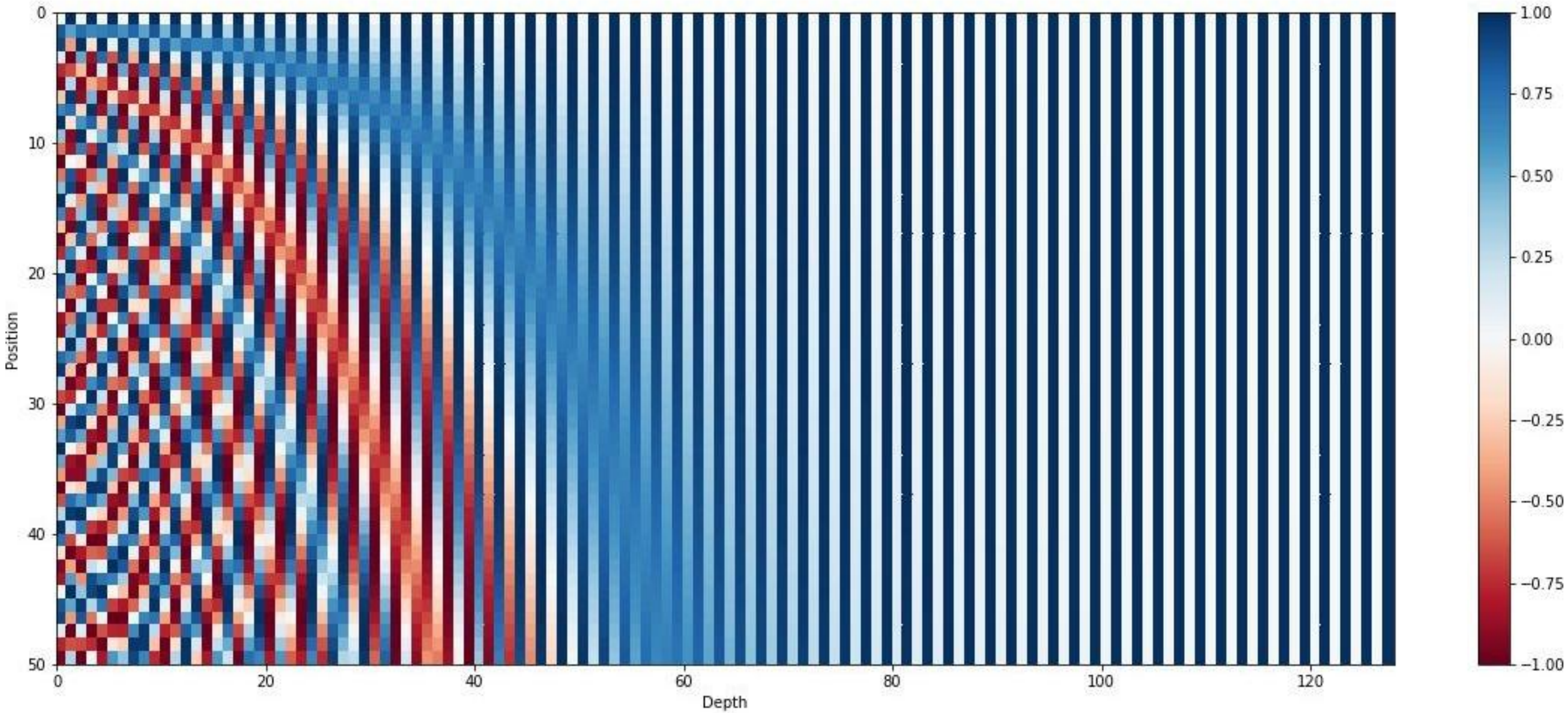


Image source: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

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/

Layer Normalization

The Transformer: recap

