Machine learning basic

# Lecture 12
# Self-attention
# Transformer

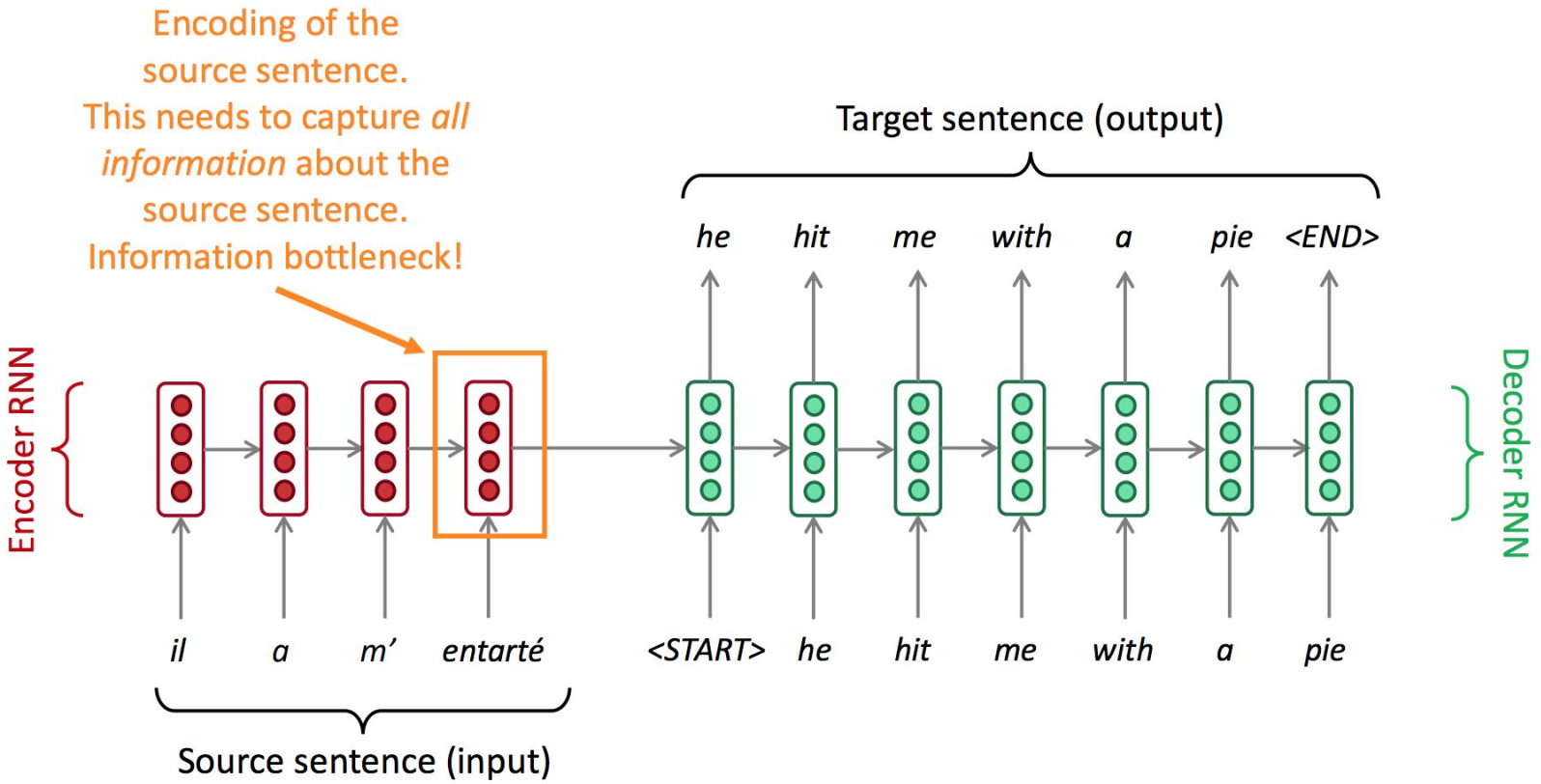
Vladislav Goncharenko

# Outline

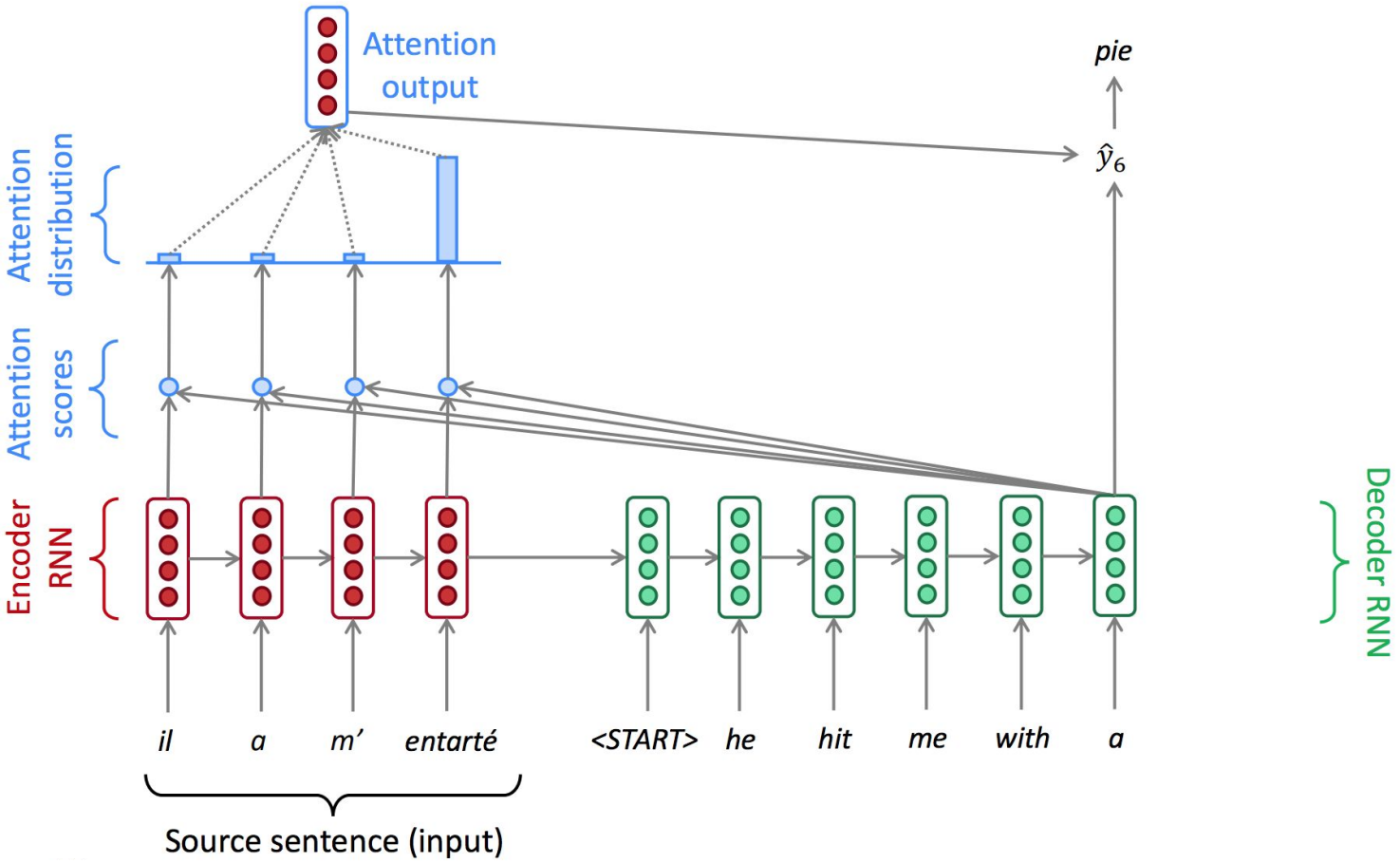
1. recap: Attention in seq2seq
2. CNN for textual data
3. Transformer architecture
4. Self-Attention
5. Positional encoding
6. Layer normalization

