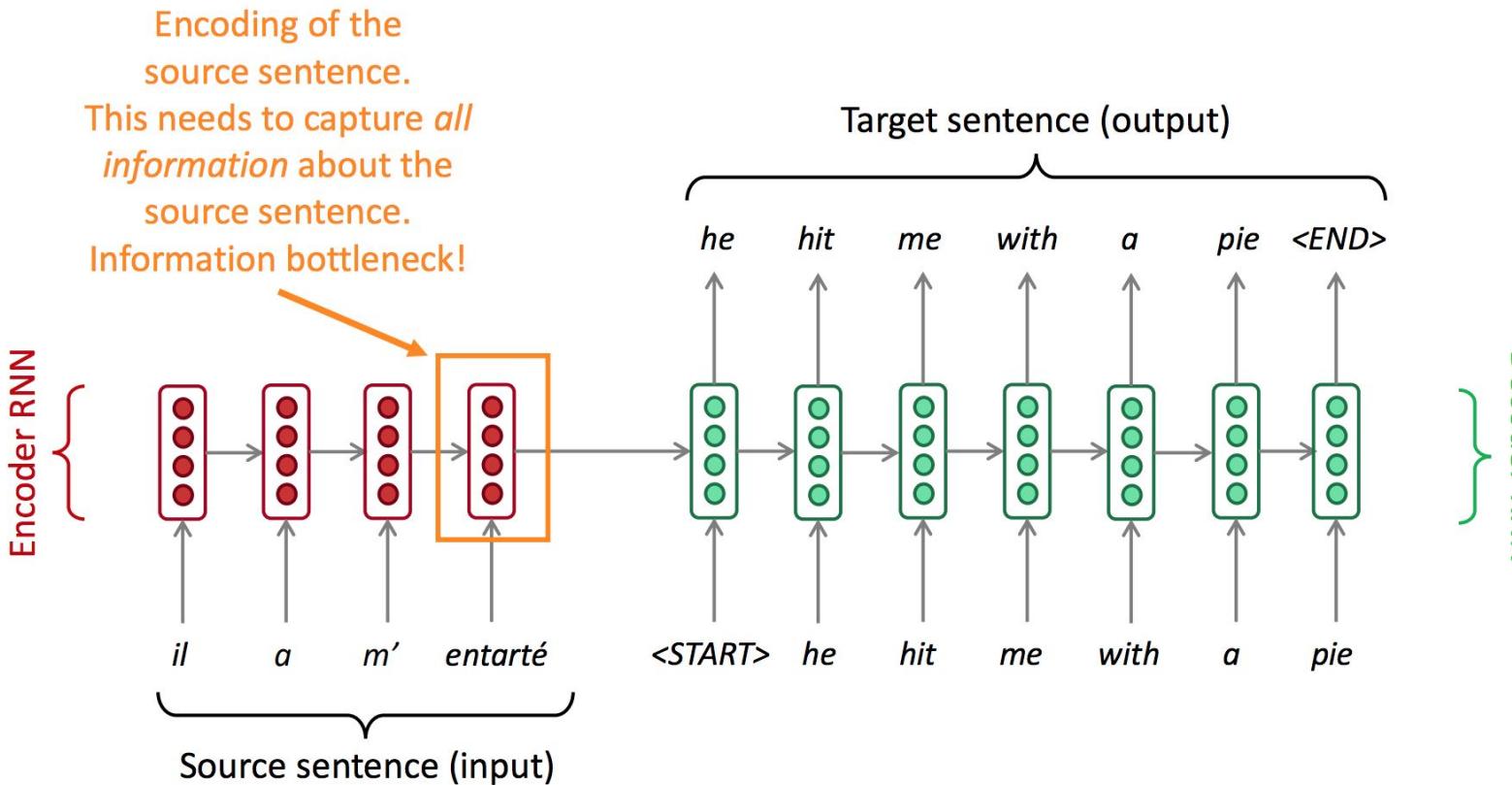


Lecture 3: Self-attention & Transformer

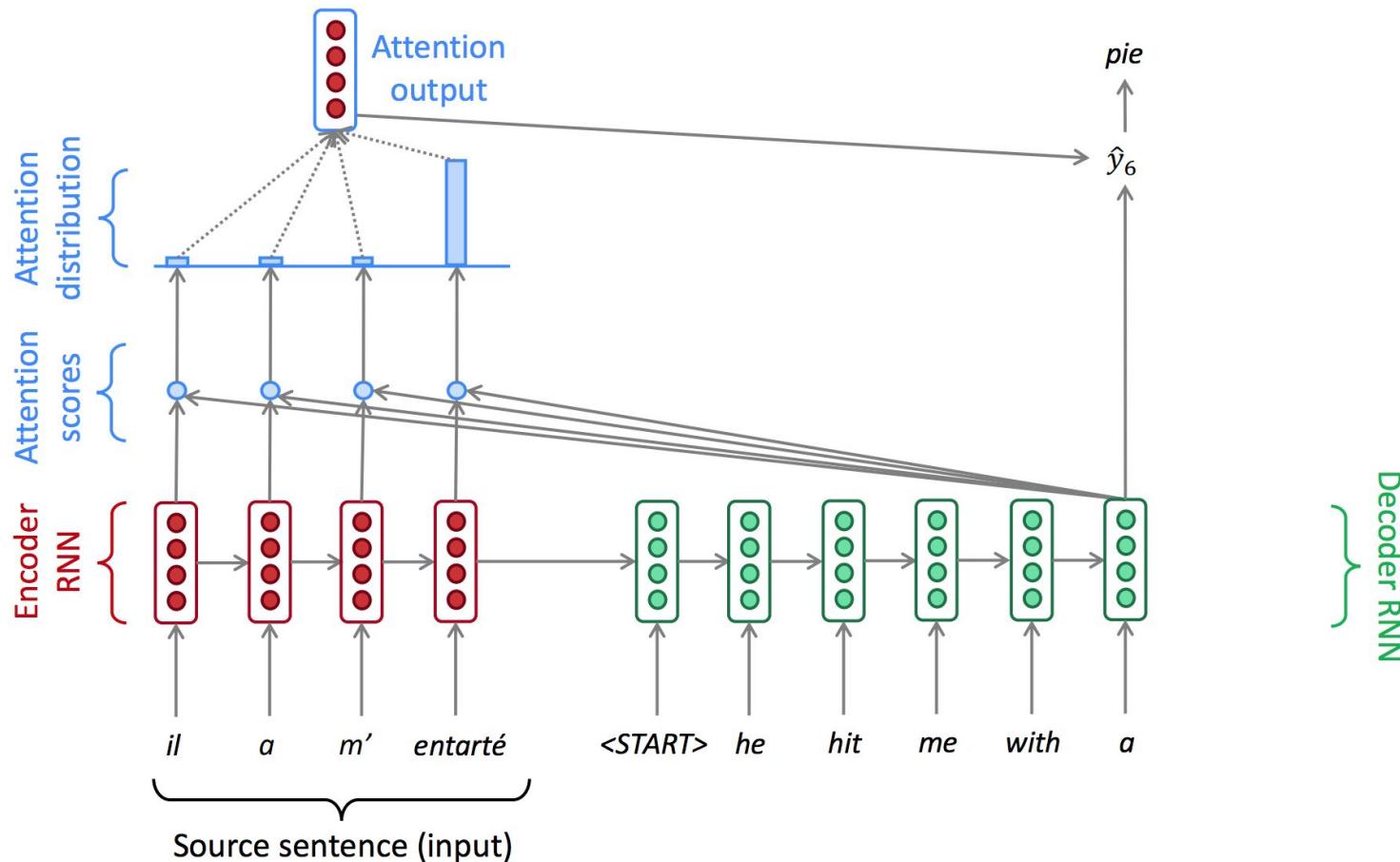
Nikolay Karpachev
19.02.24

1. recap: Attention in seq2seq
2. CNN for textual data
3. Transformer architecture
4. Self-Attention
5. Positional encoding
6. Layer normalization
7. Q & A

recap: Attention in seq2seq



recap: Attention in seq2seq



recap: Attention in seq2seq

We have encoder hidden states $h_1, \dots, h_N \in \mathbb{R}^h$

On timestep t , we have decoder hidden state $s_t \in \mathbb{R}^h$

We get the attention scores e^t for this step:

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

Finally we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

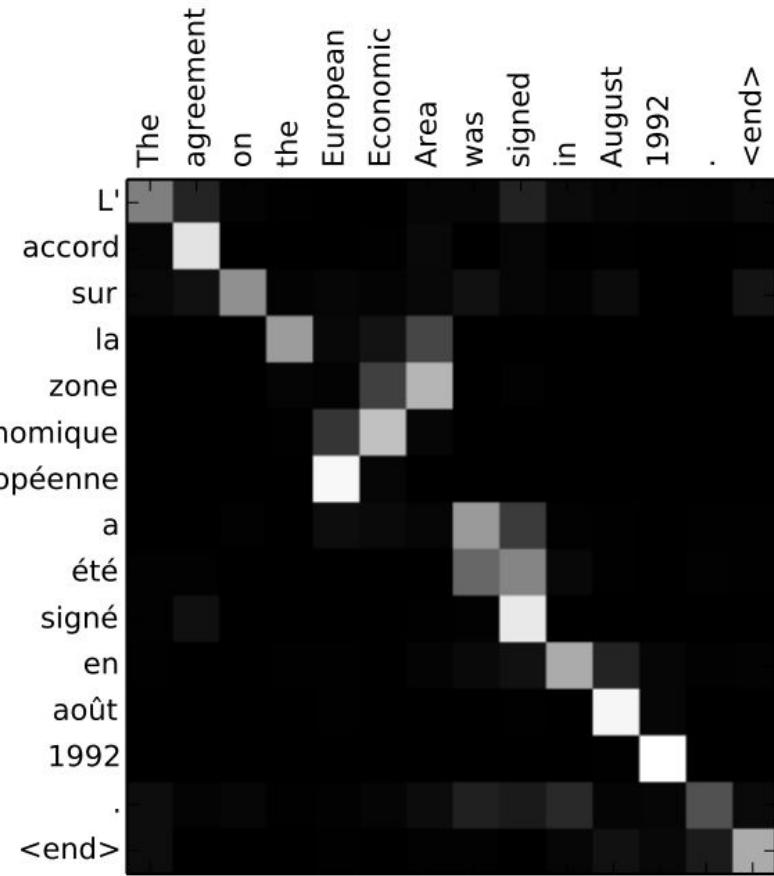
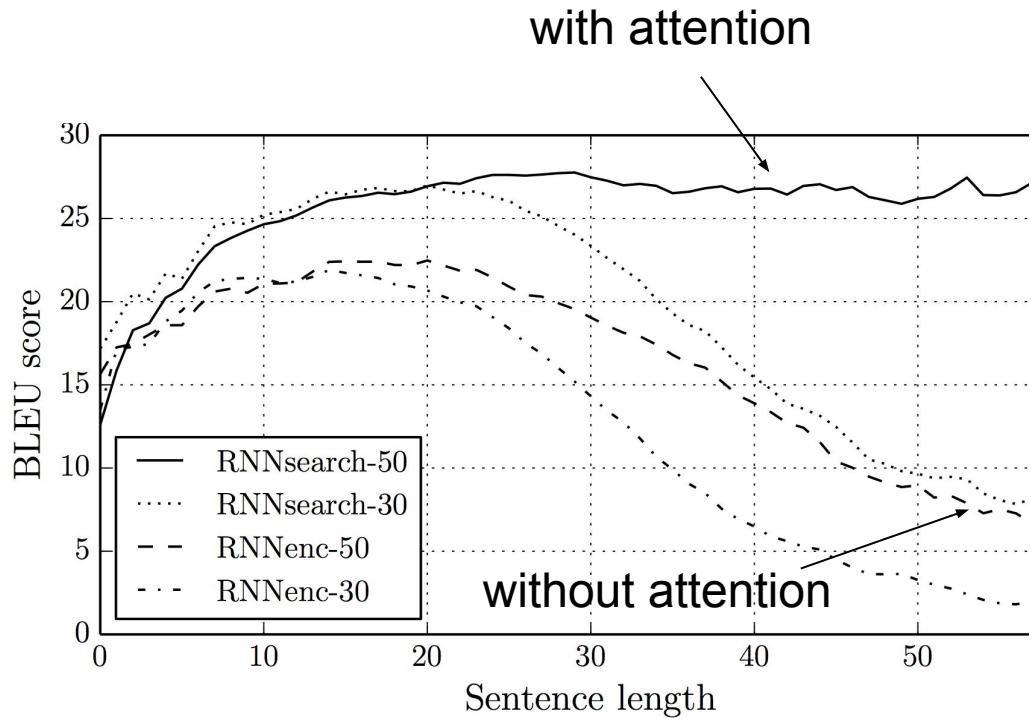
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

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

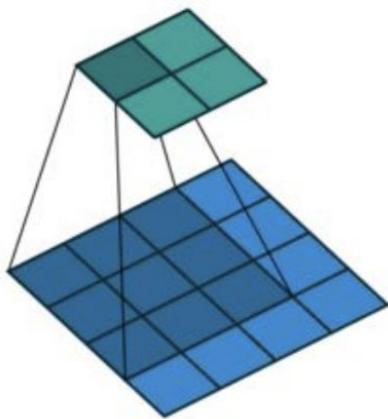
Attention advantages

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

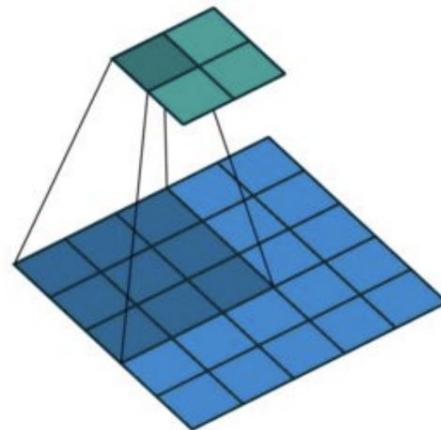


CNN for textual data

CNN recap

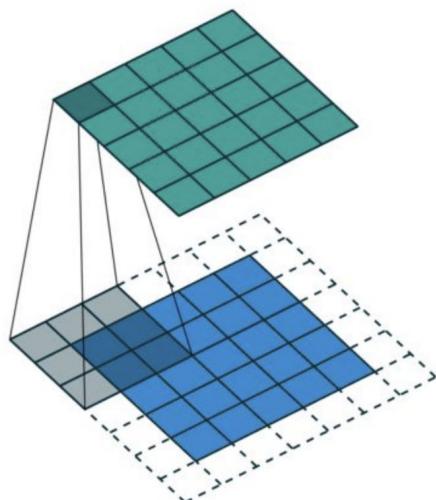


No padding,
no strides

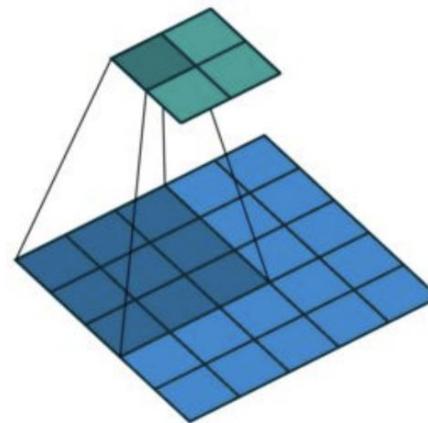


No padding,
with strides

CNN recap

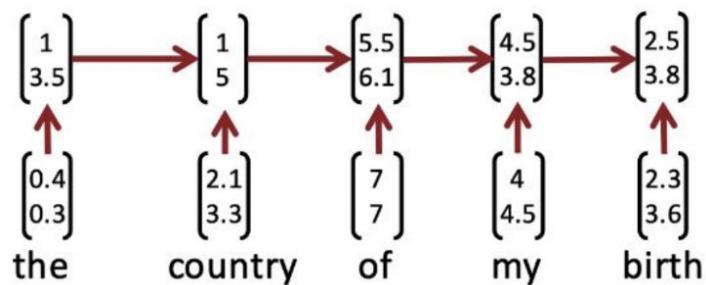


With padding,
no strides



No padding,
with strides

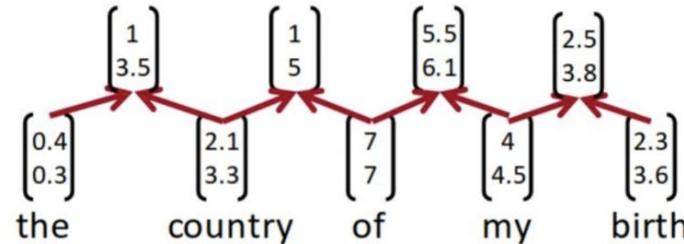
From RNN to CNN



- Recurrent neural nets can not capture phrases without prefix context and often capture too much of last words in final vector

From RNN to CNN

- Imagine using only bigrams
- Same operation as in RNN, but for every pair
- Can be interpreted as convolution over the word vectors



$$p = \tanh \left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b \right)$$