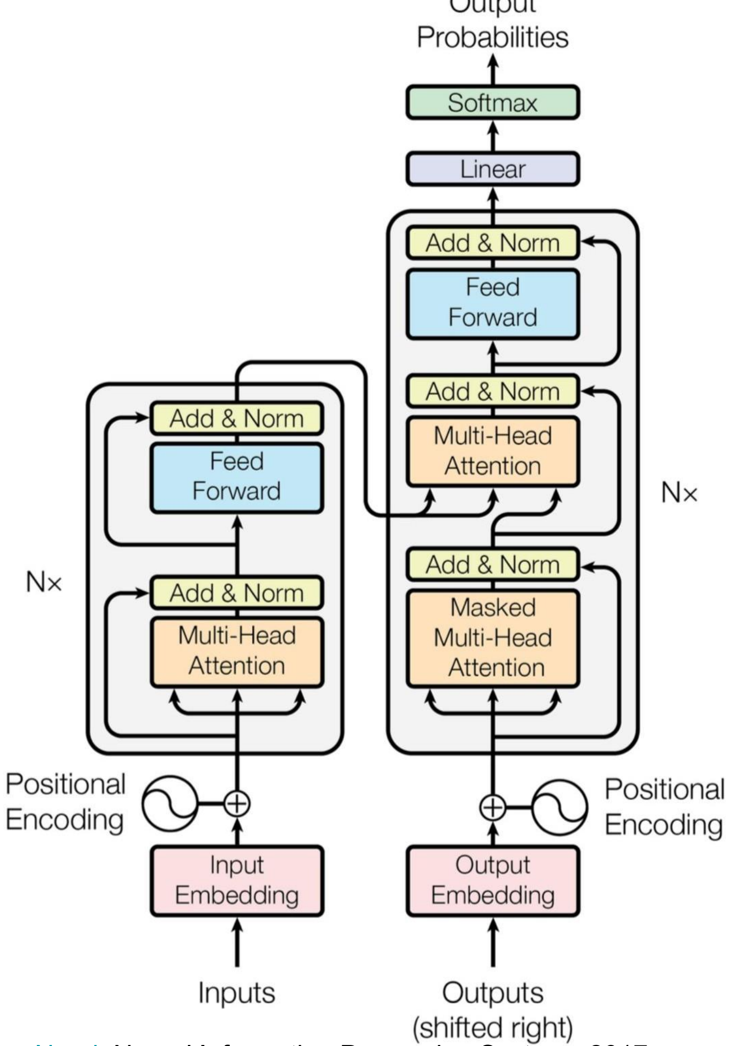


Image source: [Attention Is All You Need](), Neural Information Processing Systems 2017

Layer Normalization

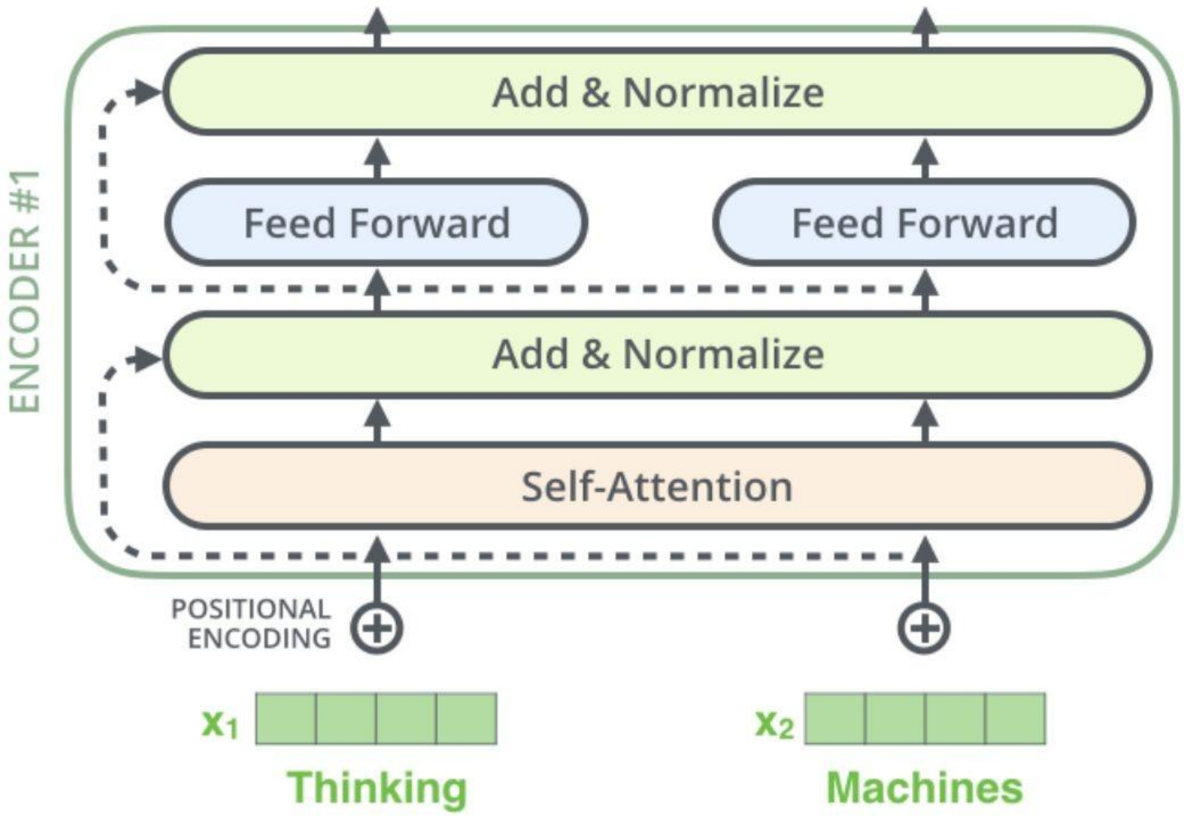
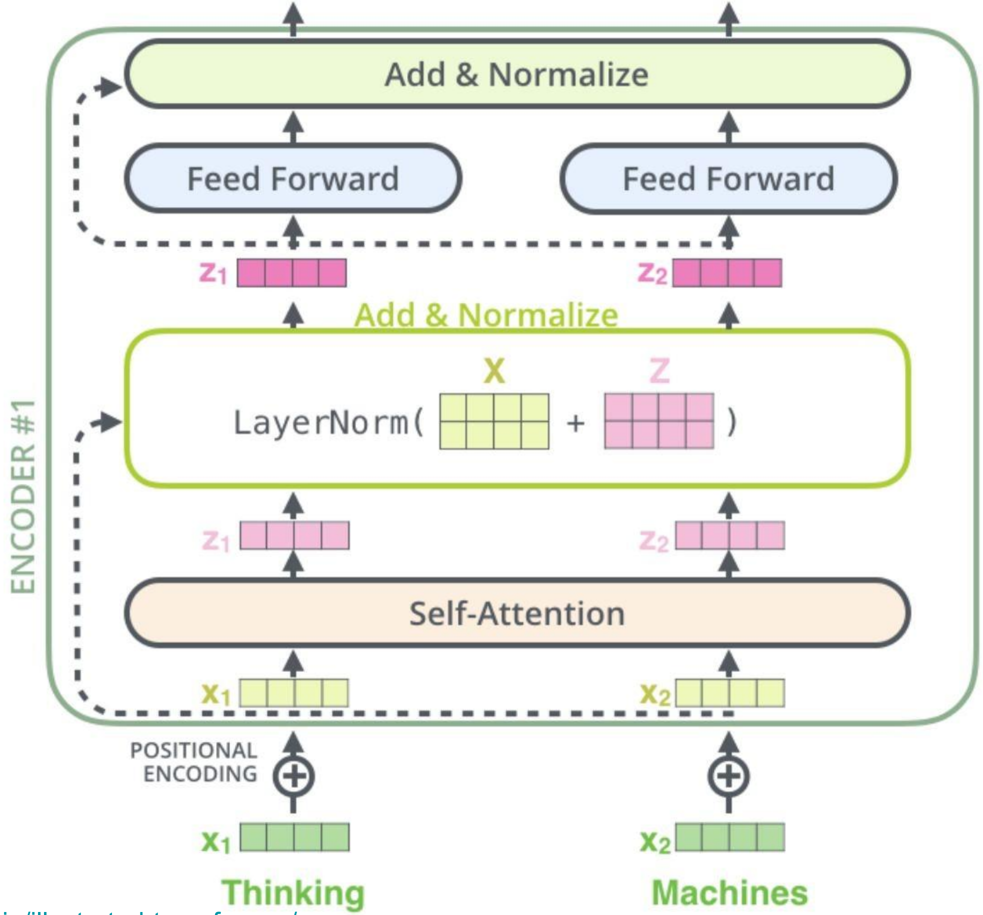