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

# recap: Attention in seq2seq



# recap: Attention in seq2seq



# recap: Attention in seq2seq

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

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

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

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

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

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

We use $\alpha^t$ 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

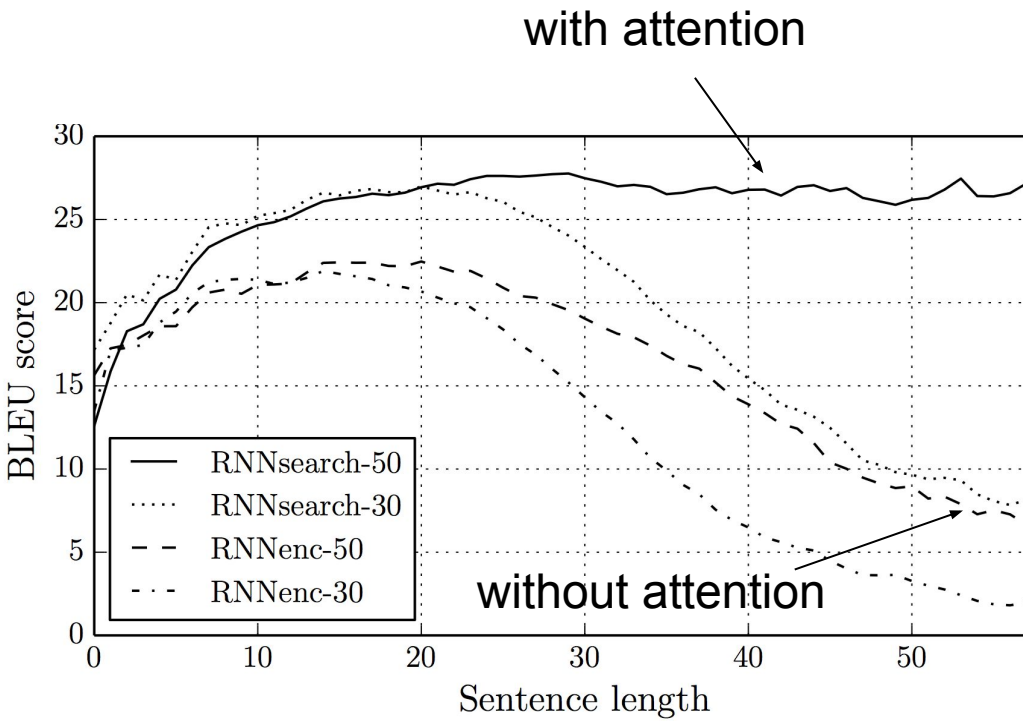
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

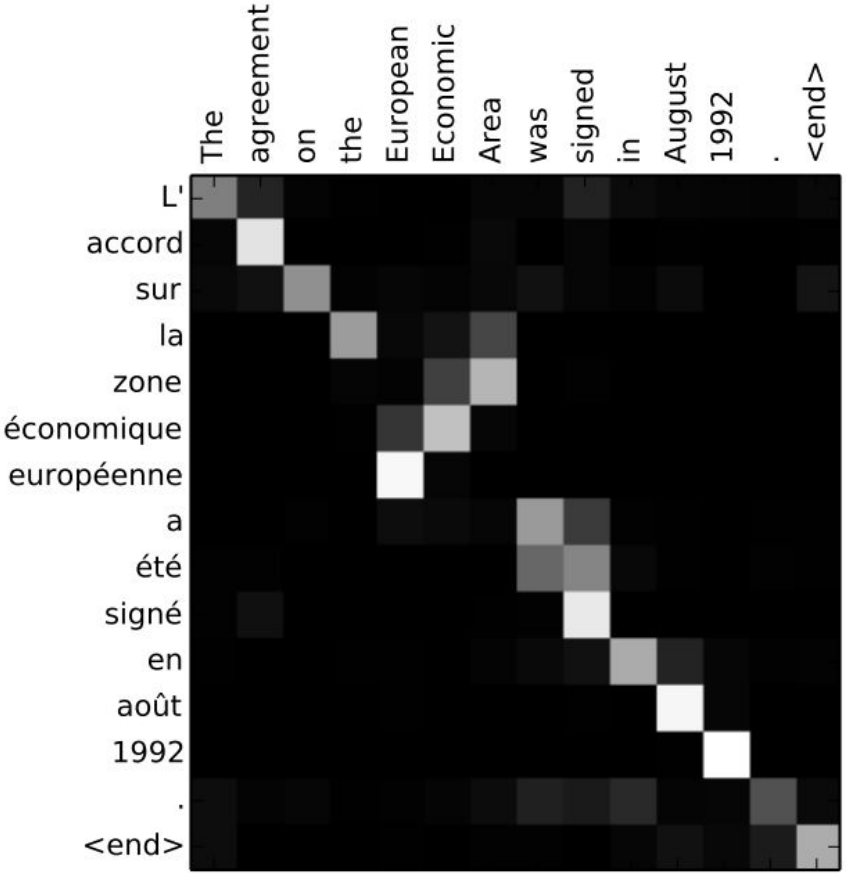
# Attention variants

- Basic dot-product (the one discussed before): $\boldsymbol{e}_i = \boldsymbol{s}^T \boldsymbol{h}_i \in \mathbb{R}$
- 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