One layer CNN

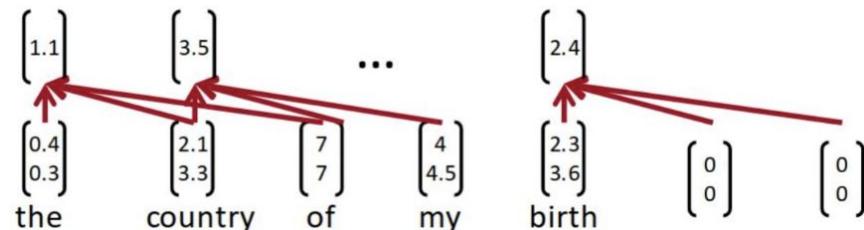
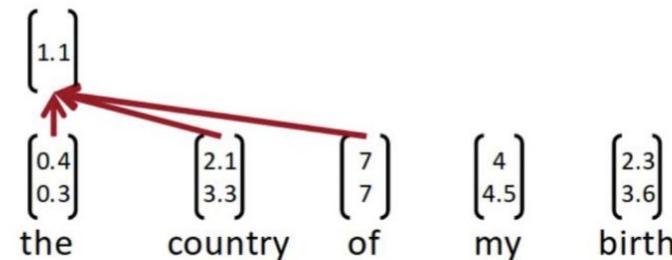
- Simple convolution + pooling
- Window size may be different (2 or more)
- The feature map based on bigrams:

$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

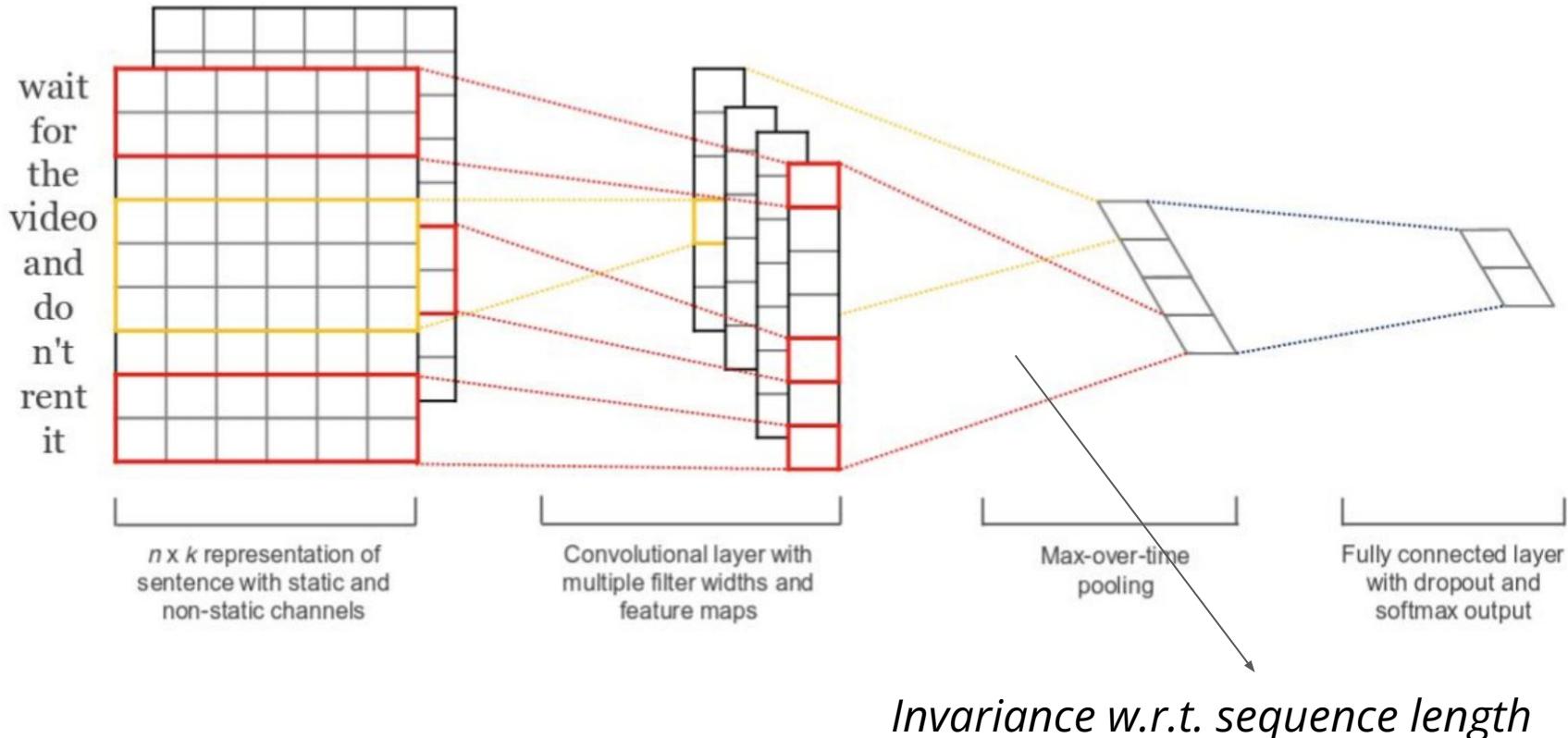
What's next?

We need more features!

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

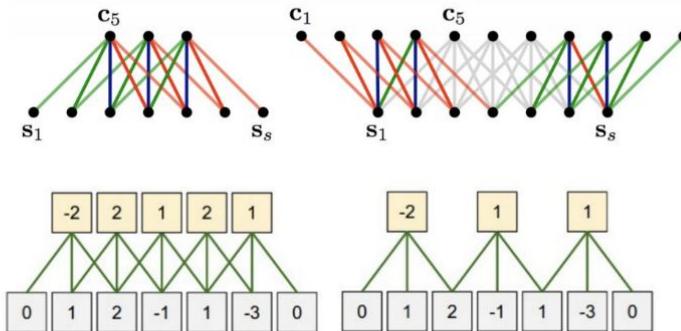


CNN over text embeddings: general scheme

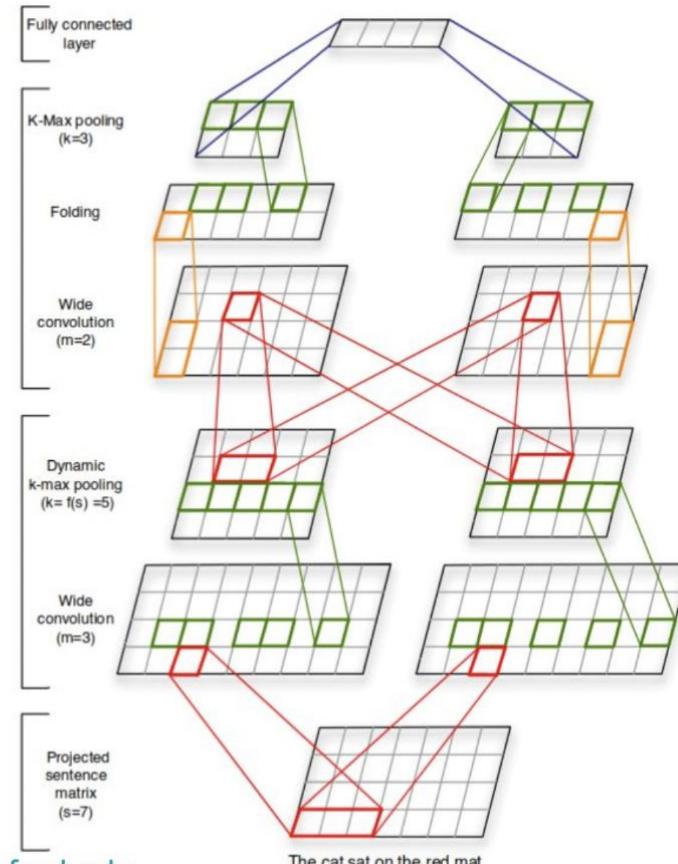


More about CNN

- Narrow vs wide convolution (stride and zero-padding)

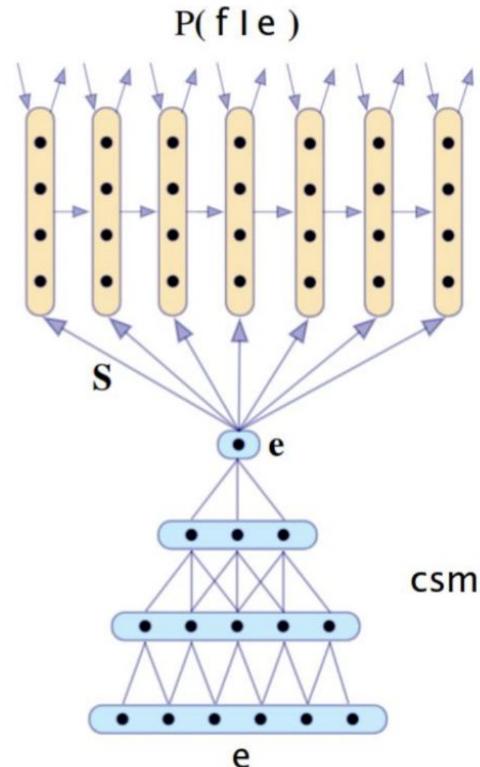


- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)



CNN in seq2seq

- Neural machine translation: CNN as encoder, RNN as decoder
- One of the first neural machine translation efforts
- Paper: [Recurrent Continuous Translation Models, Kalchbrenner and Blunsom, 2013](#)