Image source: https://jalammar.github.io/illustrated-transformer/

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

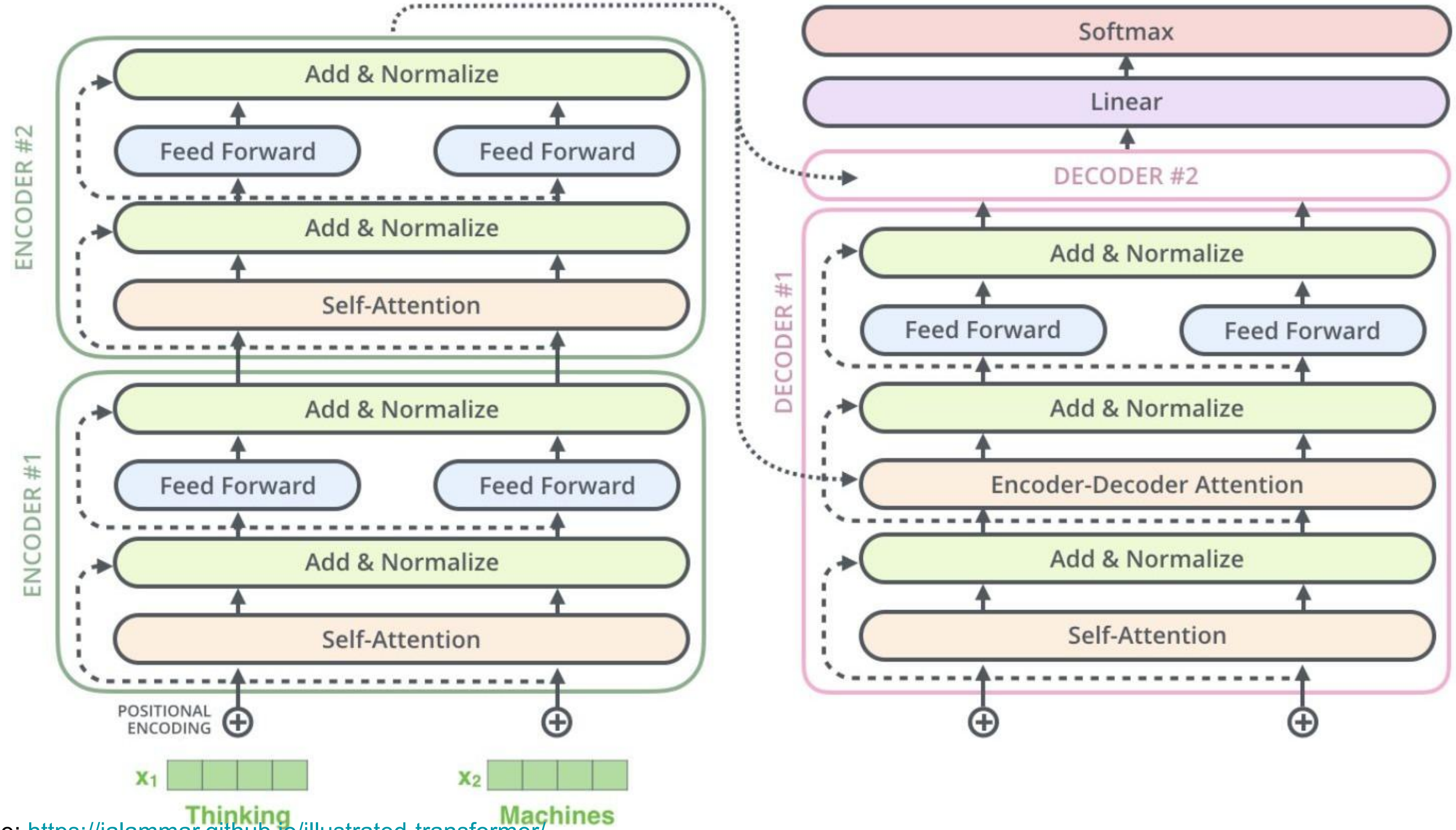
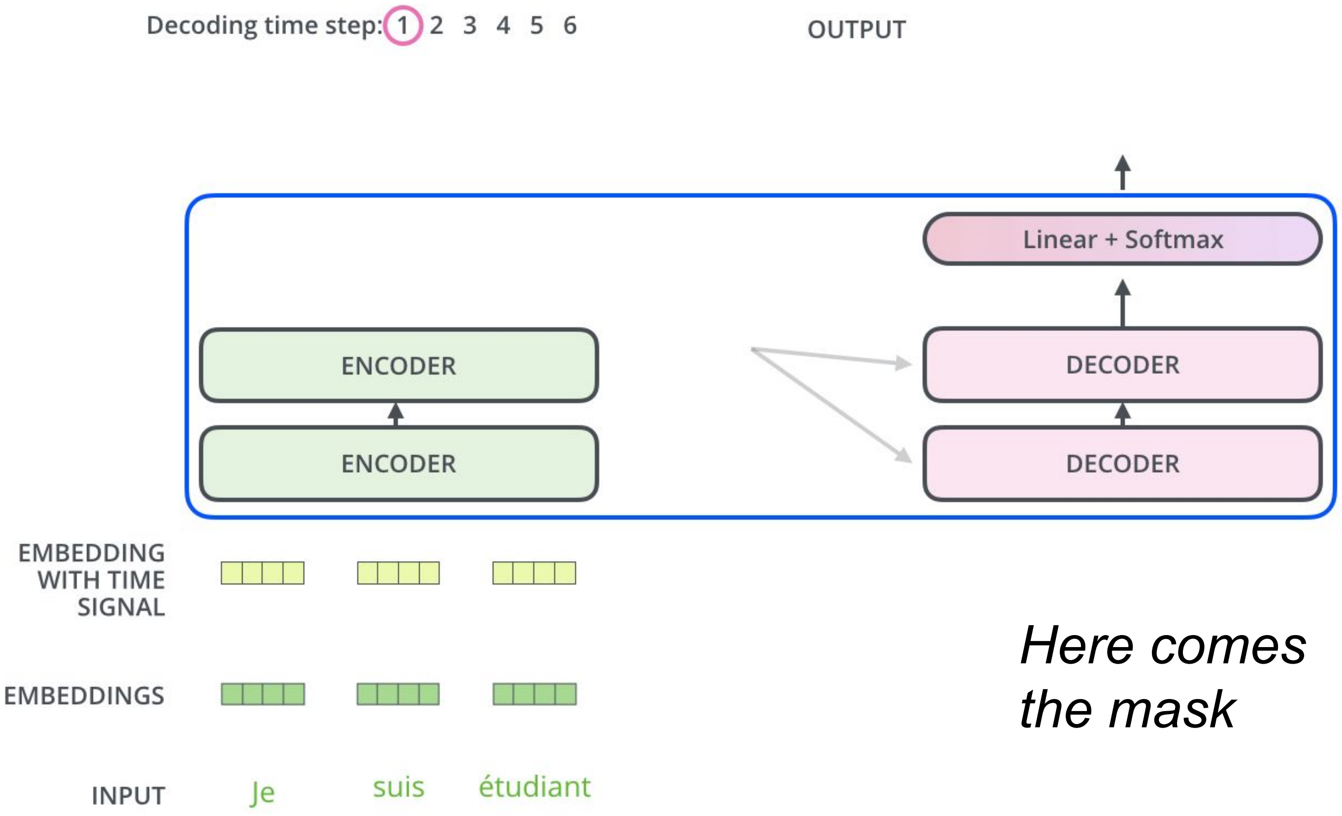