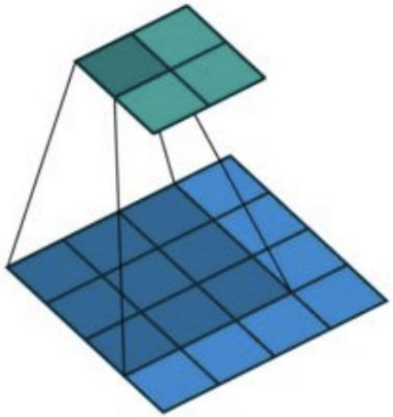
- “Free” word alignment
- Better results on long sequences



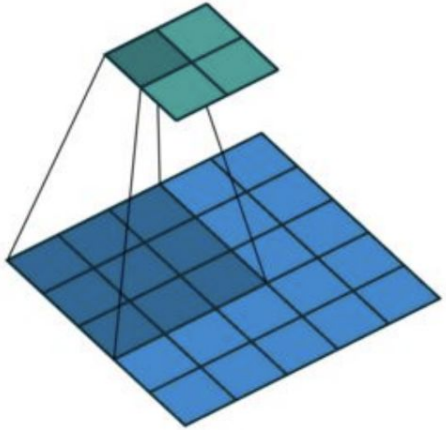



# CNN for textual data

# CNN recap

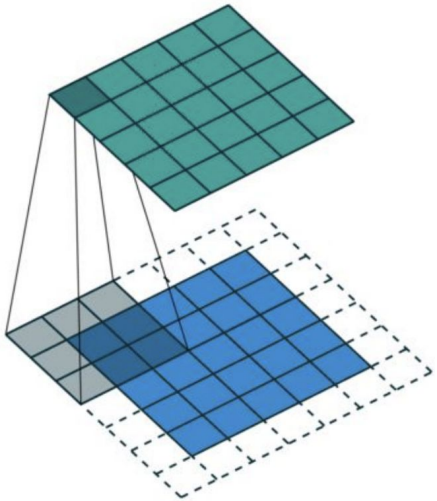


No padding,
no strides

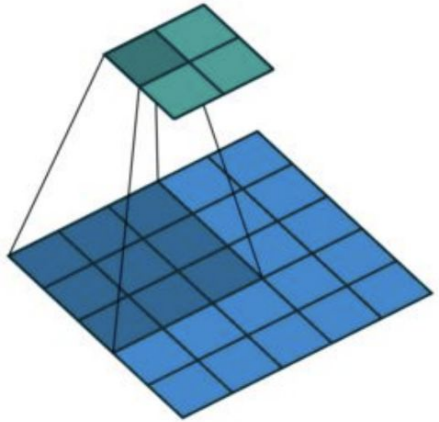


No padding,
with strides

# CNN recap

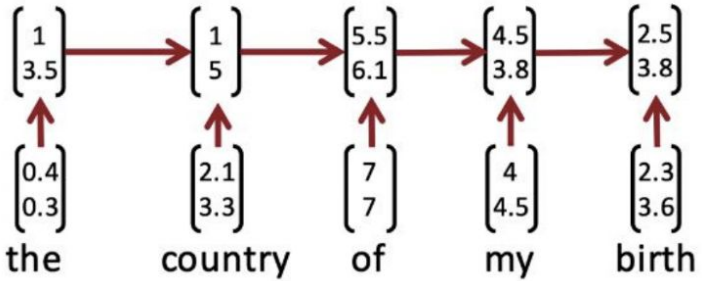


With padding,
no strides



No padding,
with strides

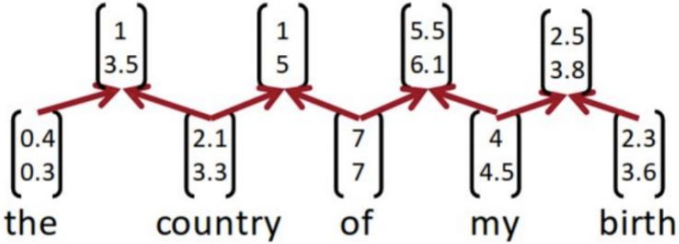
# From RNN to CNN




- 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

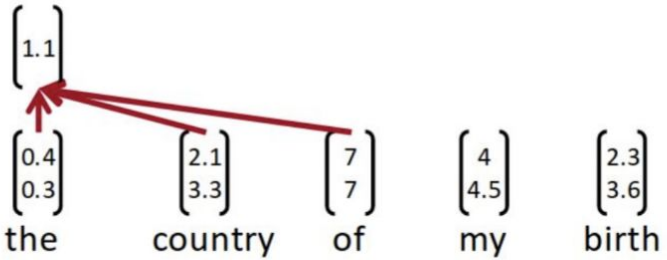
$$p = \tanh\left(W \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + b\right)$$

- Can be interpreted as convolution over the word vectors

# One layer CNN

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

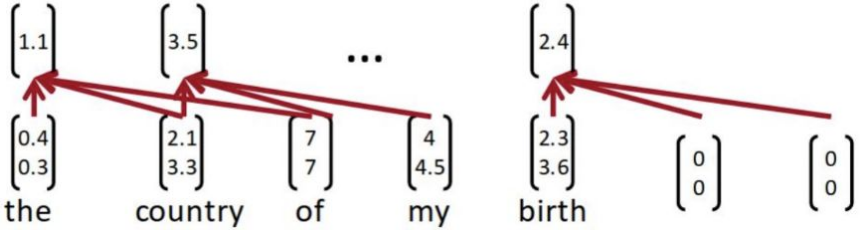
$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$