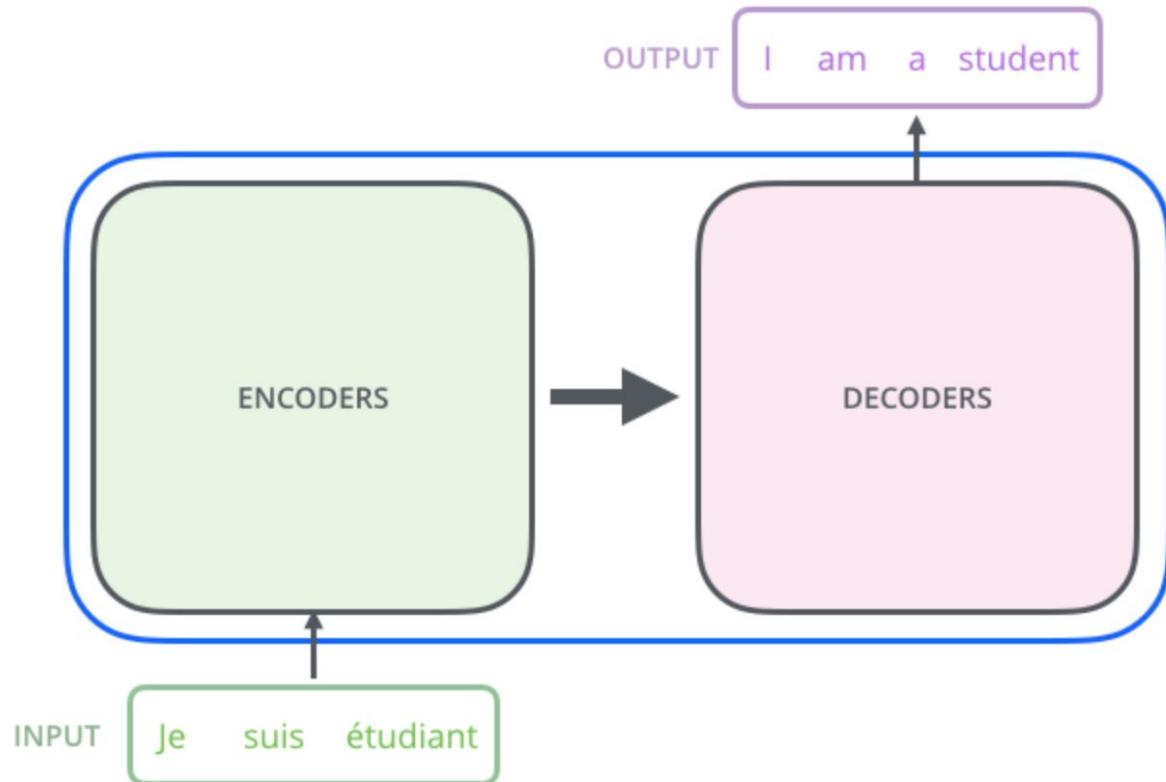


The Transformer

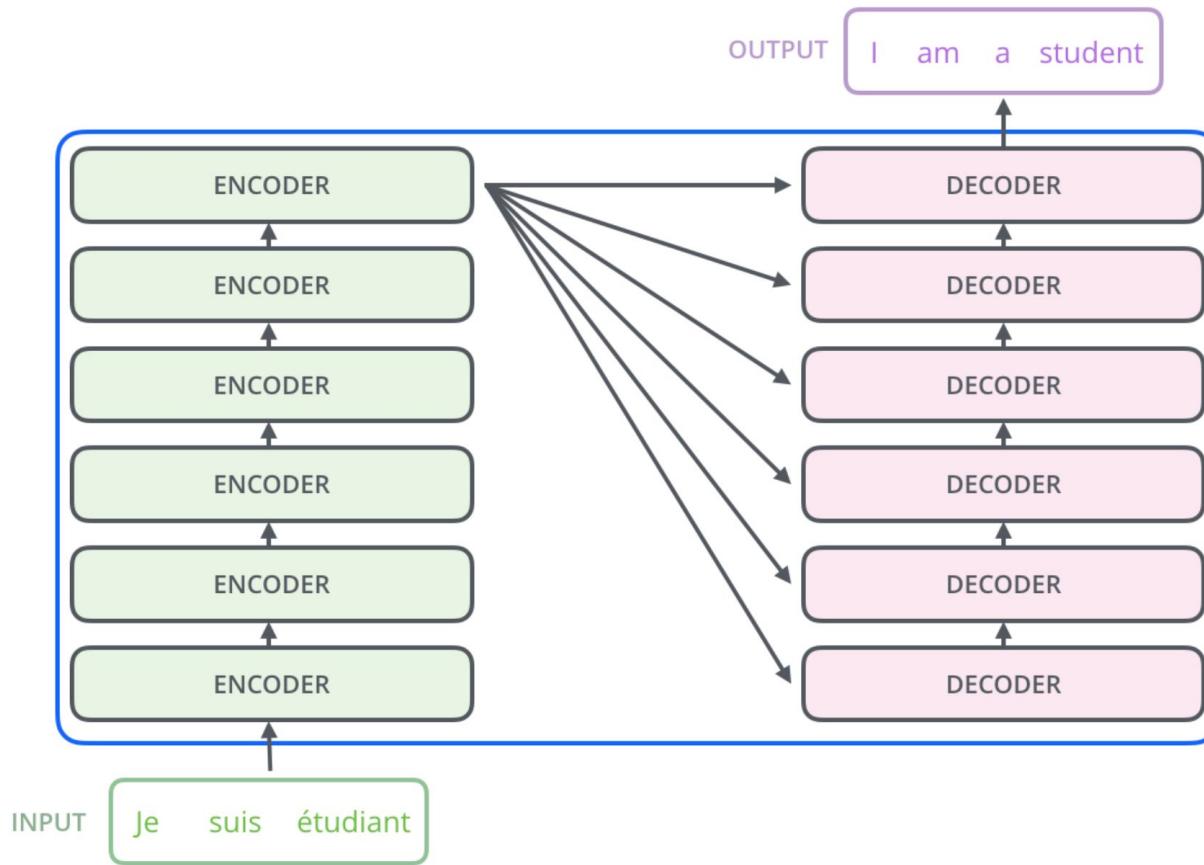
The Transformer



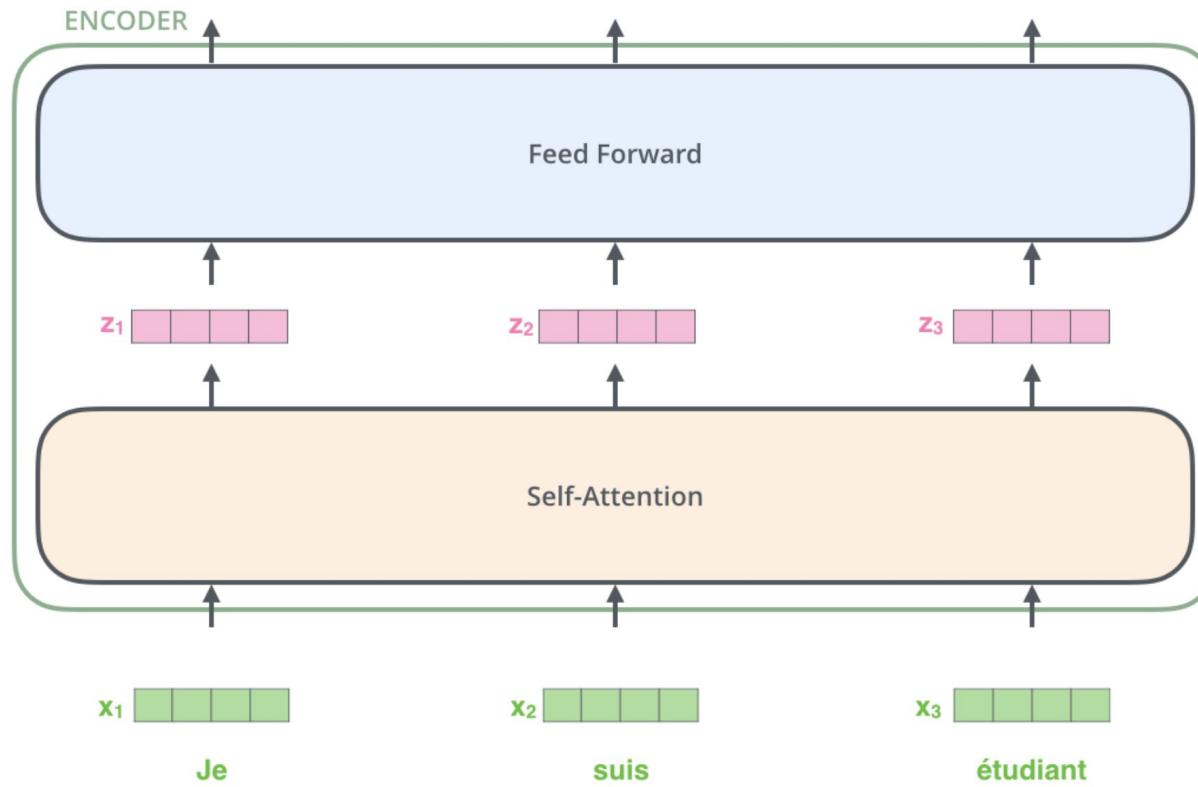
The Transformer



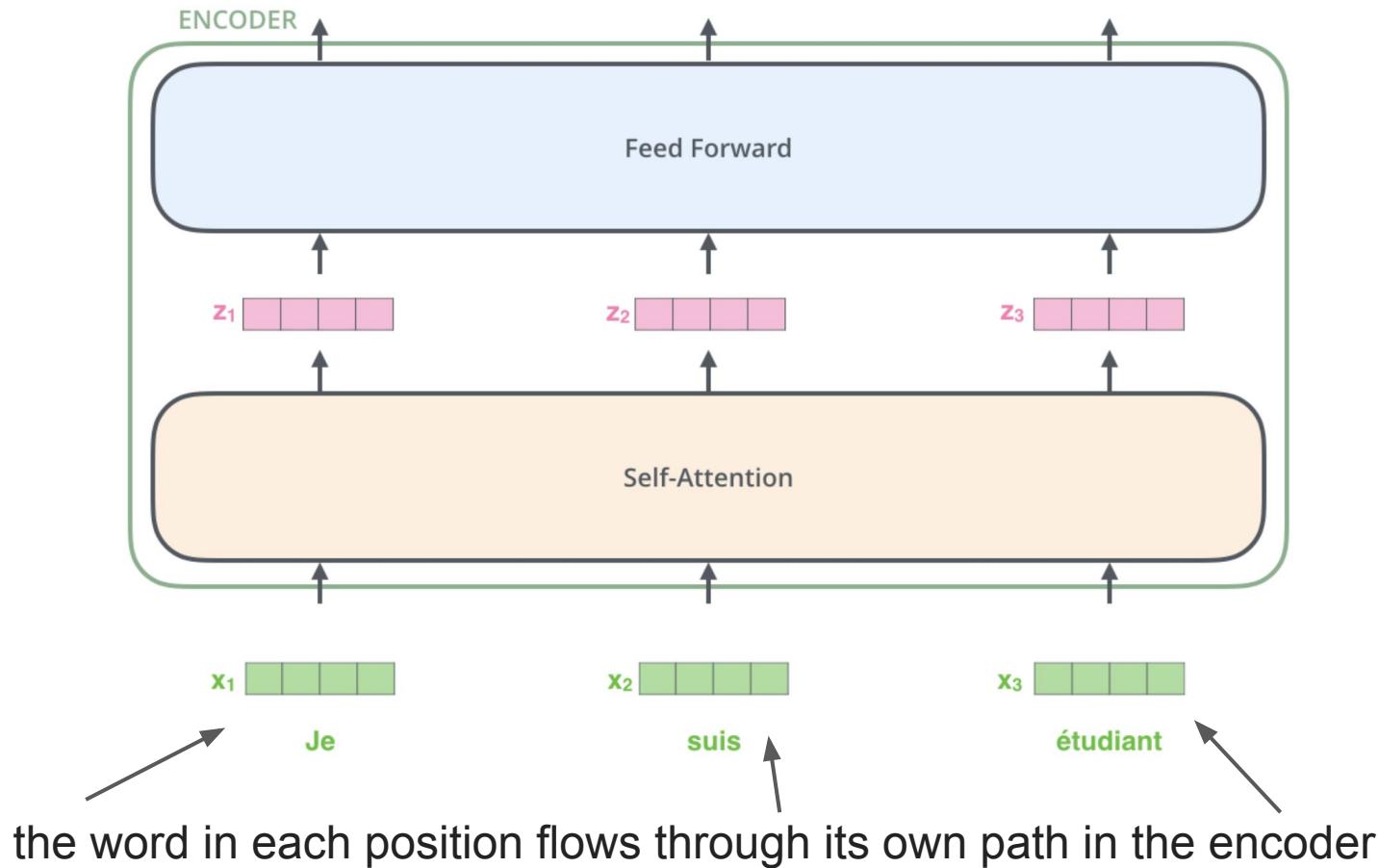
The Transformer



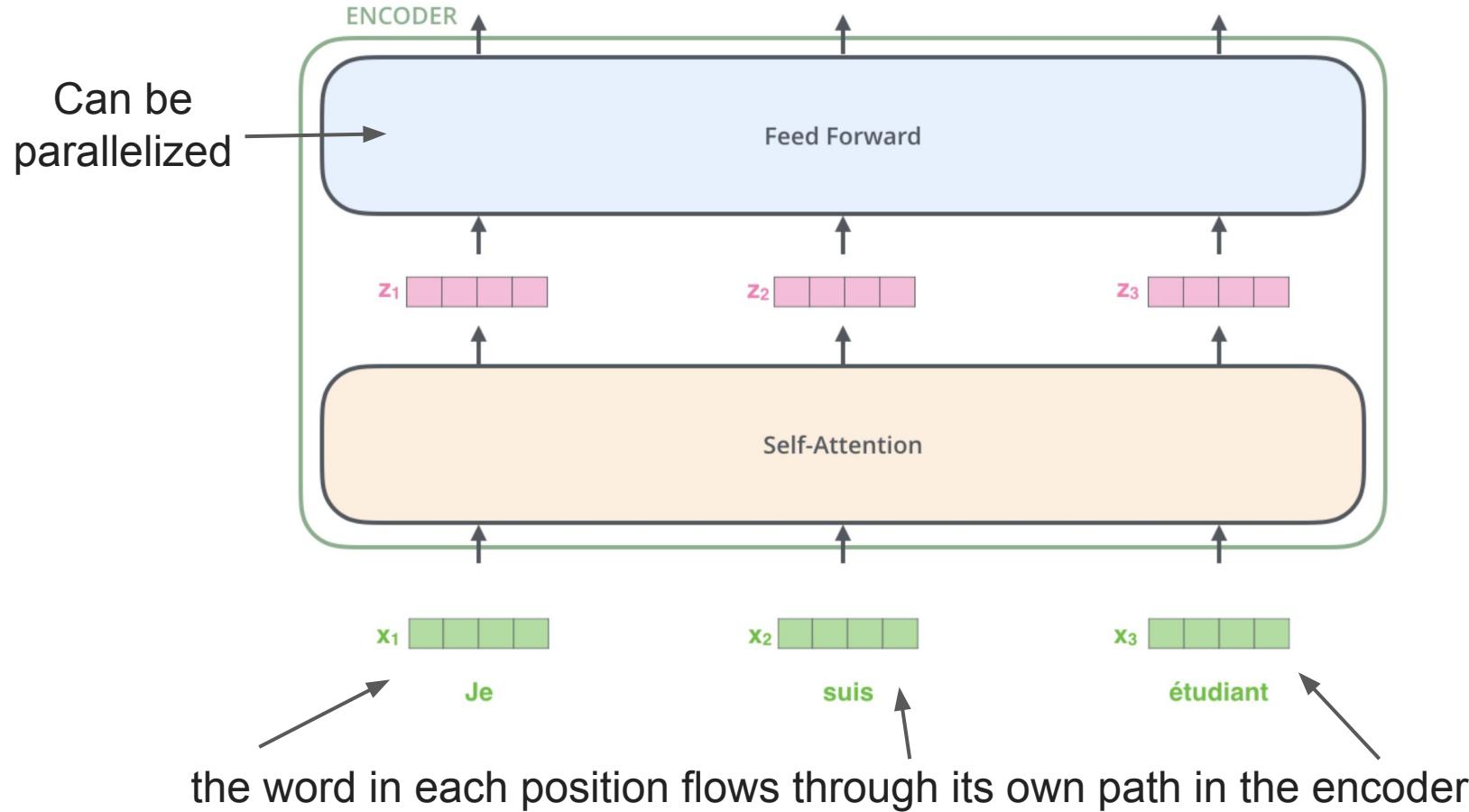
The Encoder Side



The Encoder Side

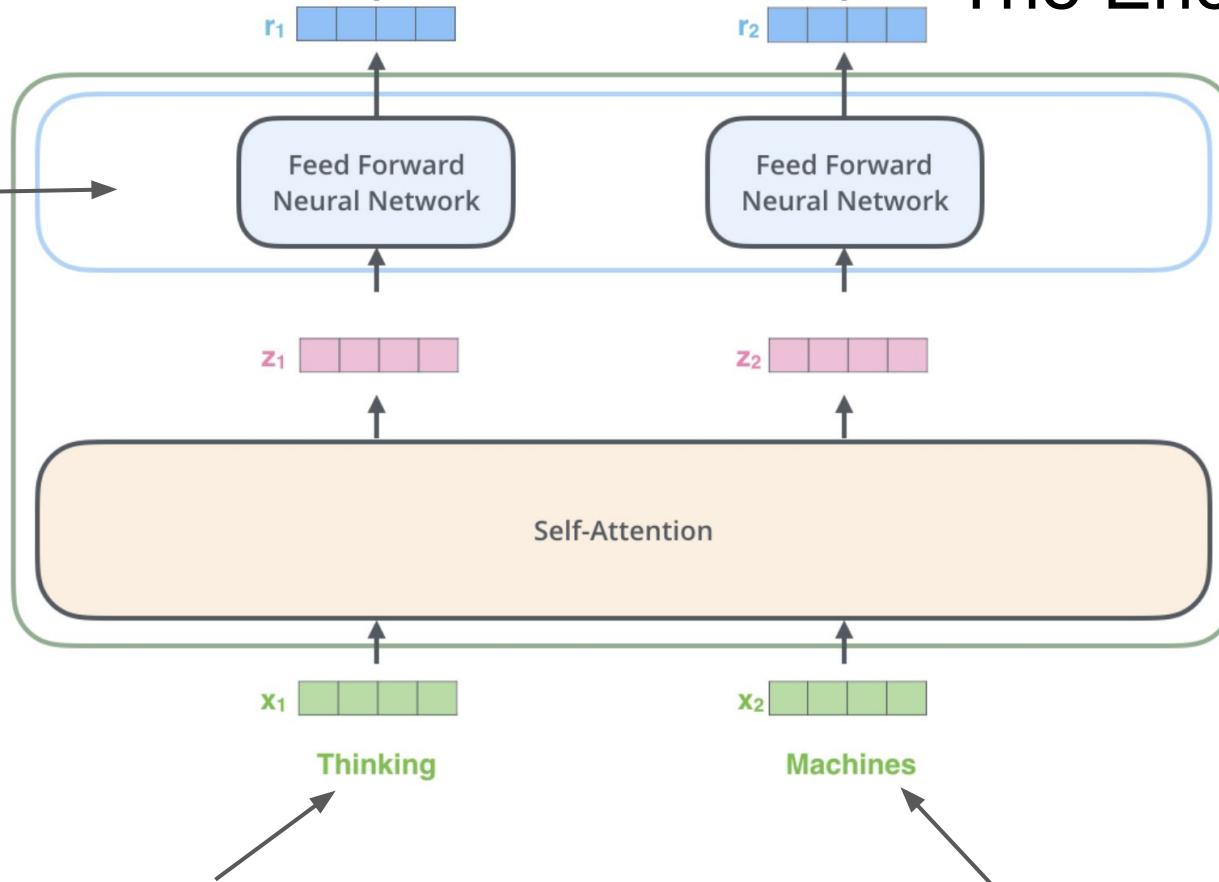


The Encoder Side



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 the paper “Attention is All You Need” (Ashish Vaswani et al.)
- No recurrent or convolutional neural networks -> just 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?

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 at a High Level

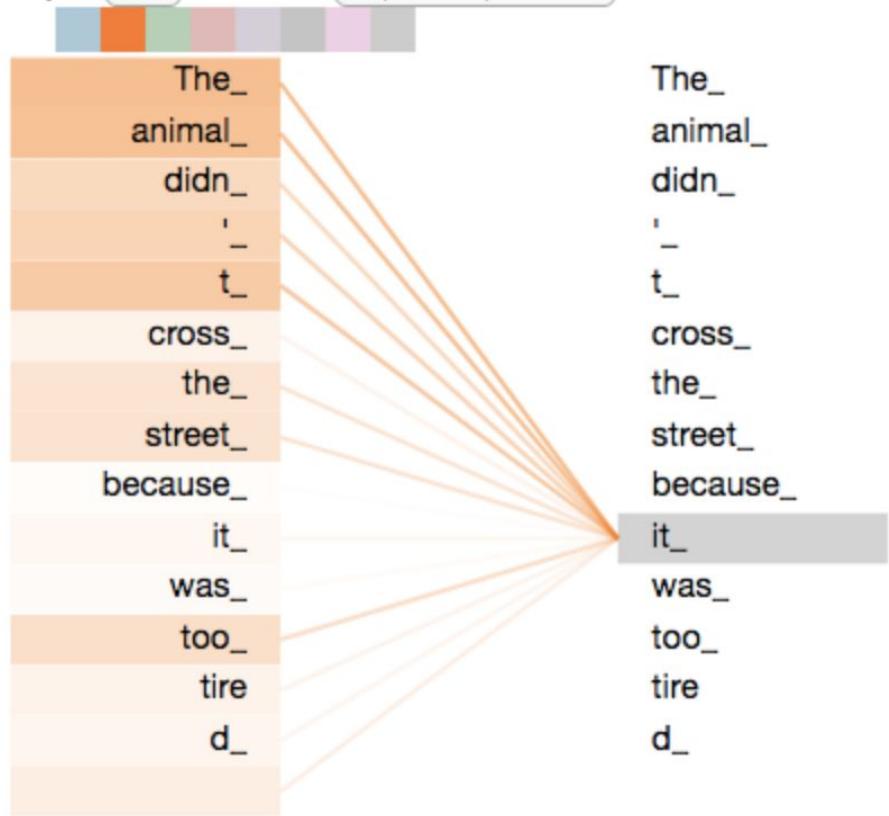
”The animal didn't cross the street because it was too tired”

- 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 ⬆️



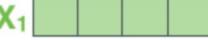
Self-Attention: detailed explanation

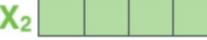
Input

Thinking

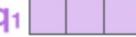
Machines

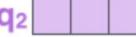
Embedding

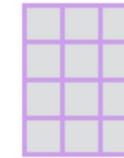
X_1 

X_2 

Queries

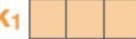
q_1 

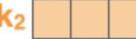
q_2 

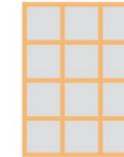


W^Q

Keys

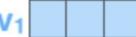
k_1 

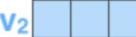
k_2 



W^K

Values

v_1 

v_2 



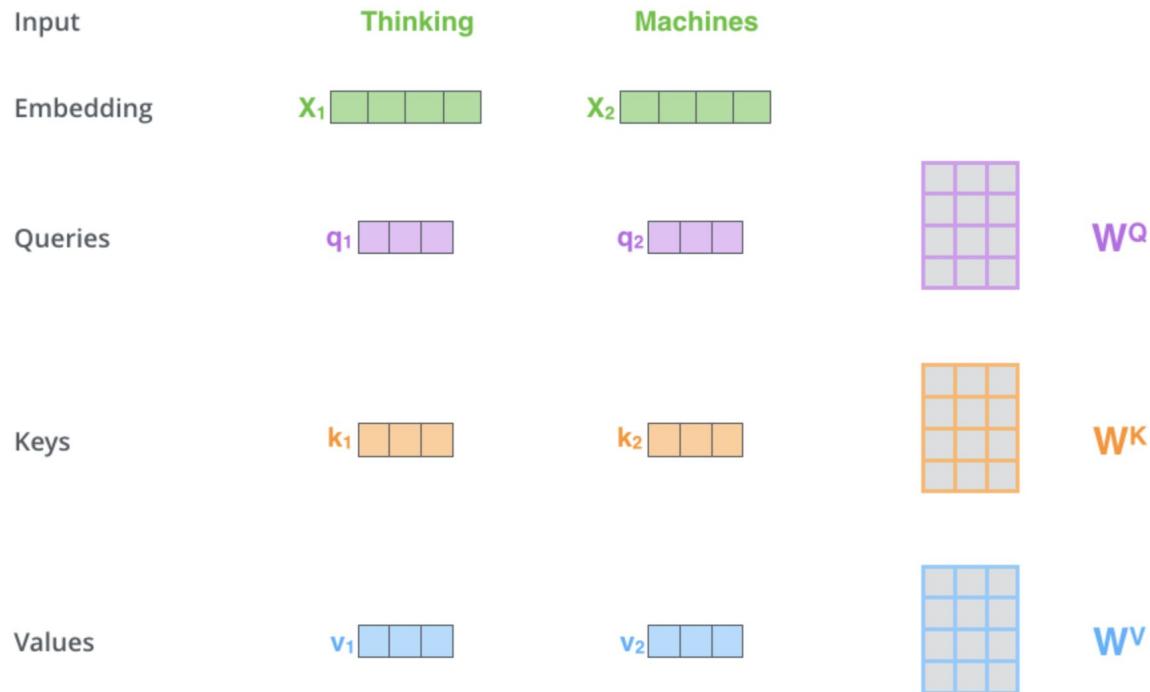
W^V

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”, and “value” vectors?