Image source: https://jalammar.github.io/illustrated-transformer/

The Decoder

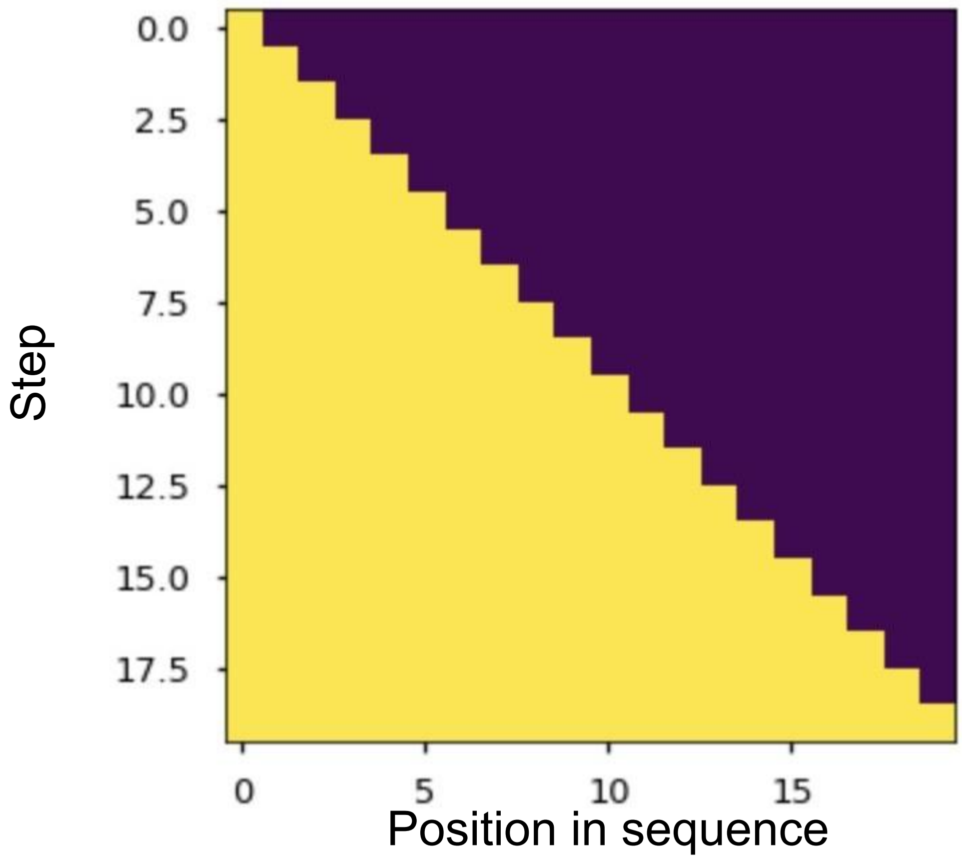
The Decoder Side



Here comes the mask

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

The masked decoder input



The Decoder Side

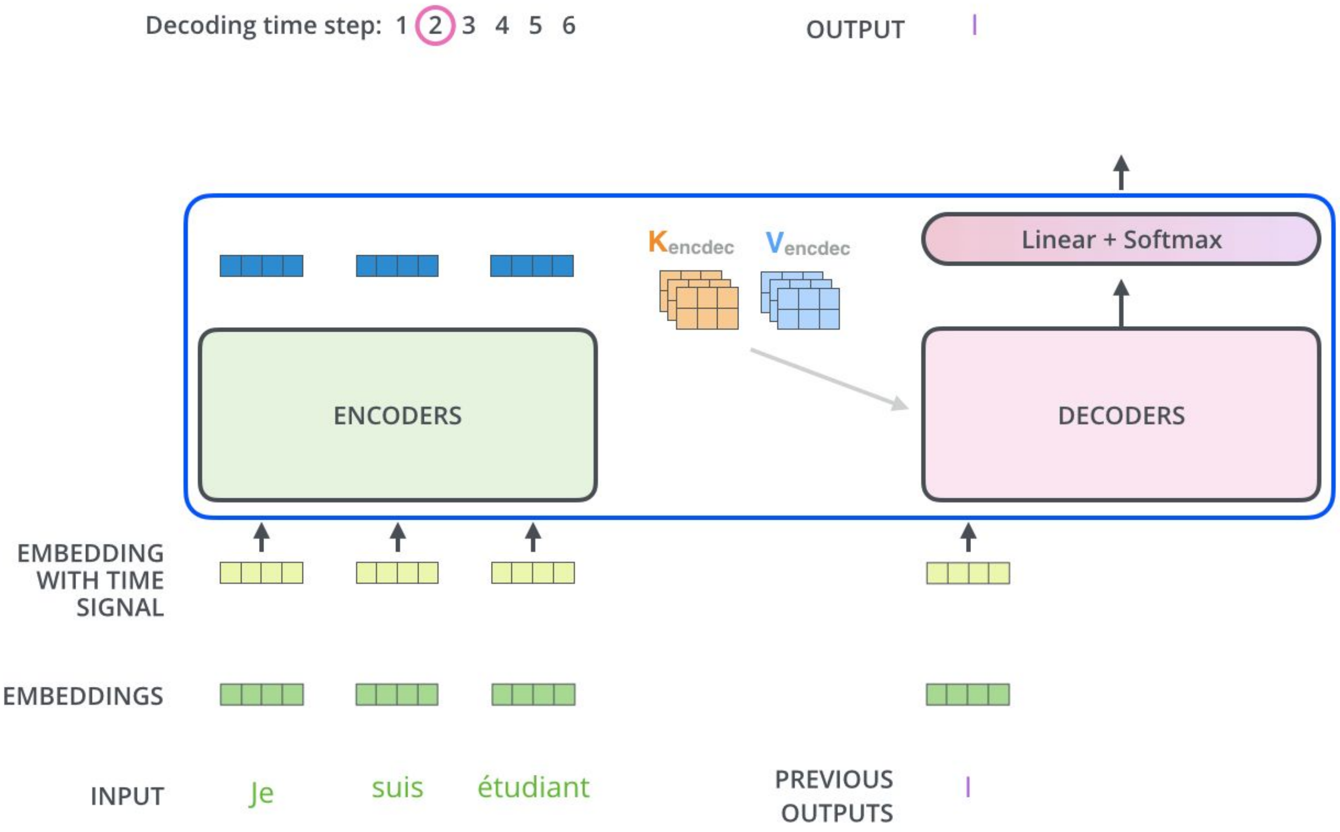


Image source: https://jalammar.github.io/illustrated-transformer/

Final Linear and Softmax Layer