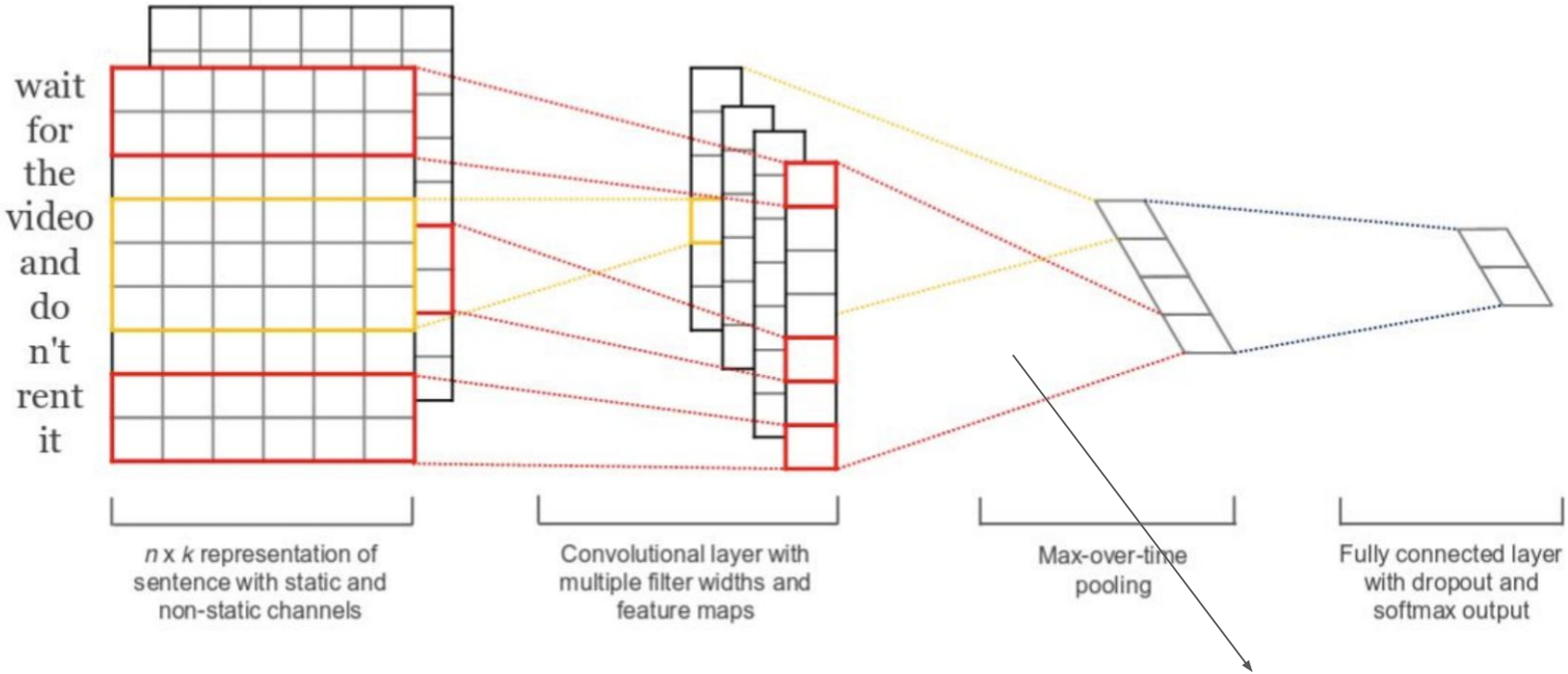

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

What's next?

We need more features!



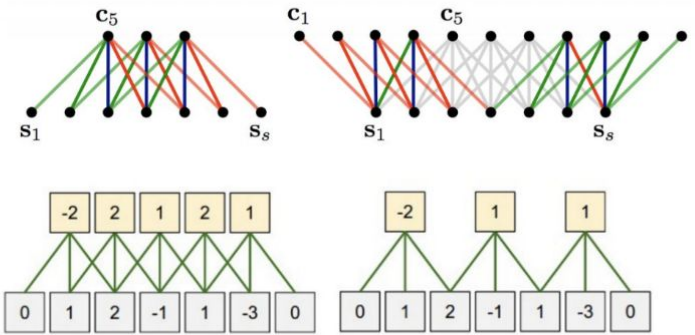
# CNN over text embeddings: general scheme