Self-Attention: detailed explanation

What are the “query”, “key”, and “value” vectors?

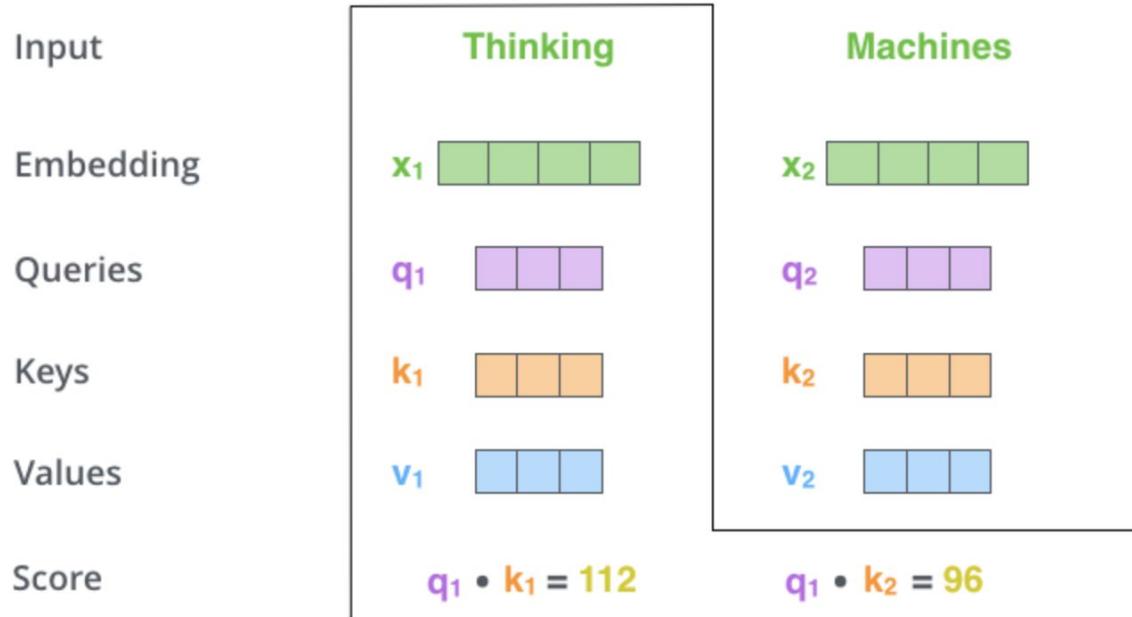
They're abstractions that are useful for calculating and thinking about attention.

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

Input

Embedding

Queries

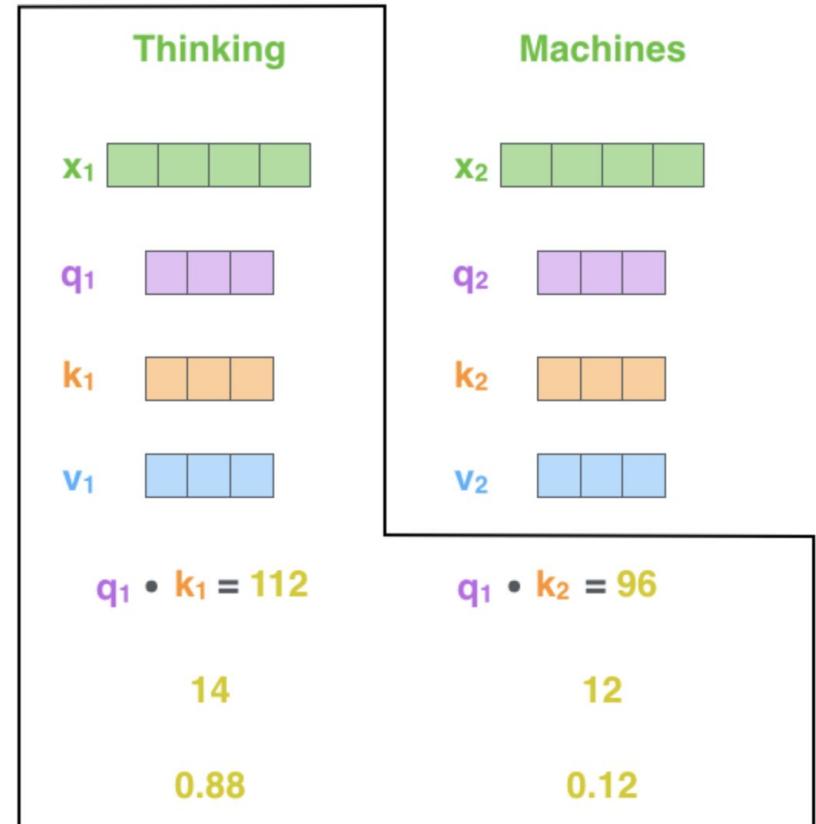
Keys

Values

Score

Divide by 8 ($\sqrt{d_k}$)

Softmax



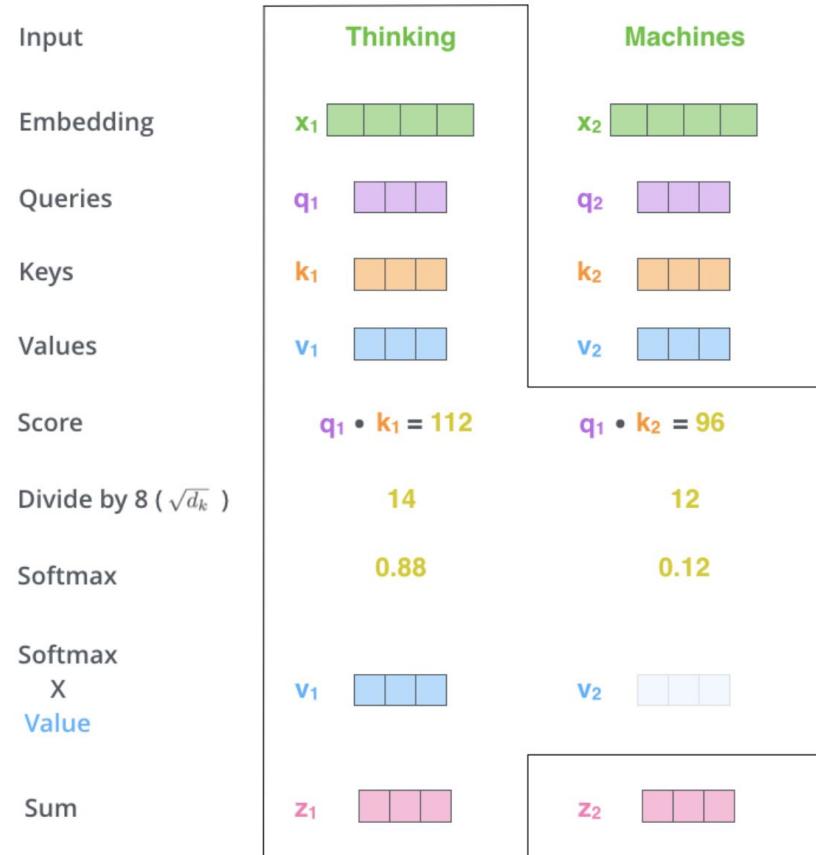
Self-Attention: detailed explanation

STEP 5:

multiply each value vector by the softmax score

STEP 6:

sum up the weighted value vectors



Self-Attention

Input

Embedding

Queries

Keys

Values

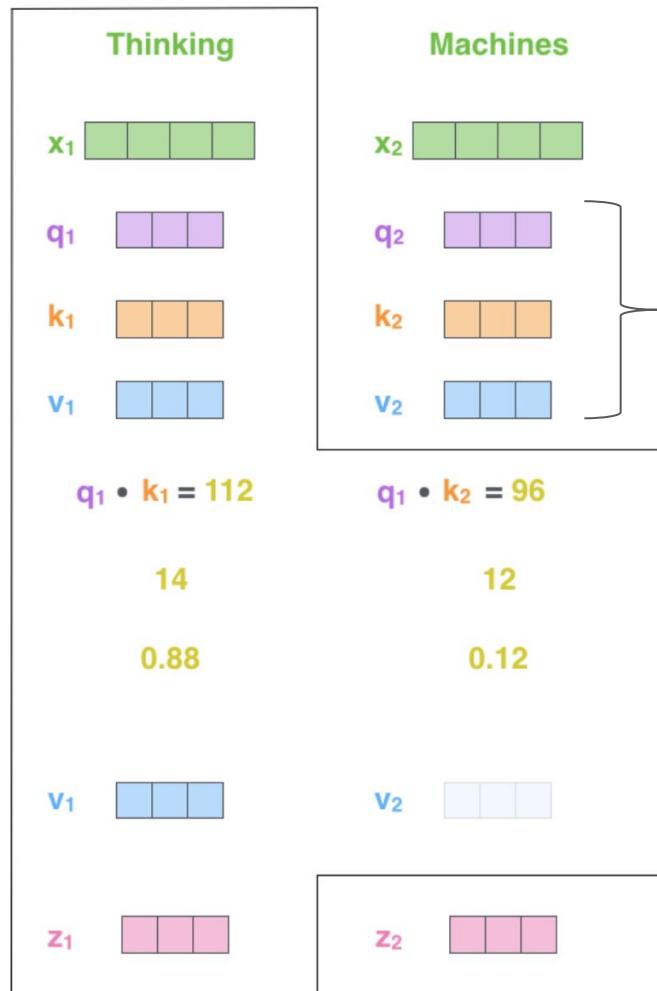
Score

Divide by 8 ($\sqrt{d_k}$)

Softmax

Softmax
X
Value

Sum



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_k**, **W_q**, **W_v**)

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W^Q} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{Q} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W^K} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{K} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

$$\begin{matrix} \mathbf{X} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \mathbf{W^V} \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix} = \begin{matrix} \mathbf{V} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

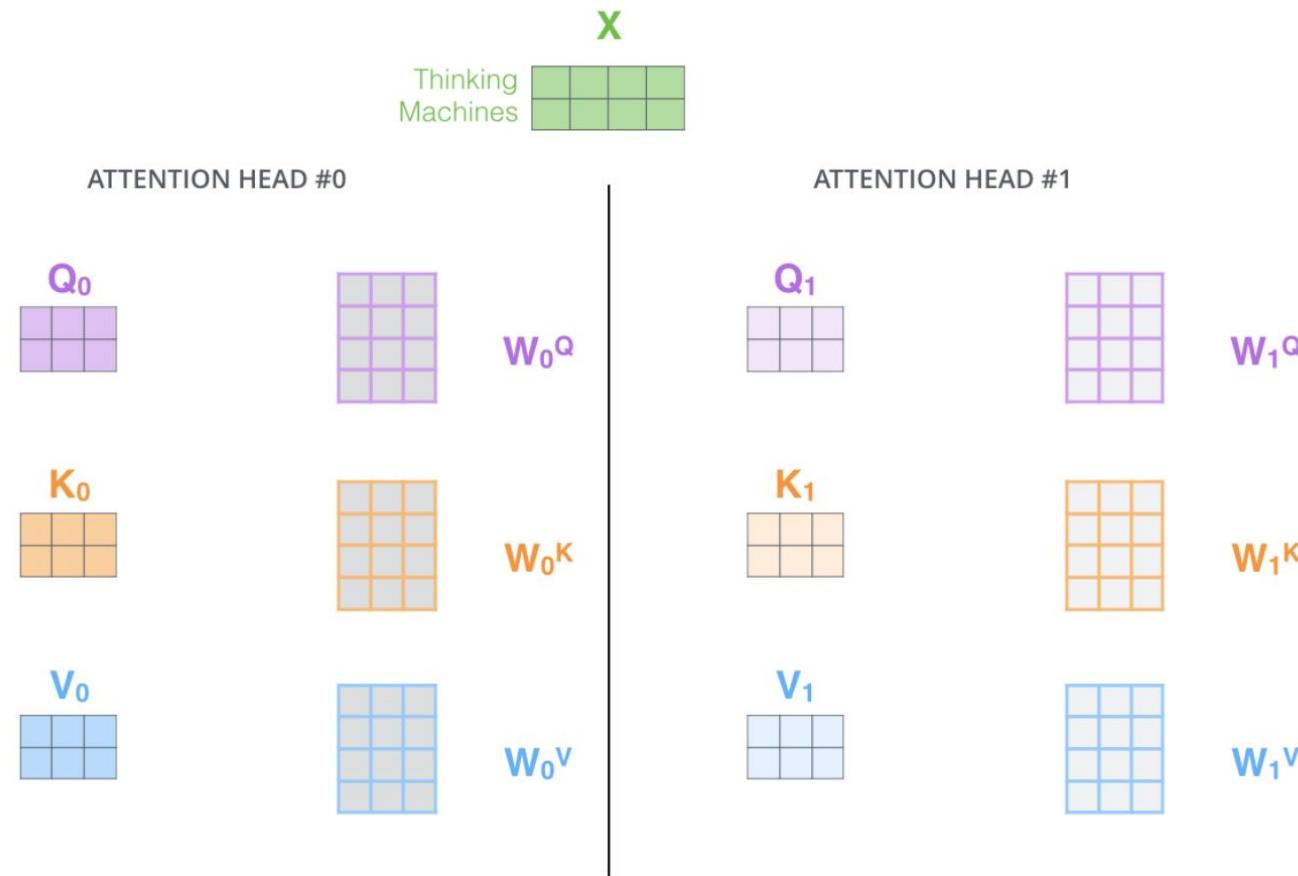
Self-Attention: Matrix Calculation

$$\text{softmax} \left(\frac{\begin{array}{c} \text{Q} \quad \text{K}^T \\ \times \\ \hline \end{array}}{\sqrt{d_k}} \right) \text{V}$$

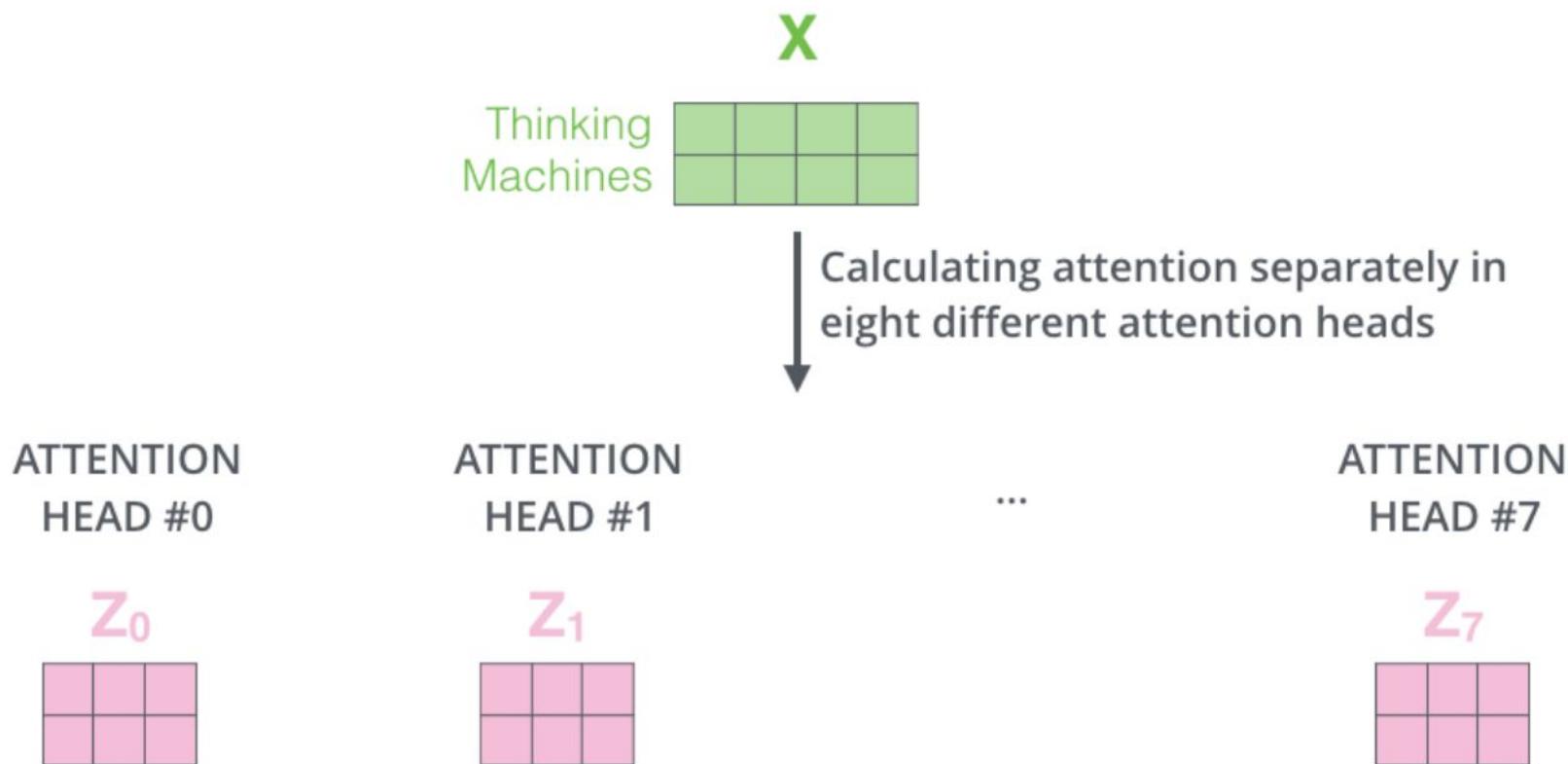
Diagram illustrating the matrix calculation for Self-Attention:

- The input matrix Q is a 3x3 purple square.
- The input matrix K^T is a 3x3 orange square.
- The output matrix V is a 3x3 blue square.
- The result is obtained by multiplying Q and K^T (indicated by \times) and then dividing by $\sqrt{d_k}$.
- The final result is labeled Z , represented by a 3x3 pink square.