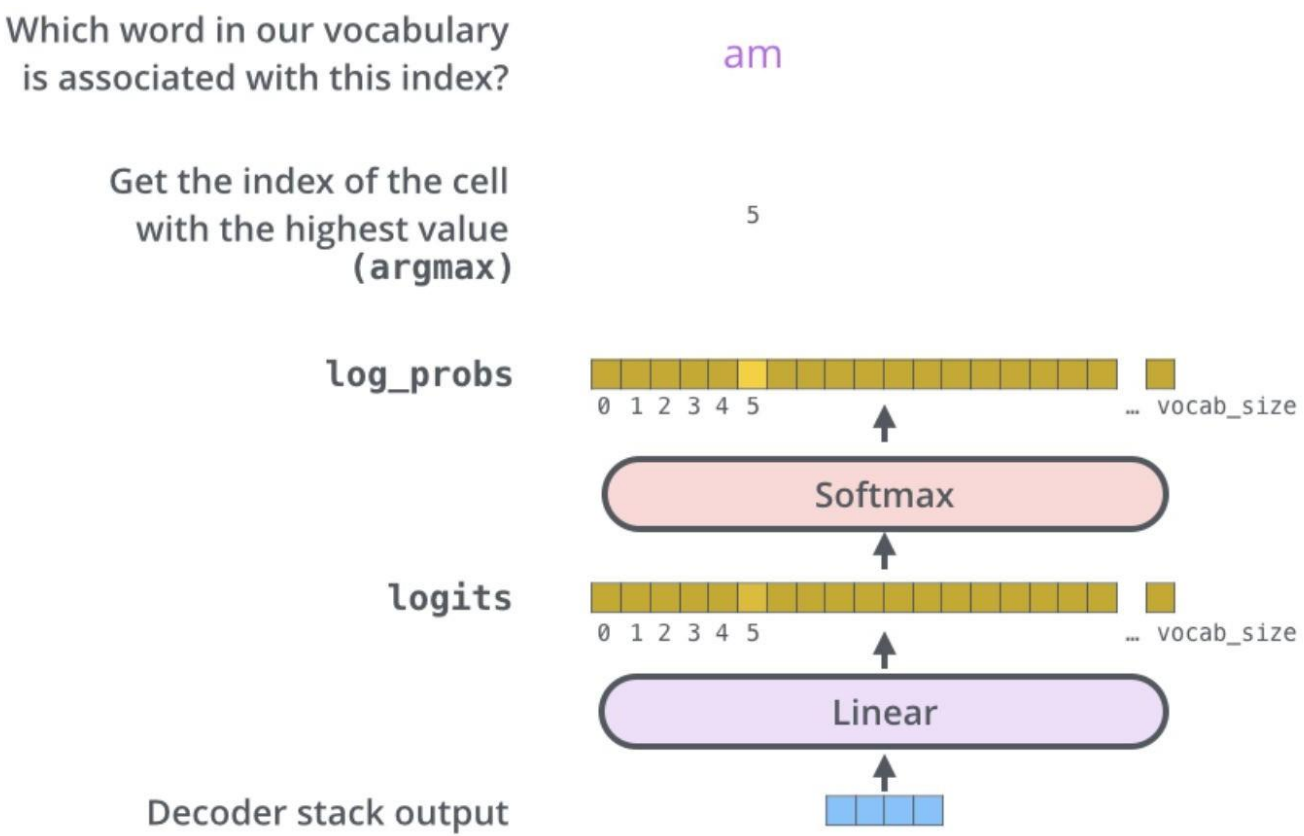


Image source: https://jalammar.github.io/illustrated-transformer/

The Transformer

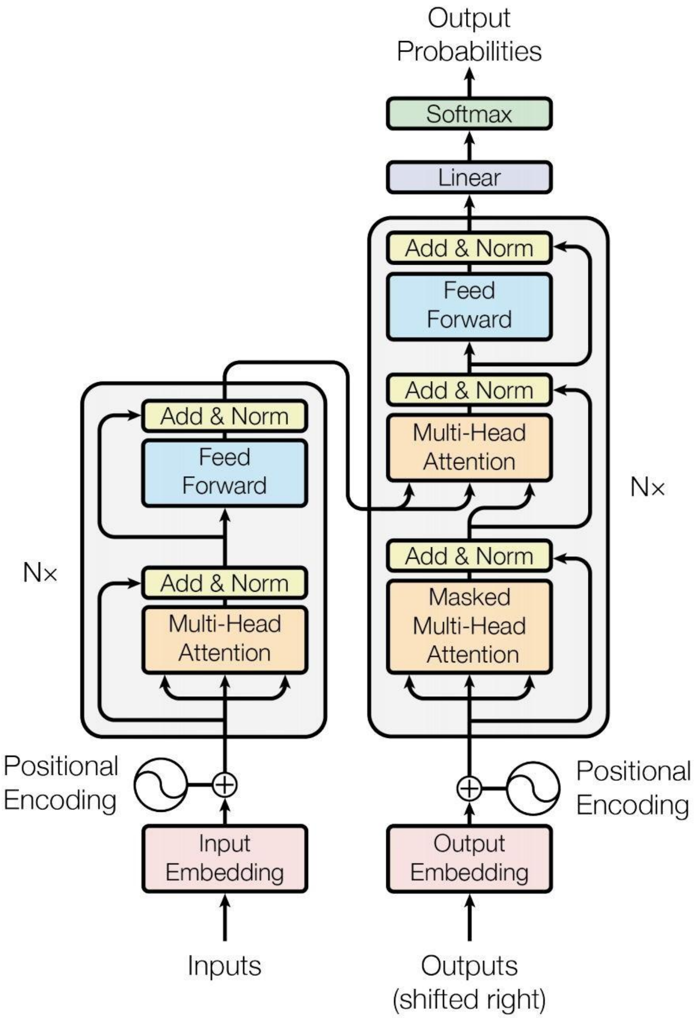


Image source: [Attention Is All You Need](), Neural Information Processing Systems 2017

Training The Transformer

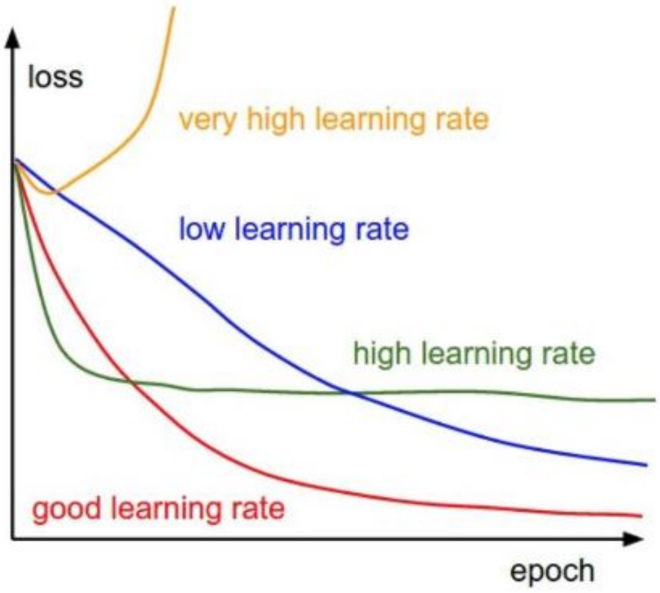
➢ Consider Adam optimizer