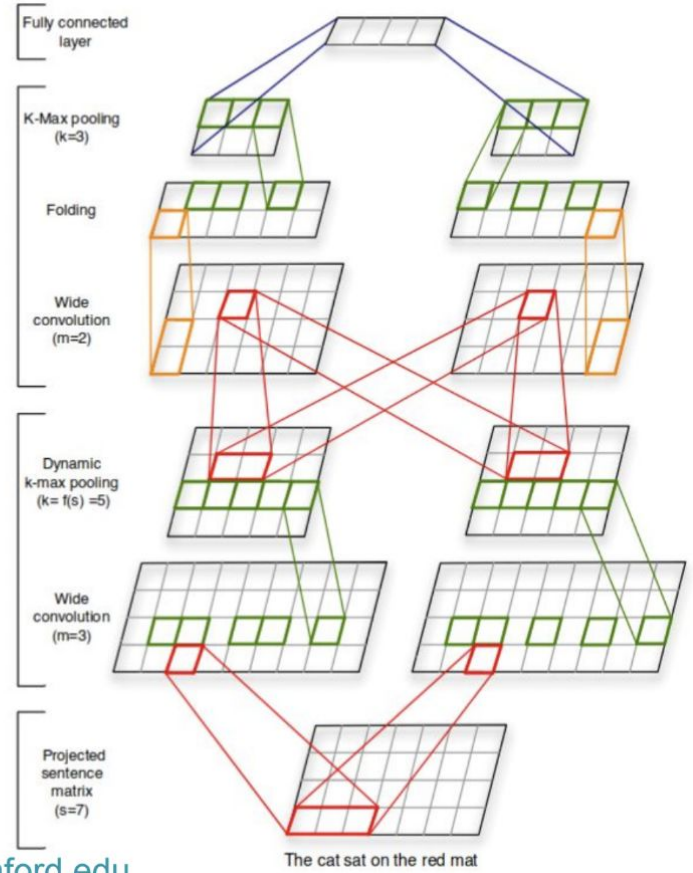

*Invariance w.r.t. sequence length*

# More about CNN

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



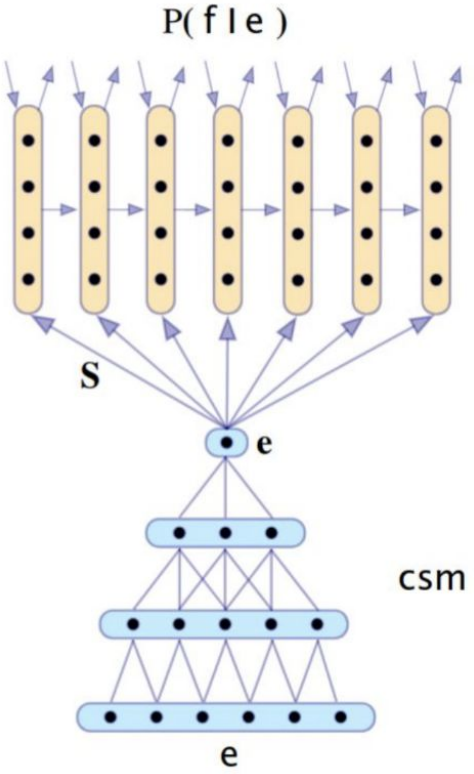

- Complex pooling schemes over sequences
- Great readings (e.g. Kalchbrenner et. al. 2014)




Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

# 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

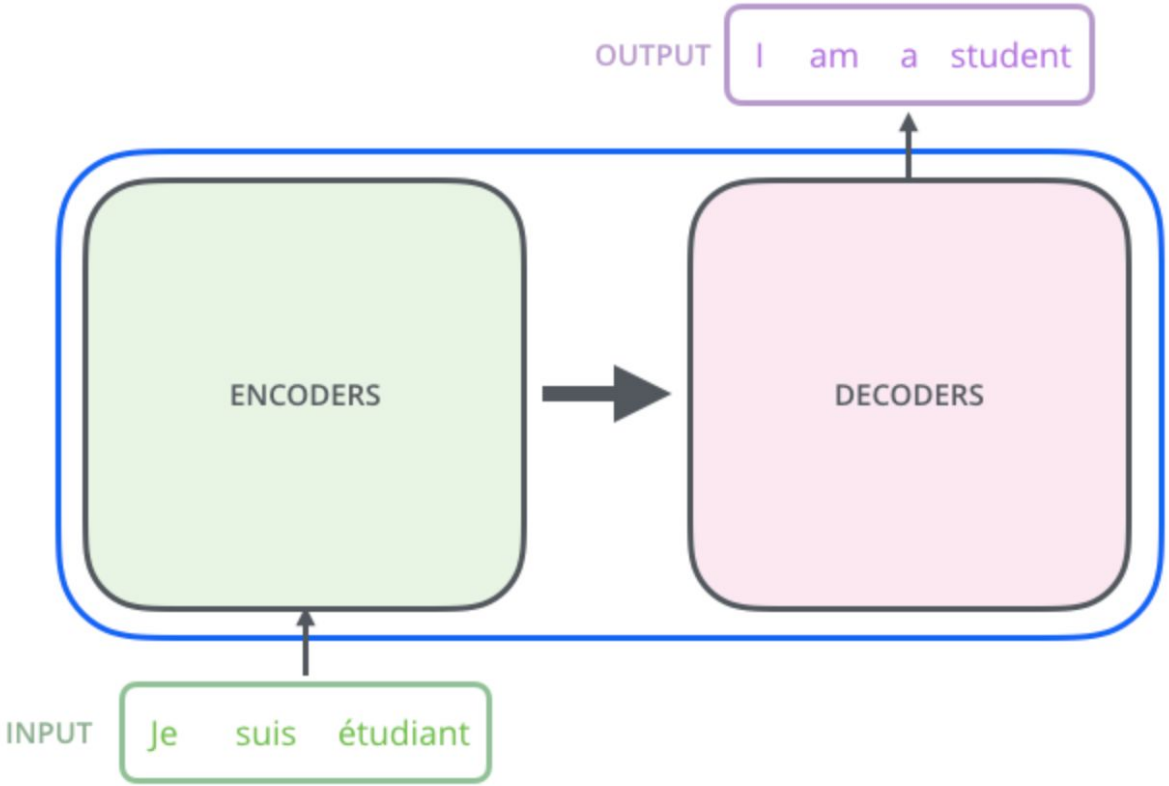



Based on: Lecture by Richard Socher 5/12/16, http://cs224d.stanford.edu

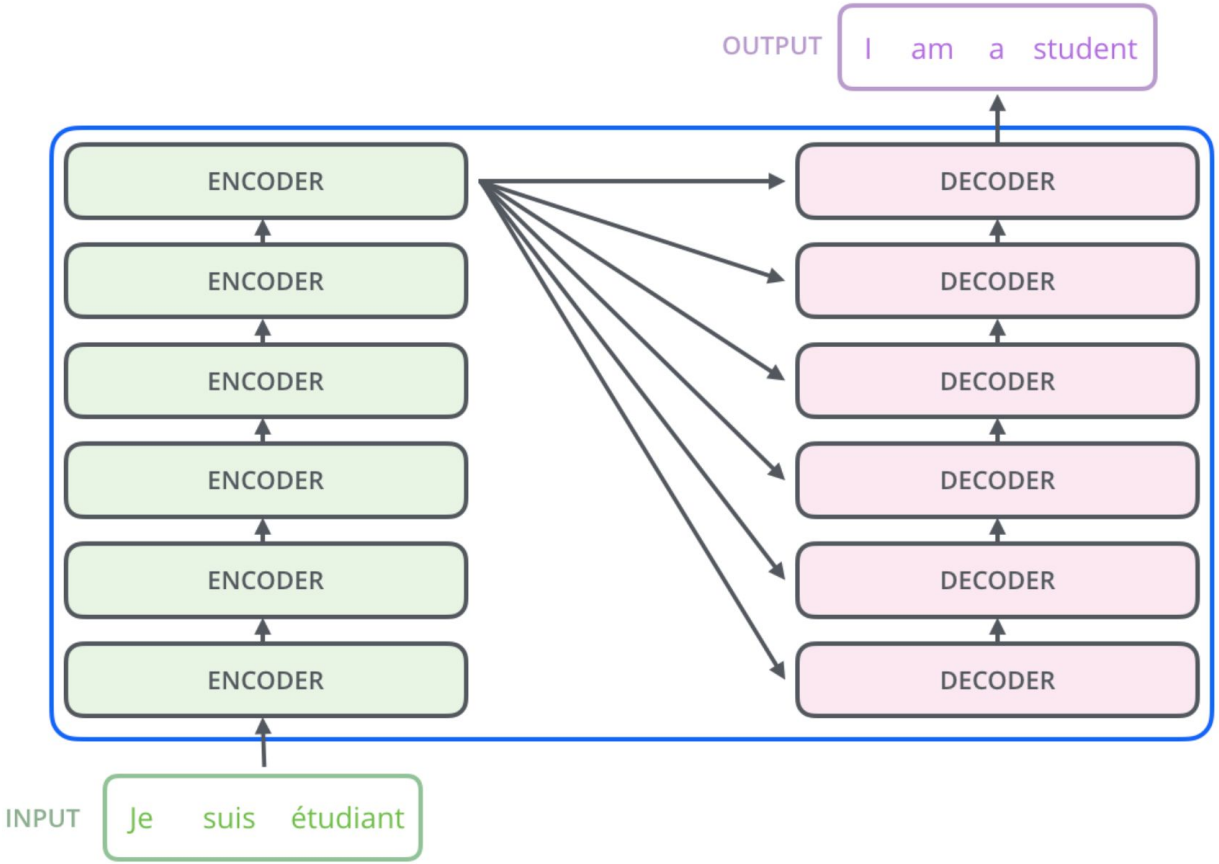