Multi-Head Attention



Multi-Head Attention



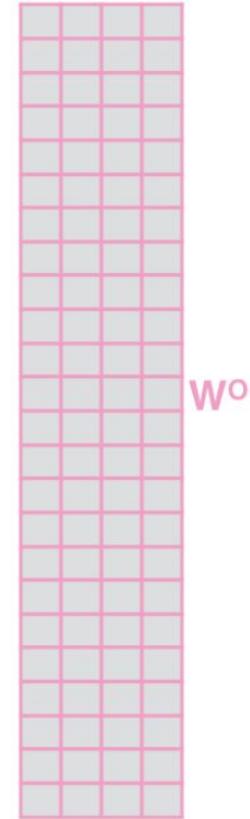
Multi-Head Attention

1) Concatenate all the attention heads



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

X



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

$$= \begin{matrix} Z \\ \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \end{matrix}$$

1) This is our input sentence*

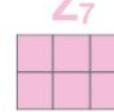
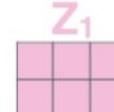
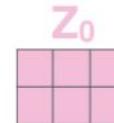
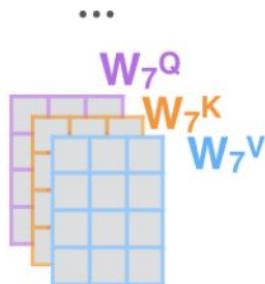
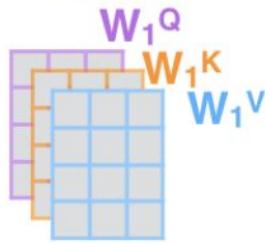
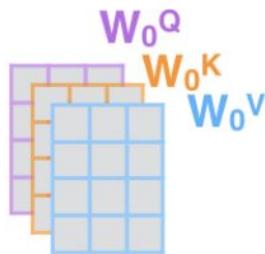
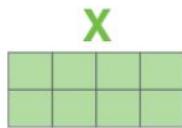
2) We embed each word*

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

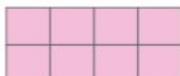
Thinking
Machines



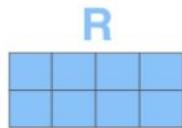
W^o



Z

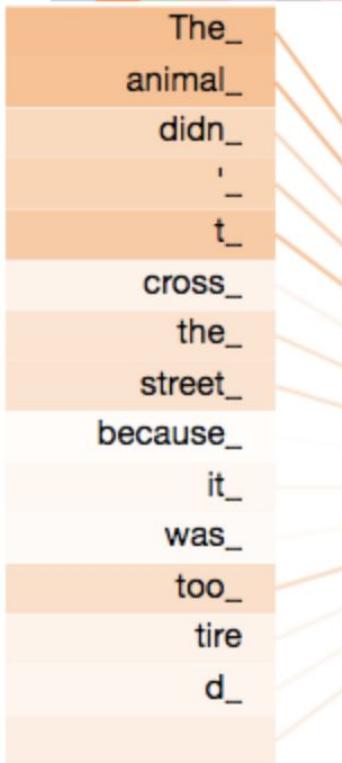


* In all encoders other than #0, we don't need embedding.
We start directly with the output of the encoder right below this one



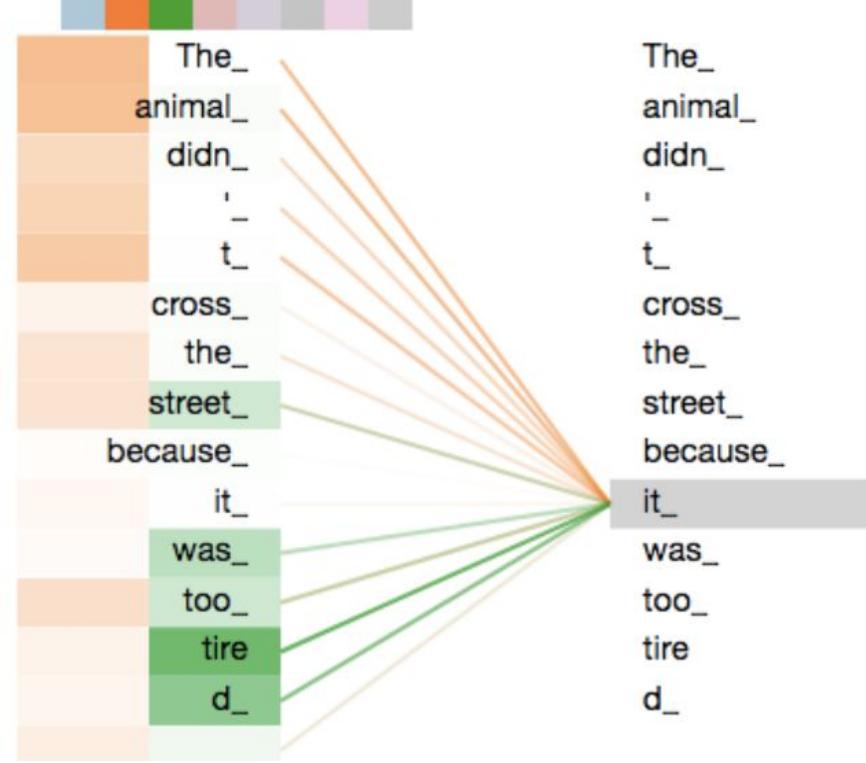
Multi-Head Attention

Layer: 5 ⬆️ Attention: Input - Input ⬆️



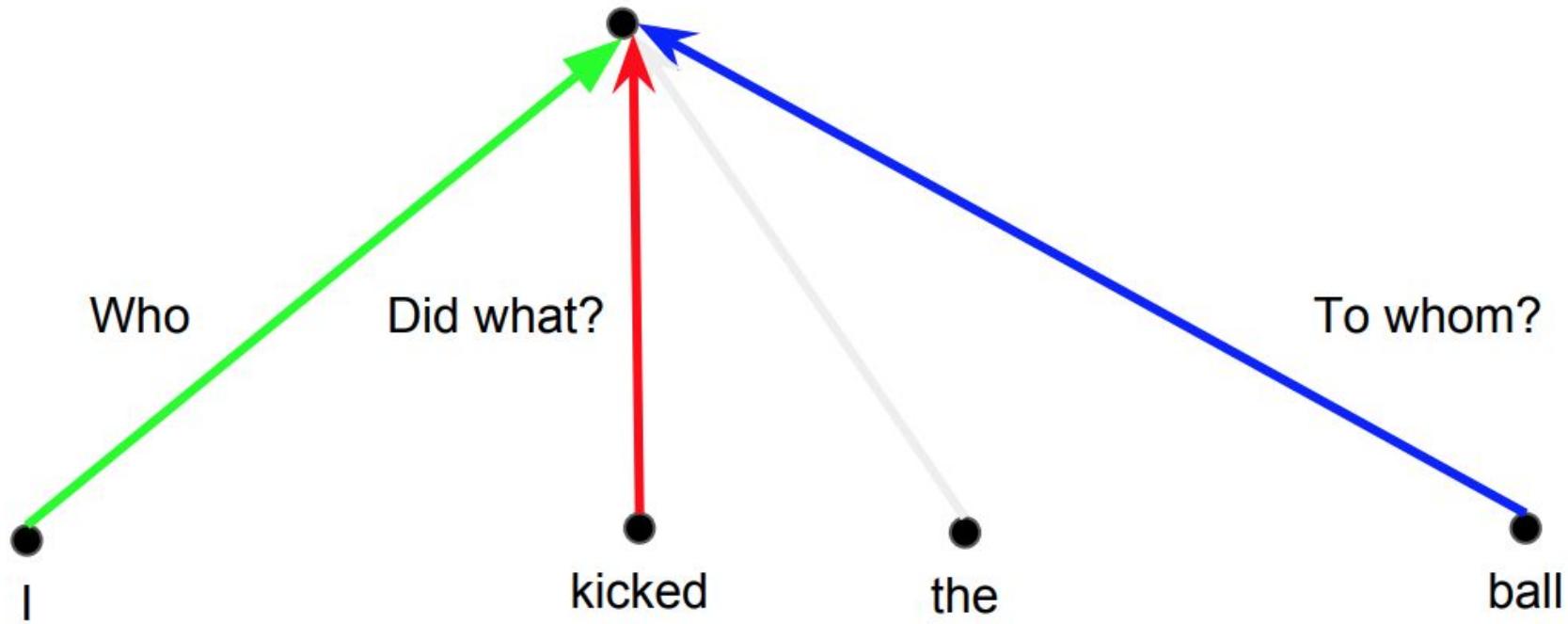
The_ animal_ didn_ '
t_ cross_ the_ street_ because_ it_ was_ too_ tire d_

Layer: 5 ⬆️ Attention: Input - Input ⬆️

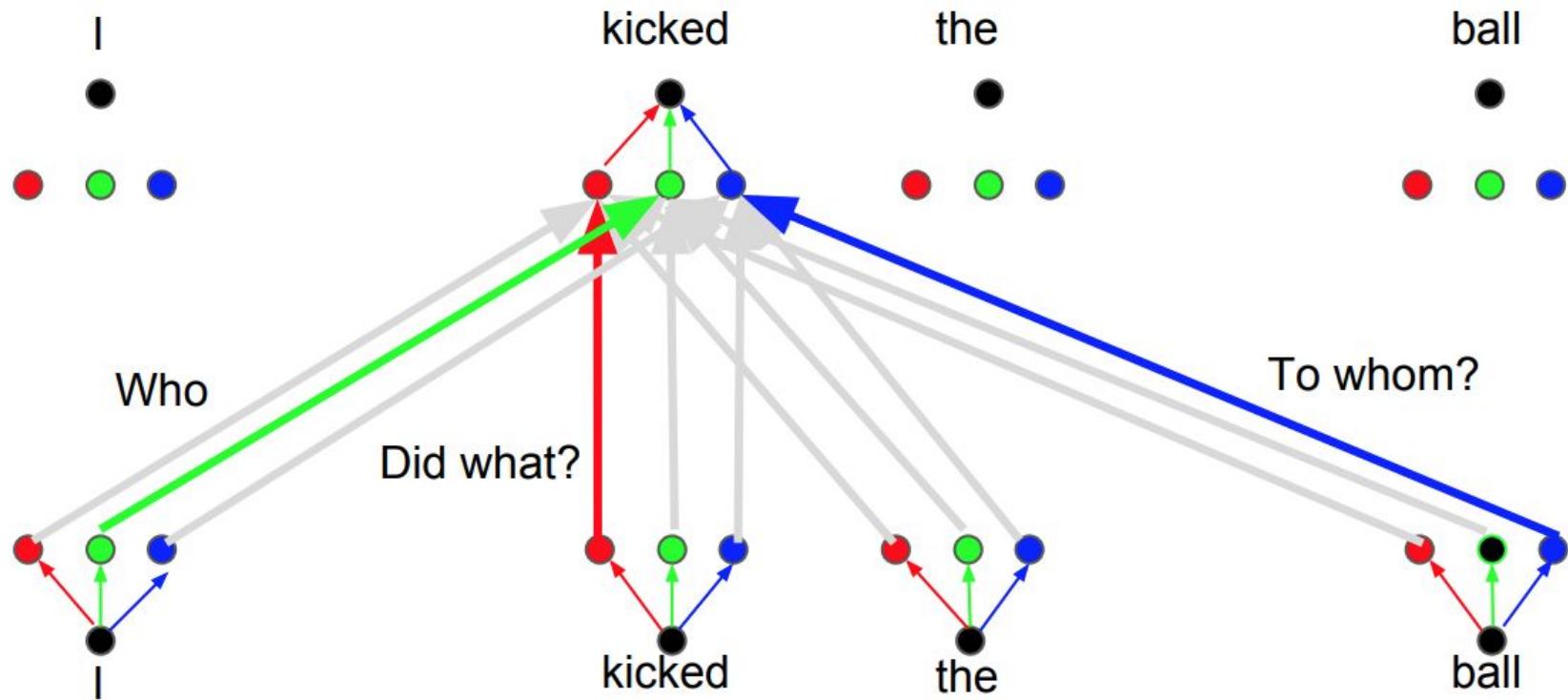


The_ animal_ didn_ '
t_ cross_ the_ street_ because_ it_ was_ too_ tire d_

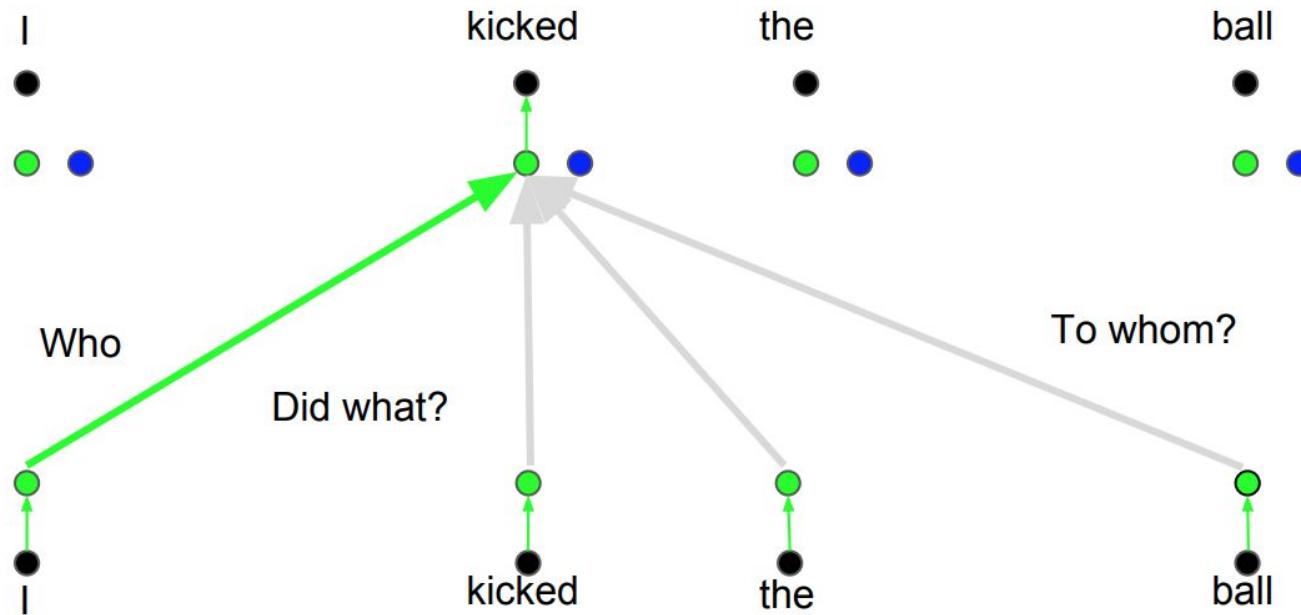
Why Multi-Head Attention?