$$\mu_{t+1} = \beta_1 \cdot \mu_t + (1 - \beta_1) \cdot \nabla_\theta L$$

$$v_{t+1} = \beta_2 \cdot v_t + (1 - \beta_2) \cdot ||\nabla_\theta L||^2$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v_t} + \epsilon} \mu_t$$

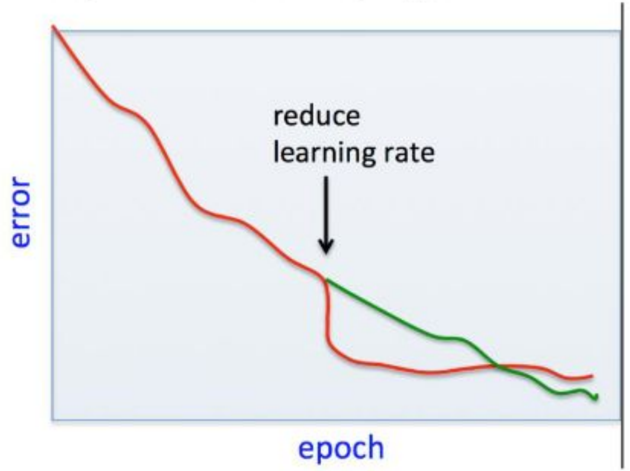
➢ **The choice of α is crucial!**



https://raw.githubusercontent.com/yandexdataschool/nlp_course/master/resources/slides/nlp18_13_abstractive_summarization.pdf