# The Transformer

# The Transformer

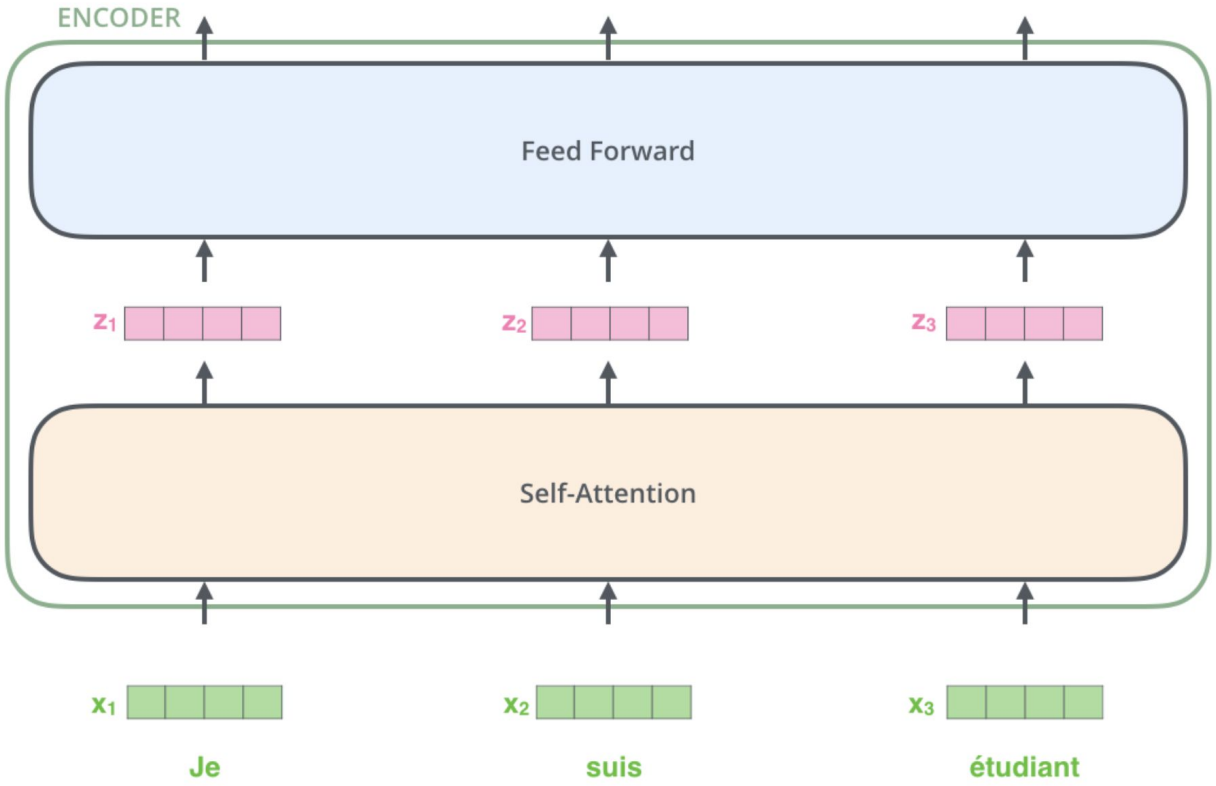
# The Transformer



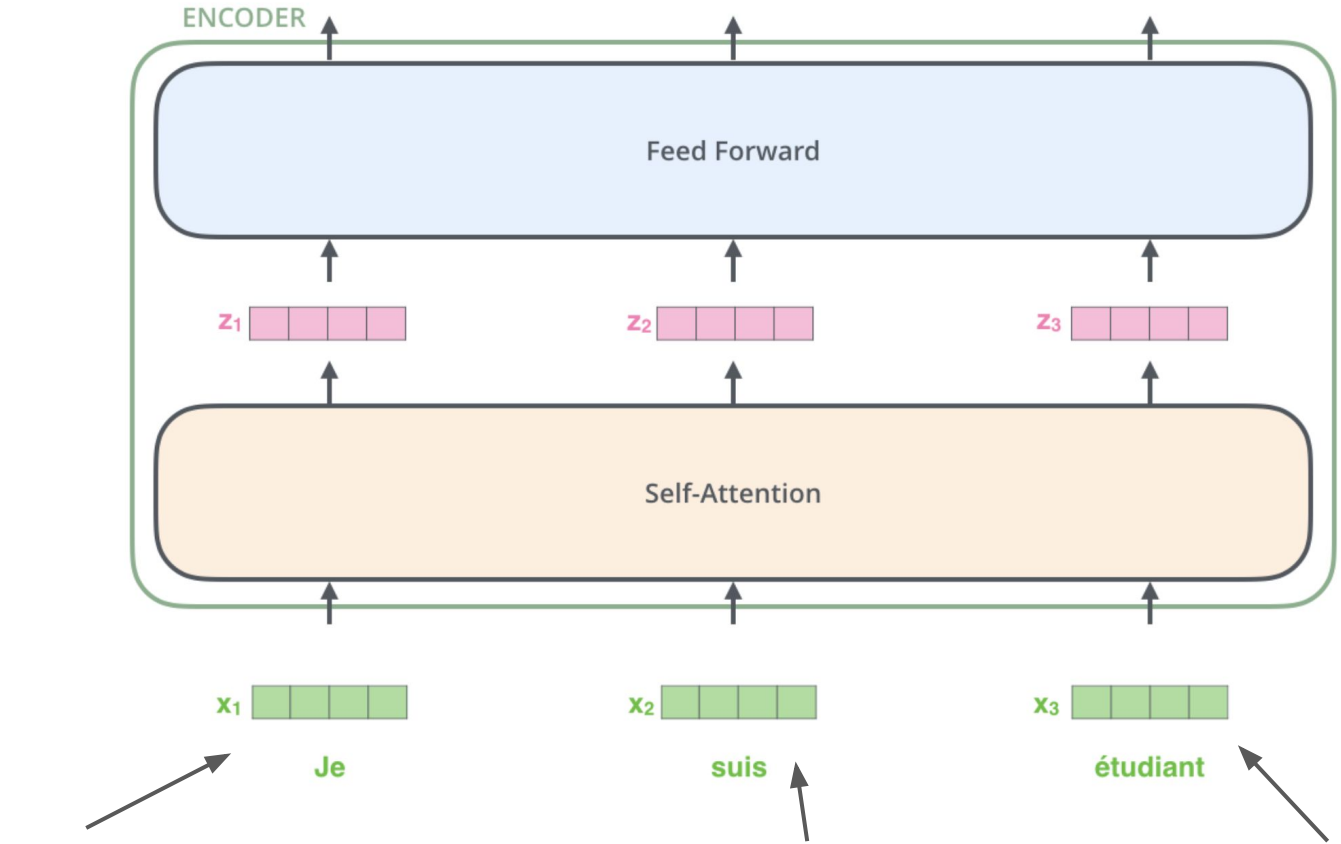
# The Transformer



# The Encoder Side



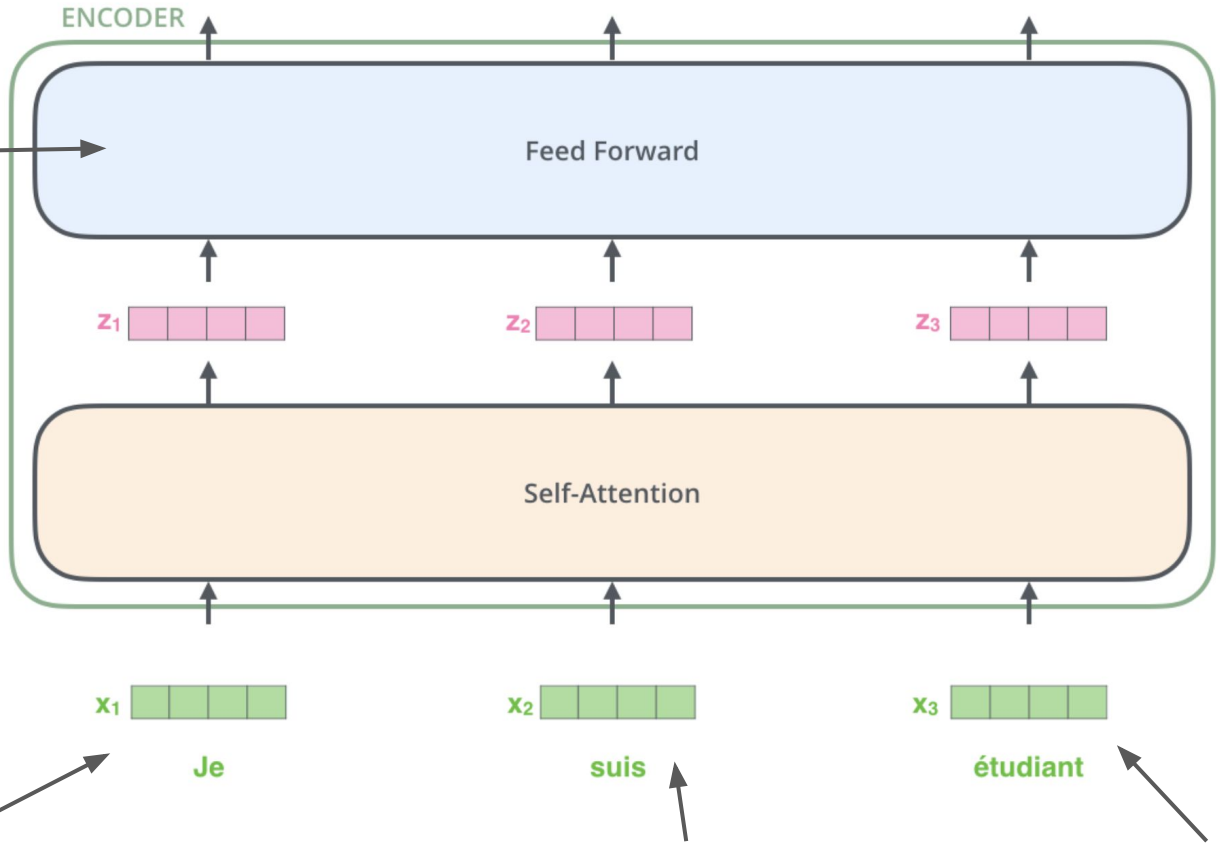
# The Encoder Side




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