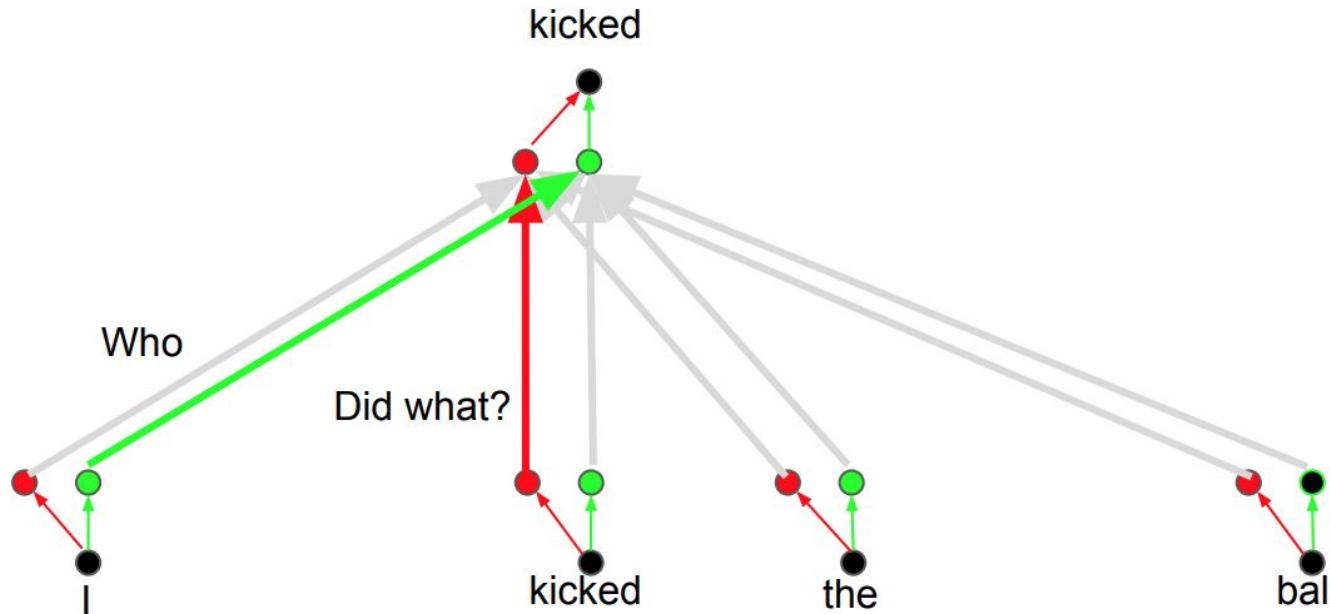
Why Multi-Head Attention?



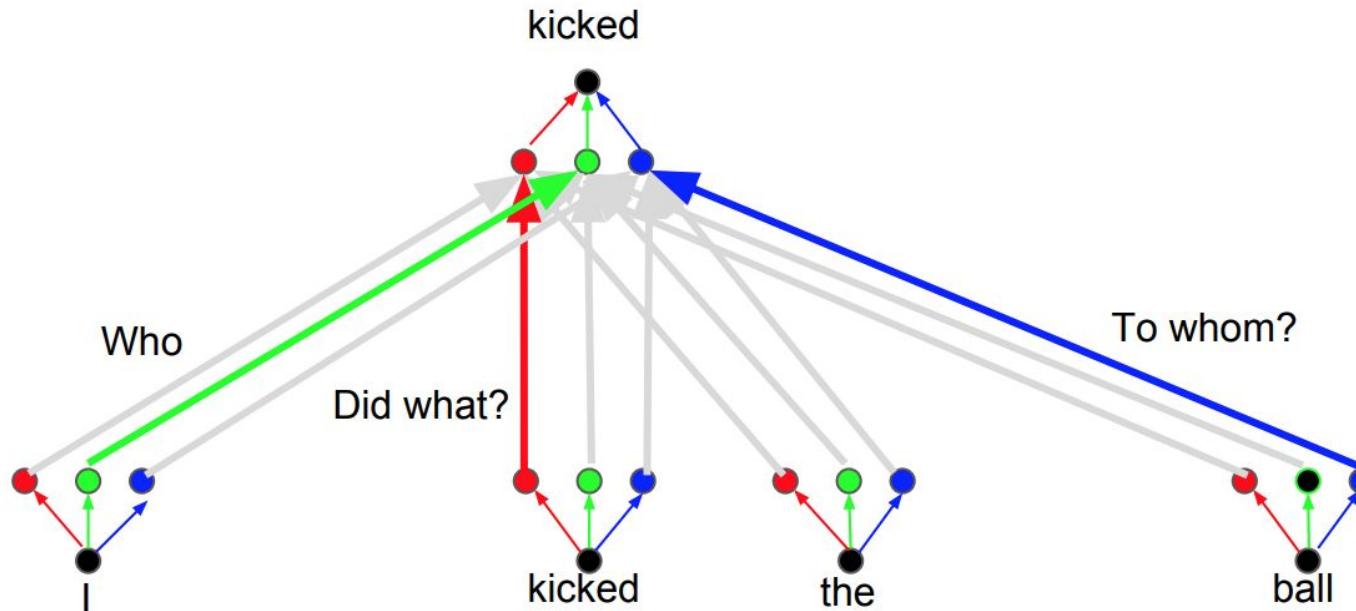
Attention head: Who



Attention head: Did What?

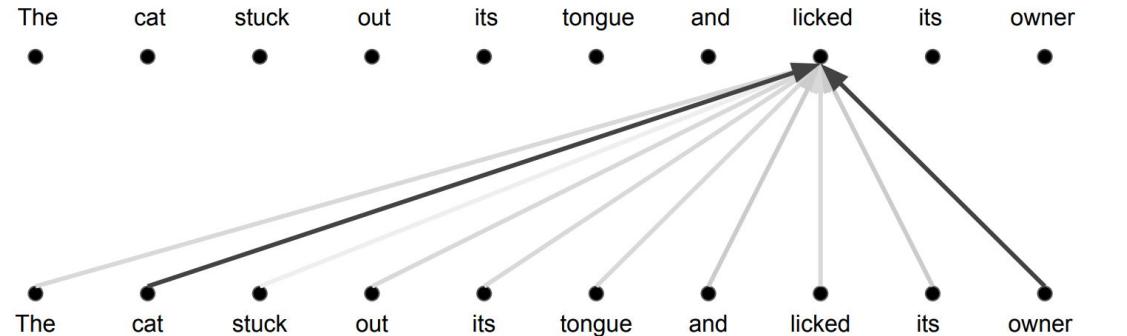


Attention head: To Whom?



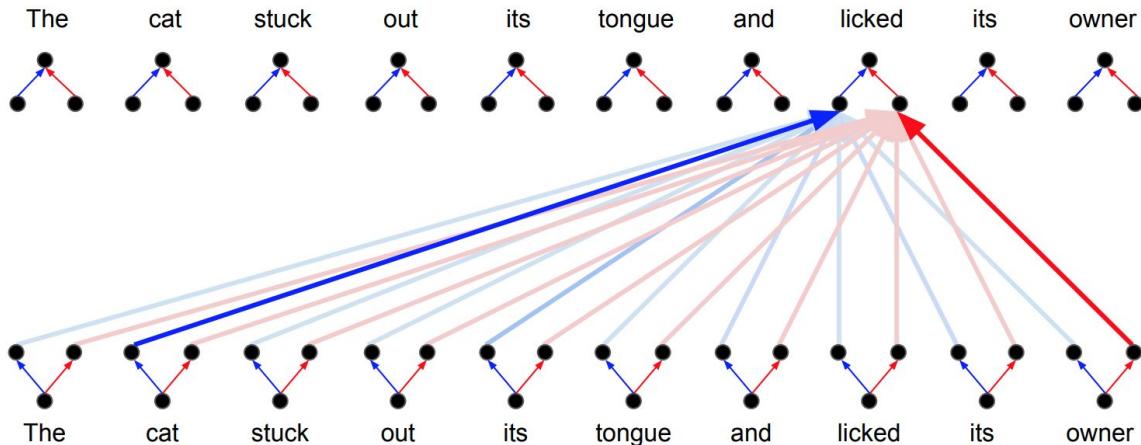
Attention vs. Multi-Head Attention

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

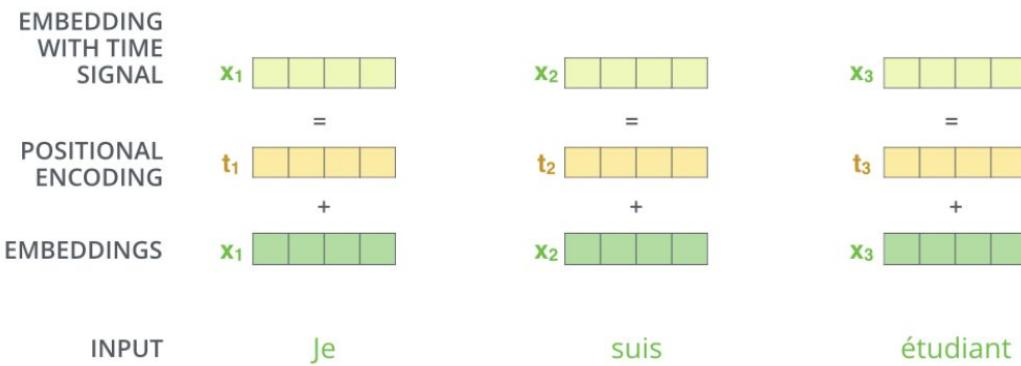
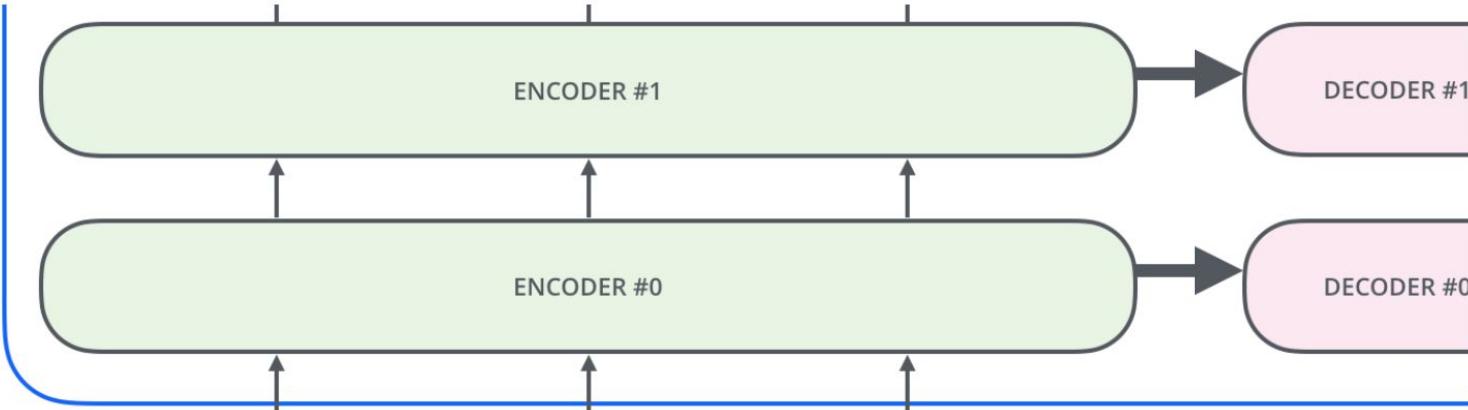
*Transformer models trained >3x faster than the others.

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

Positional Encoding



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

Positional Encoding

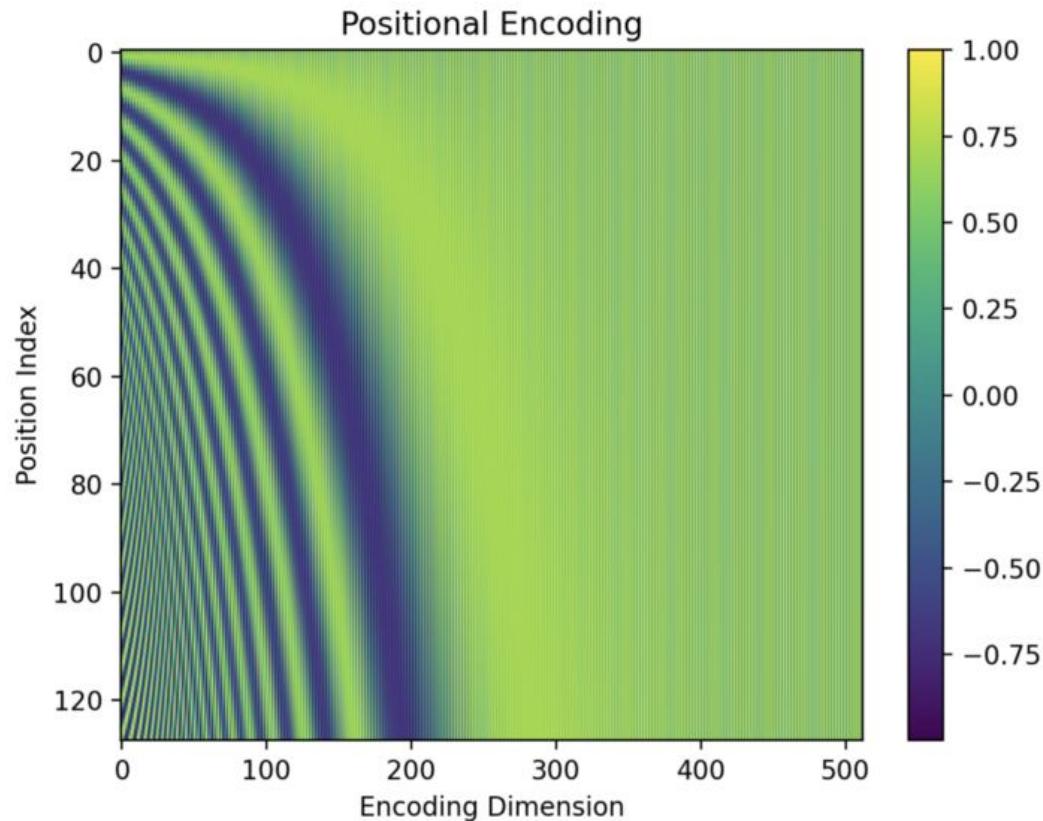
$$PE_{(pos, \ 2i)} = \sin(pos / 10000^{2i / d_{\text{model}}})$$

$$PE_{(pos, \ 2i + 1)} = \cos(pos / 10000^{2i / d_{\text{model}}})$$

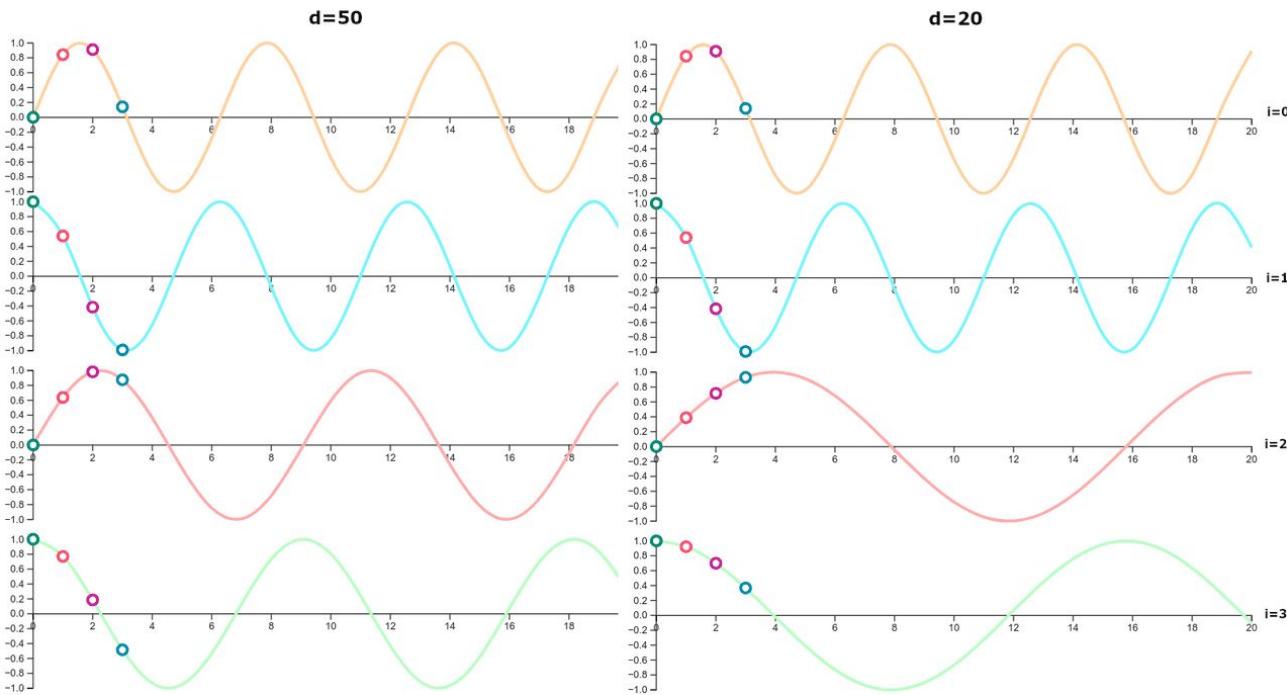
- pos is the position
- i is the dimension.