Training The Transformer

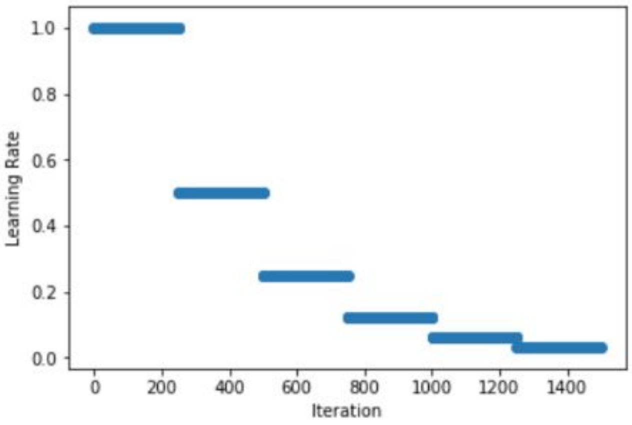
- ➢ Traditional approach: decrease learning rate in stages
  - every **k** steps or whenever progress slows down

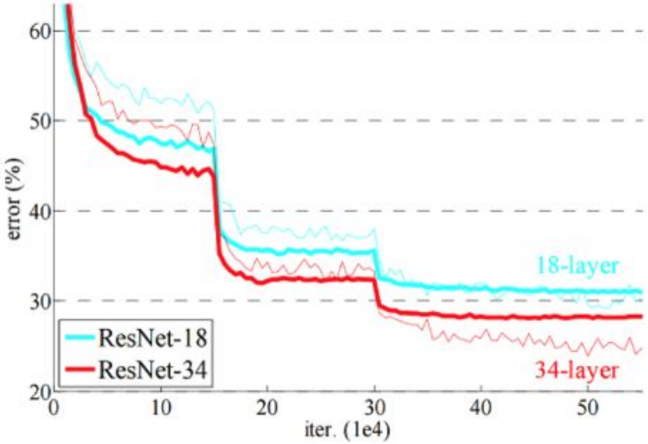


https://raw.githubusercontent.com/yandexdataschool/nlp_course/master/resources/slides/nlp18_13_abstractive_summarization.pdf

Training The Transformer

- ➢ Traditional approach: decrease learning rate in stages
  - ○ every **k** steps or whenever progress slows down





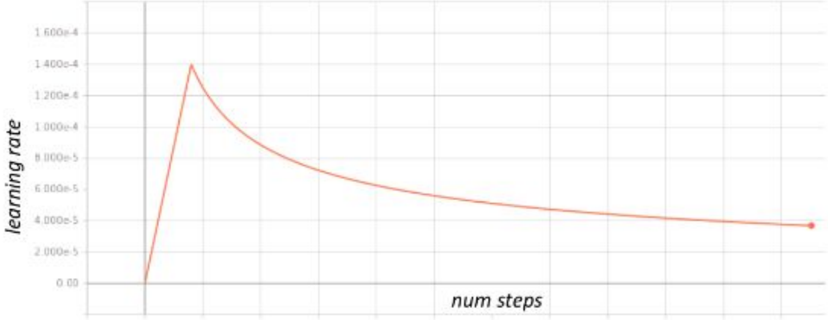
https://raw.githubusercontent.com/yandexdataschool/nlp_course/master/resources/slides/nlp18_13_abstractive_summarization.pdf

Training The Transformer

- Problem: first k steps of Adam are unstable
  - it needs time to accumulate statistics
- Use warmup time!
  keep lr small over first epochs: $\alpha = \alpha_{base} \cdot min(growth(t), decay(t))$

$$growth(t) = \frac{t}{T_{warmup}}$$

$$decay(t) = \sqrt{\frac{T_{warmup}}{t}}$$



https://raw.githubusercontent.com/yandexdataschool/nlp_course/master/resources/slides/nlp18_13_abstractive_summarization.pdf

Training The Transformer

Transformers trainings requires non-trivial efforts

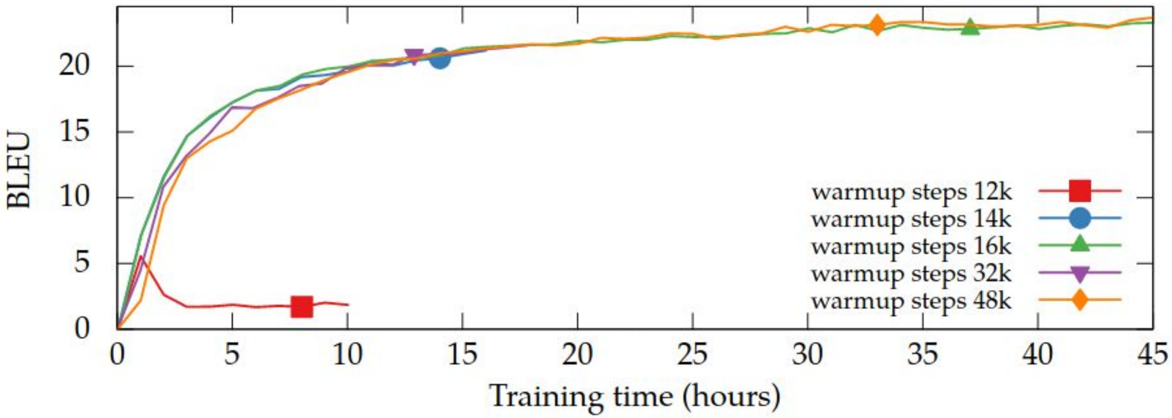


Figure 8: Effect of the warmup steps on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and learning rate (0.20).