Each dimension of the positional encoding corresponds to a sinusoid.
The wavelengths form a geometric progression from 2π to $2\pi * 10\ 000$

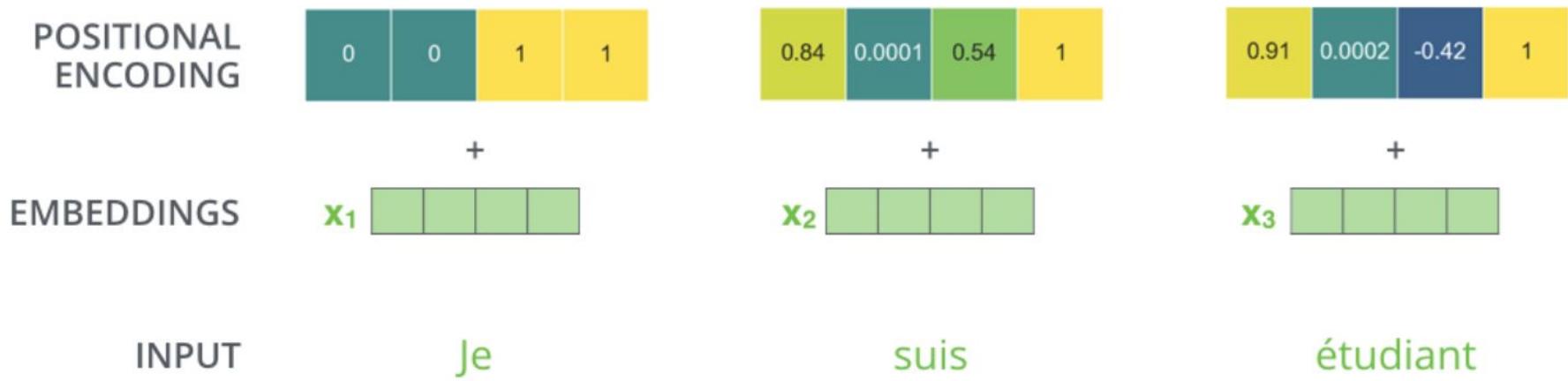
Positional Encoding



Positional Encoding

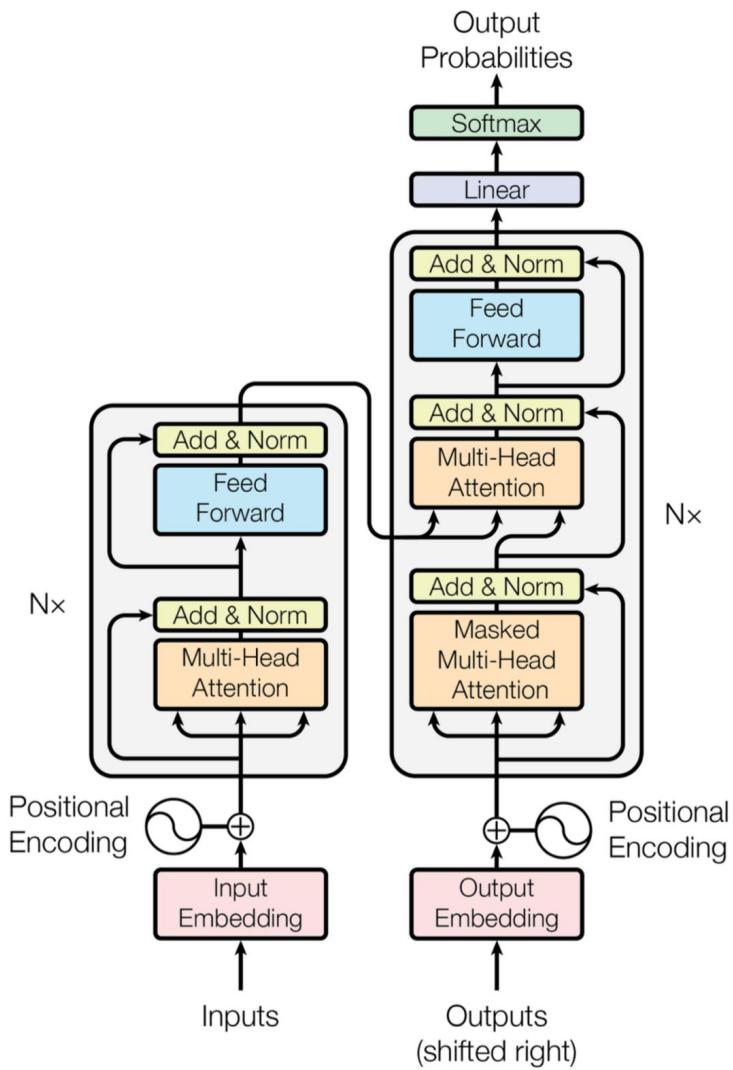


Positional Encoding

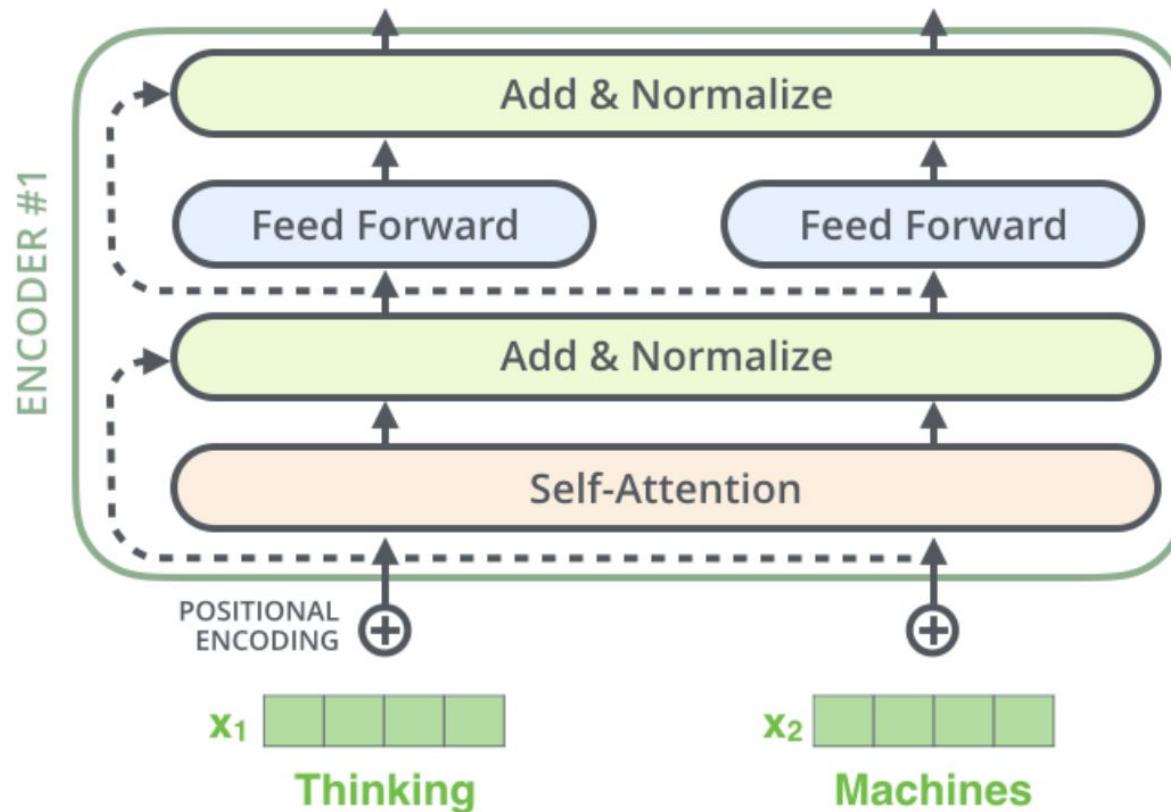


Layer Normalization

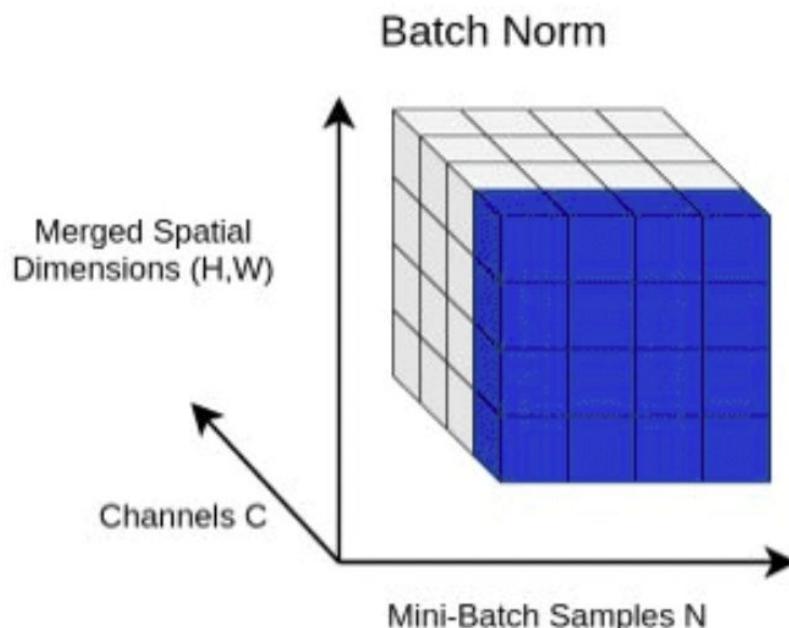
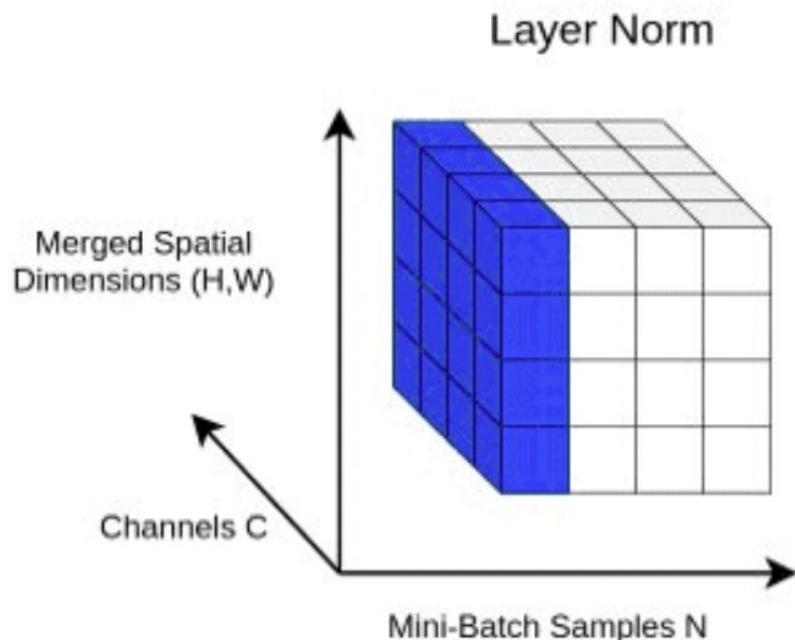
The Transformer: recap



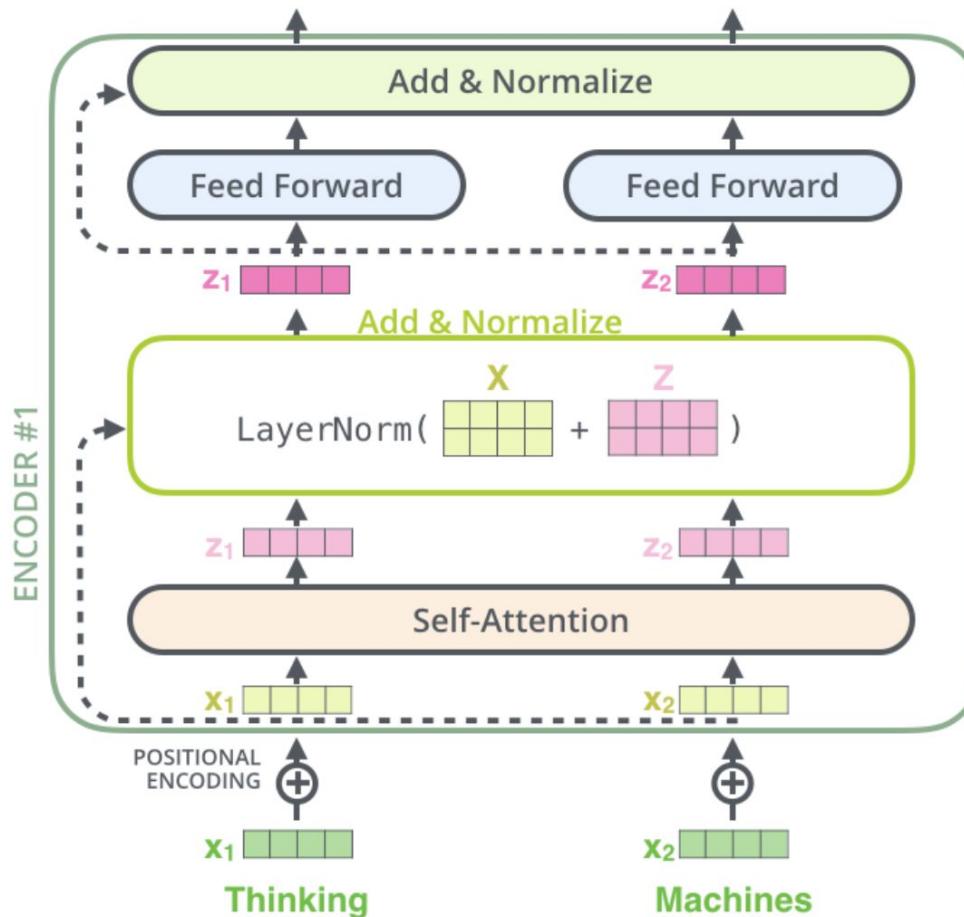
Layer Normalization



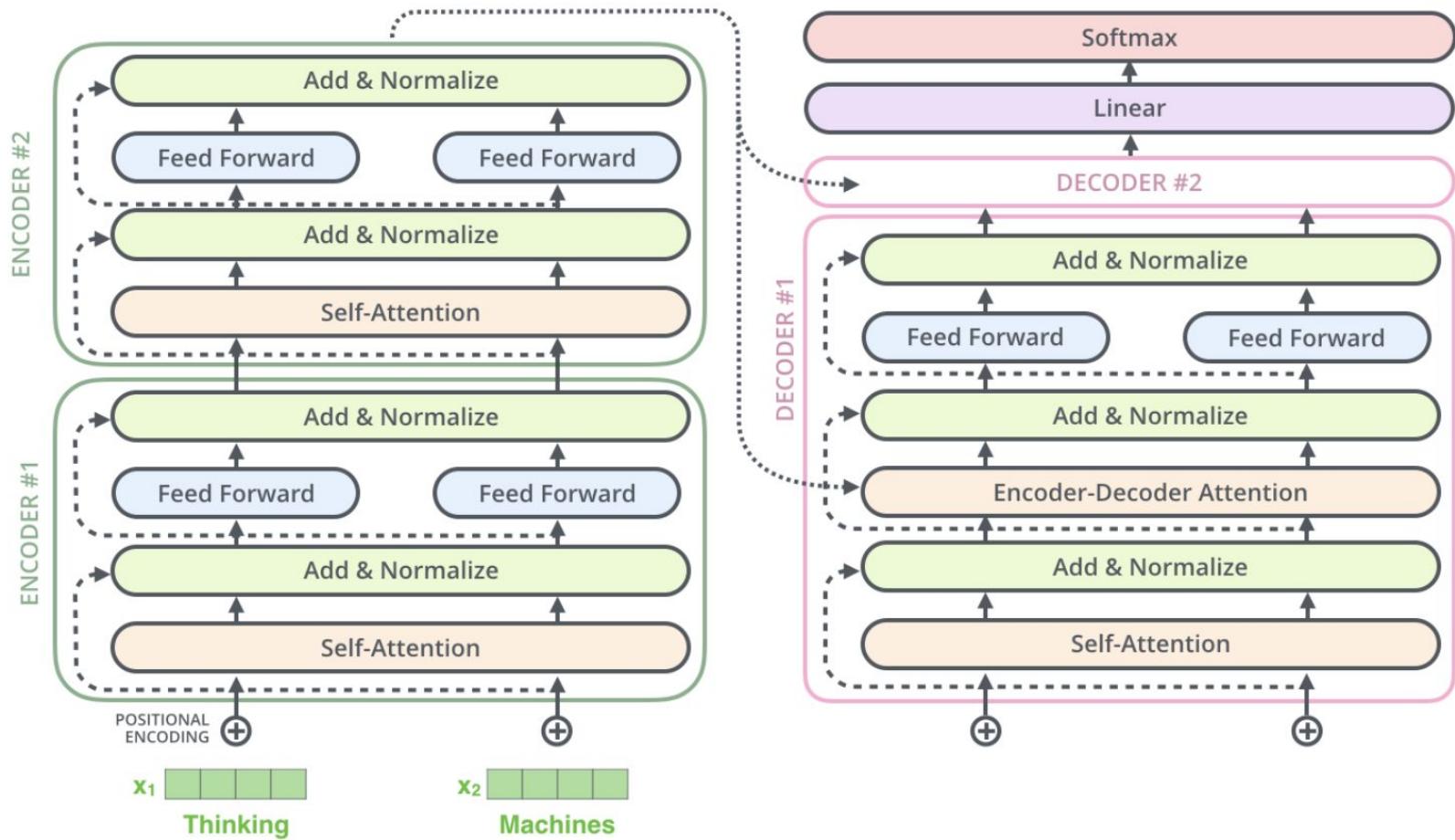
Layer Norm vs. Batch Norm



Layer Normalization

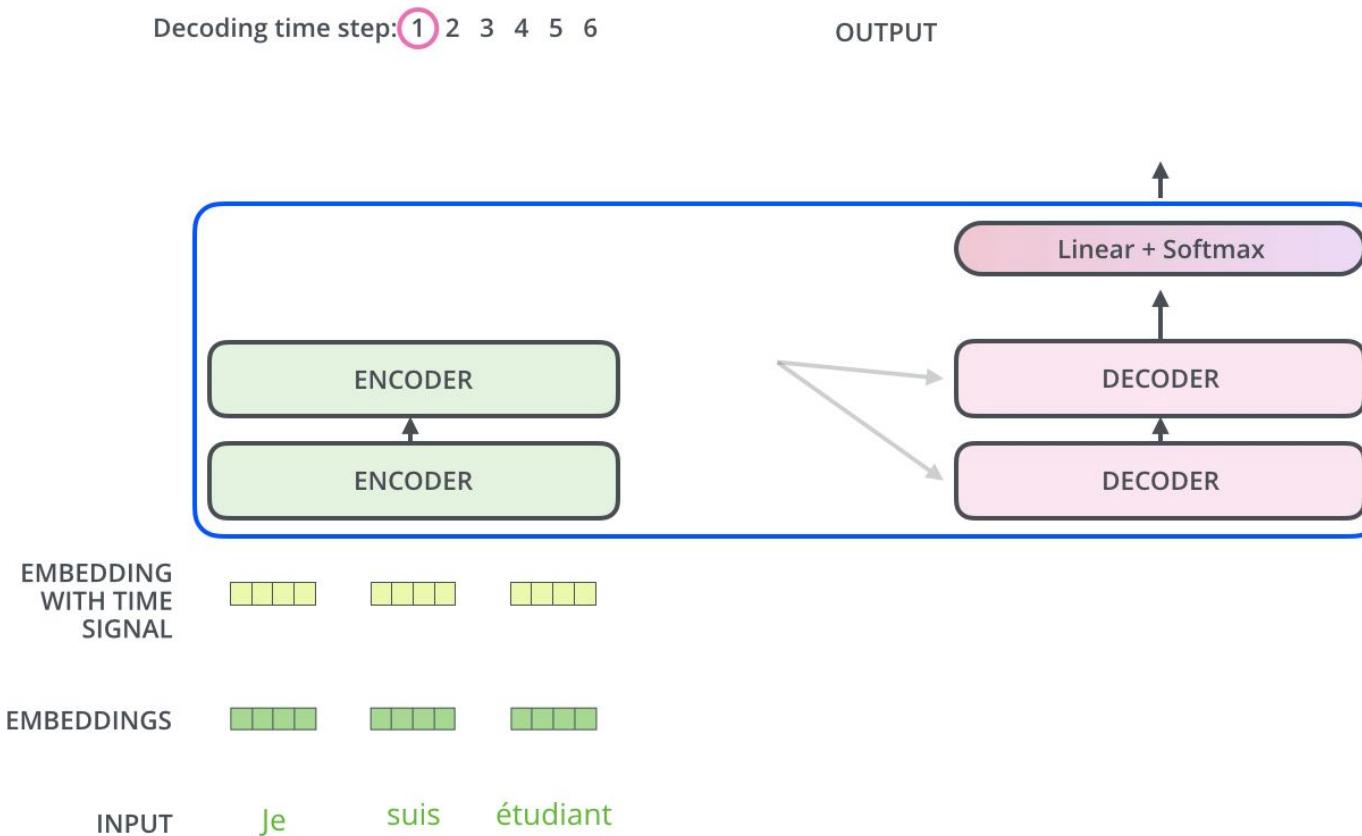


Layer Normalization

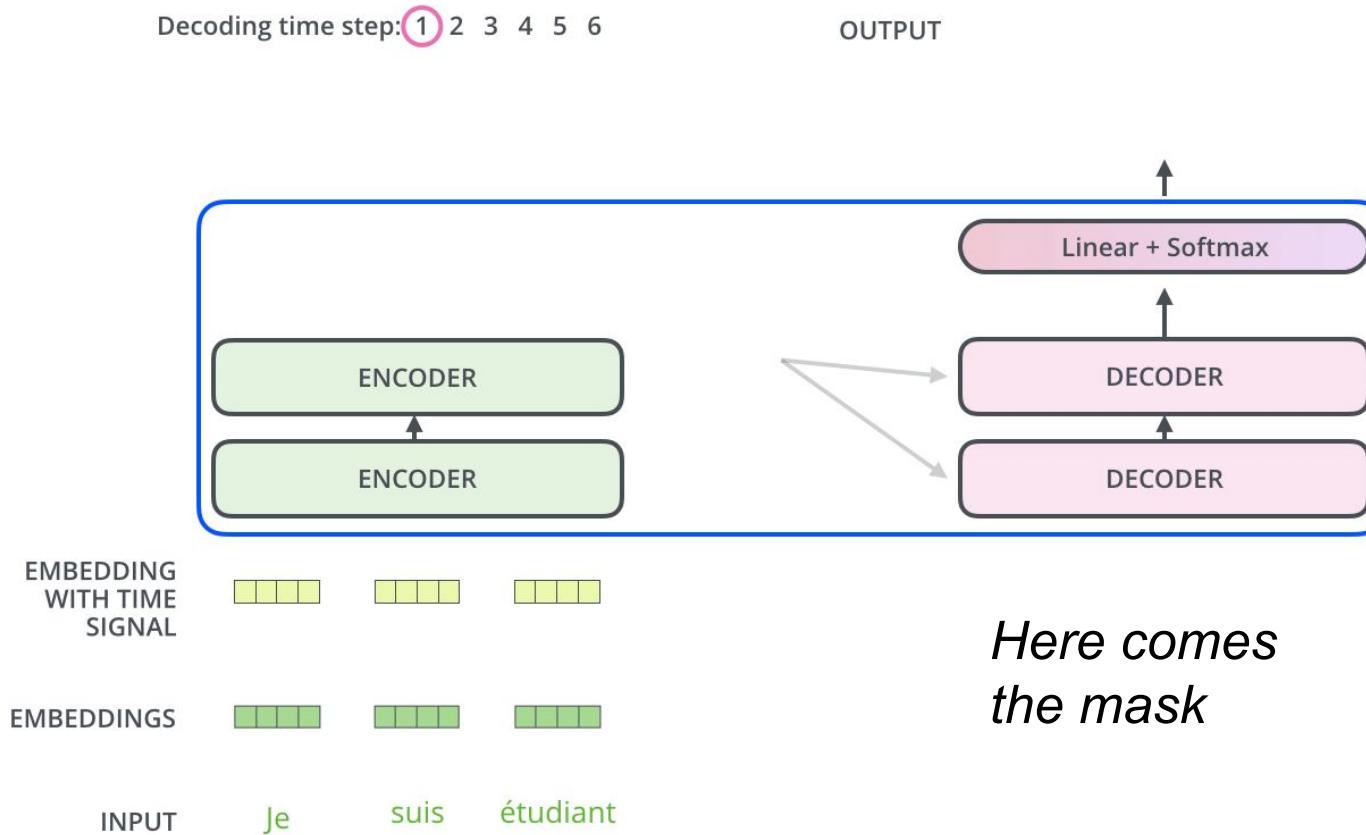


The Decoder

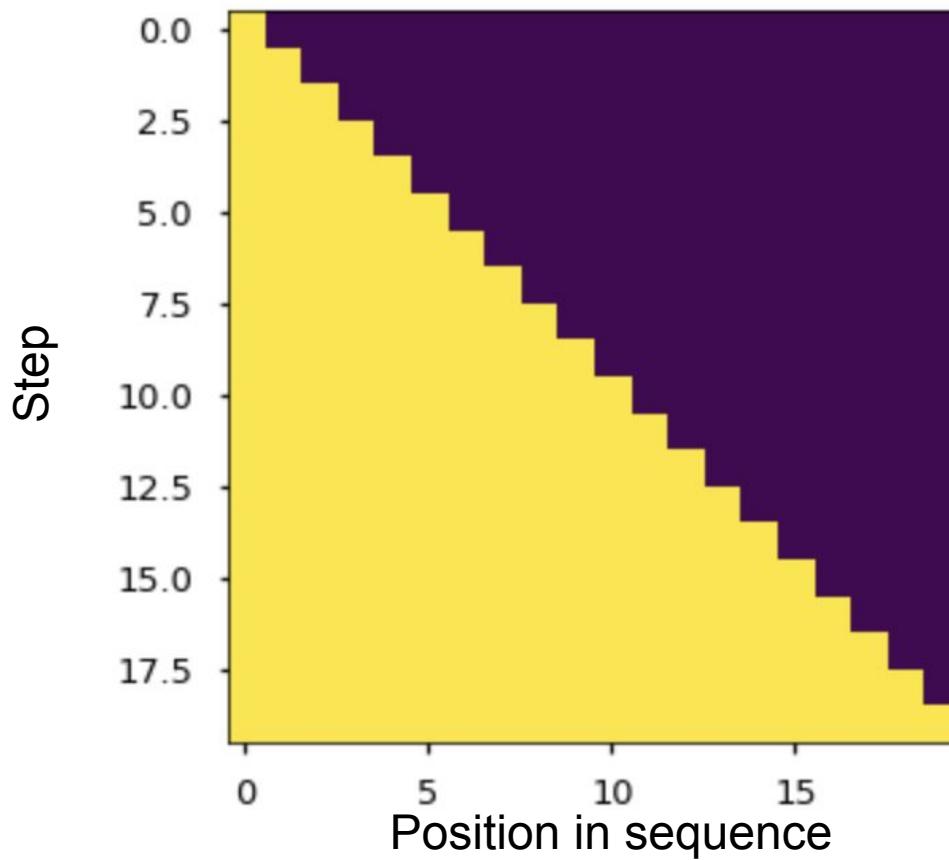
The Decoder Side



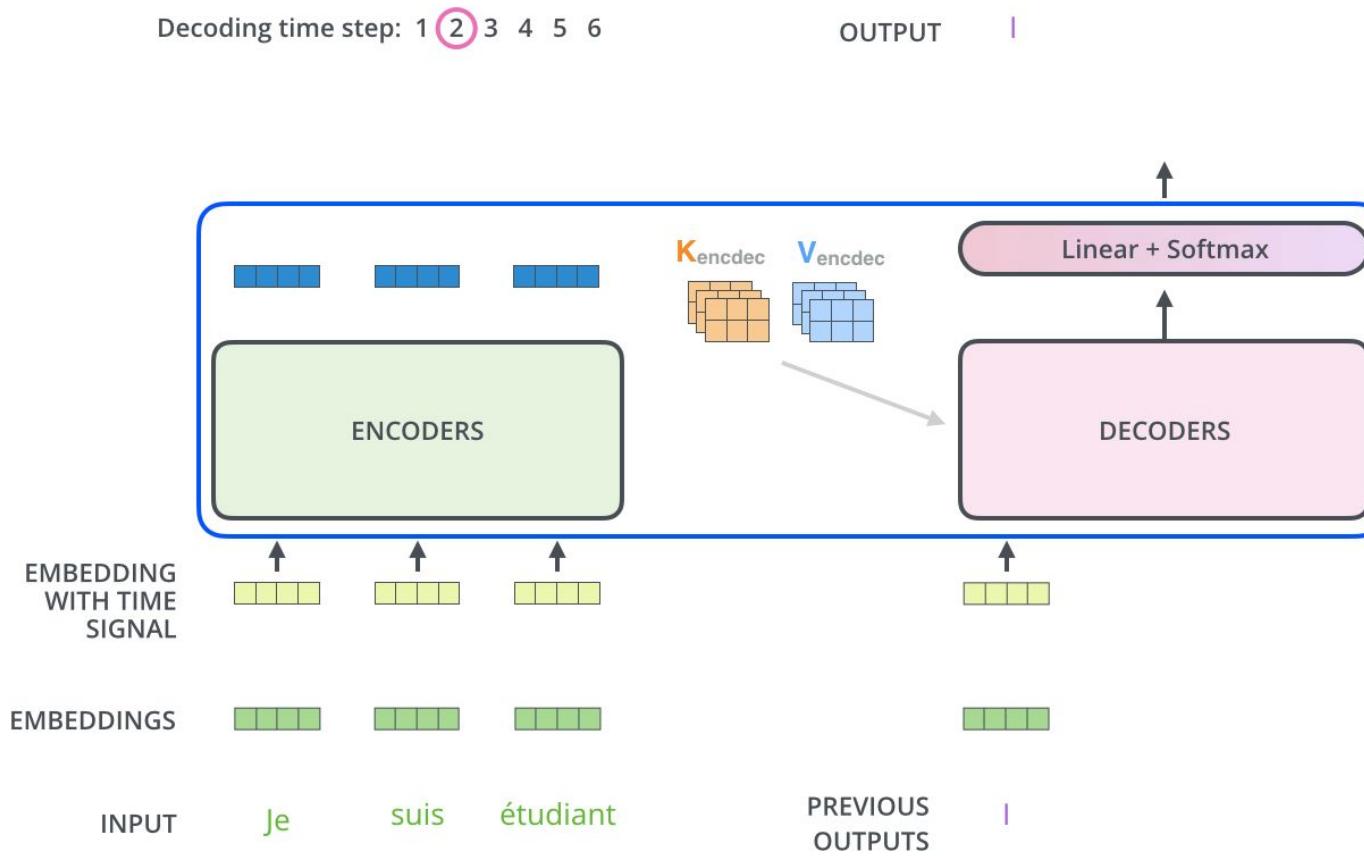
The Decoder Side



The masked decoder input



The Decoder Side



Final Linear and Softmax Layer

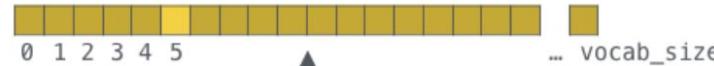
Which word in our vocabulary
is associated with this index?

am

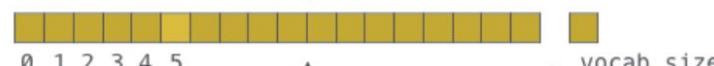
Get the index of the cell
with the highest value
(argmax)

5

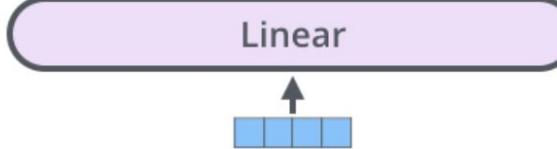
log_probs



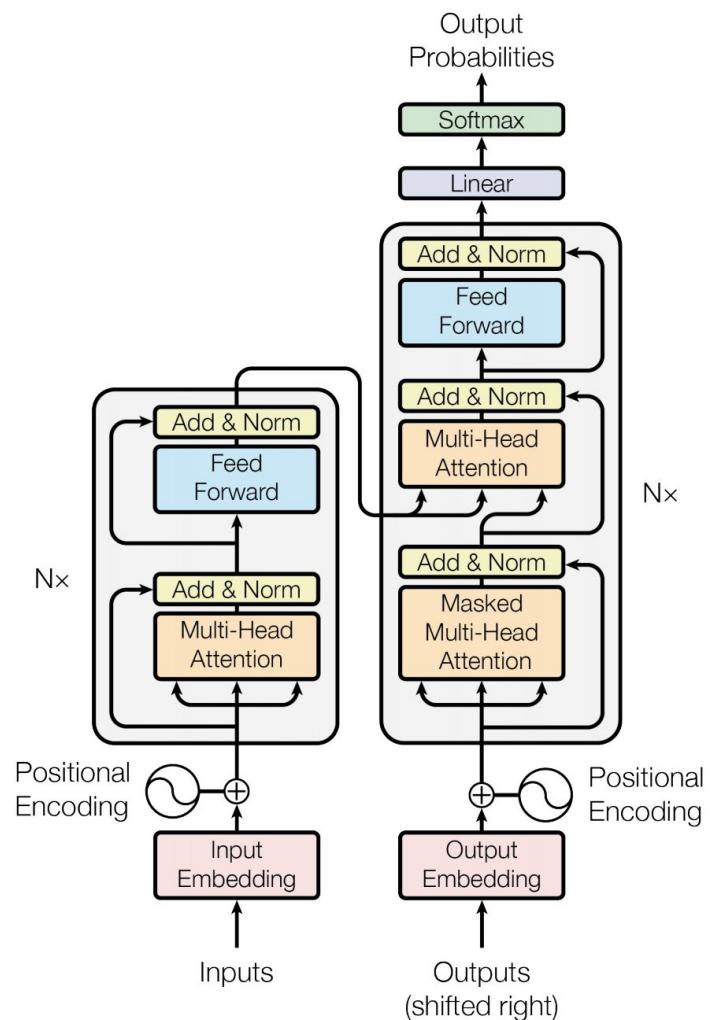
logits



Decoder stack output



The Transformer



- Transformer is a very powerful architecture based on self-attention mechanism
- Physical analogues can help you
- Further readings are available in the repo