# The Encoder Side

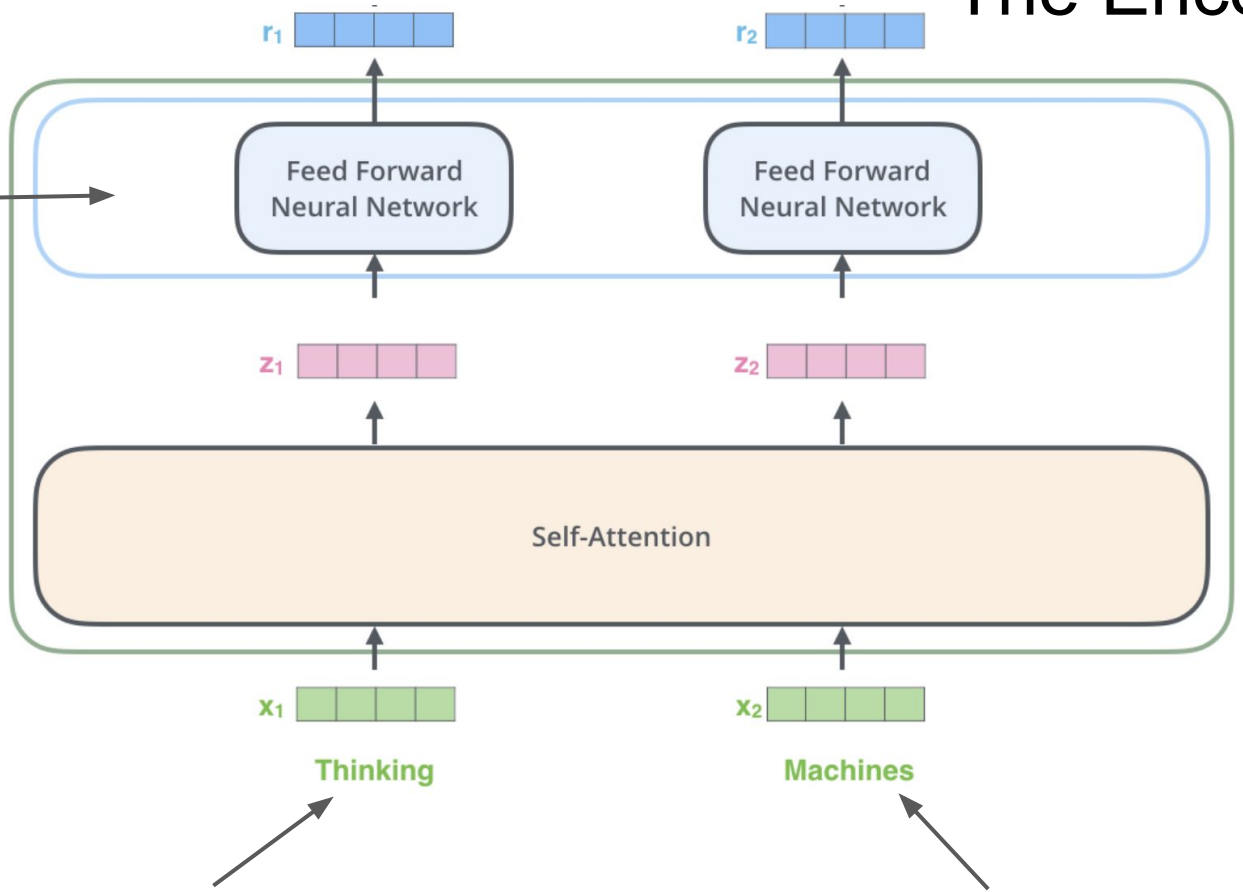
Can be parallelized




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

# The 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”

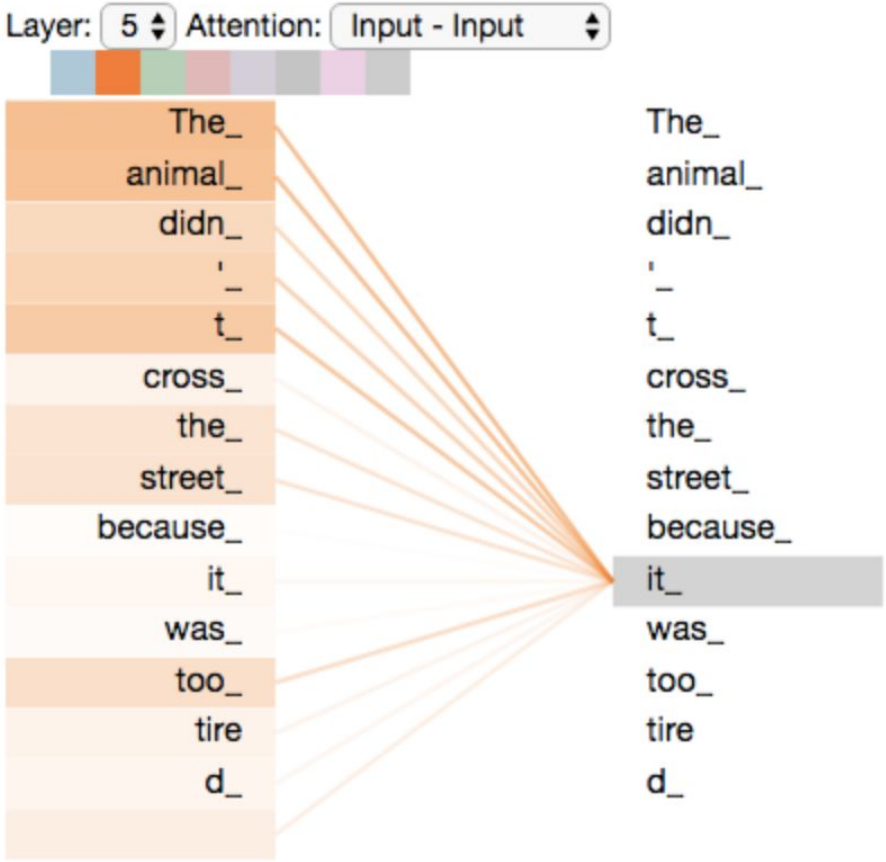
# Self-Attention at a High Level

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

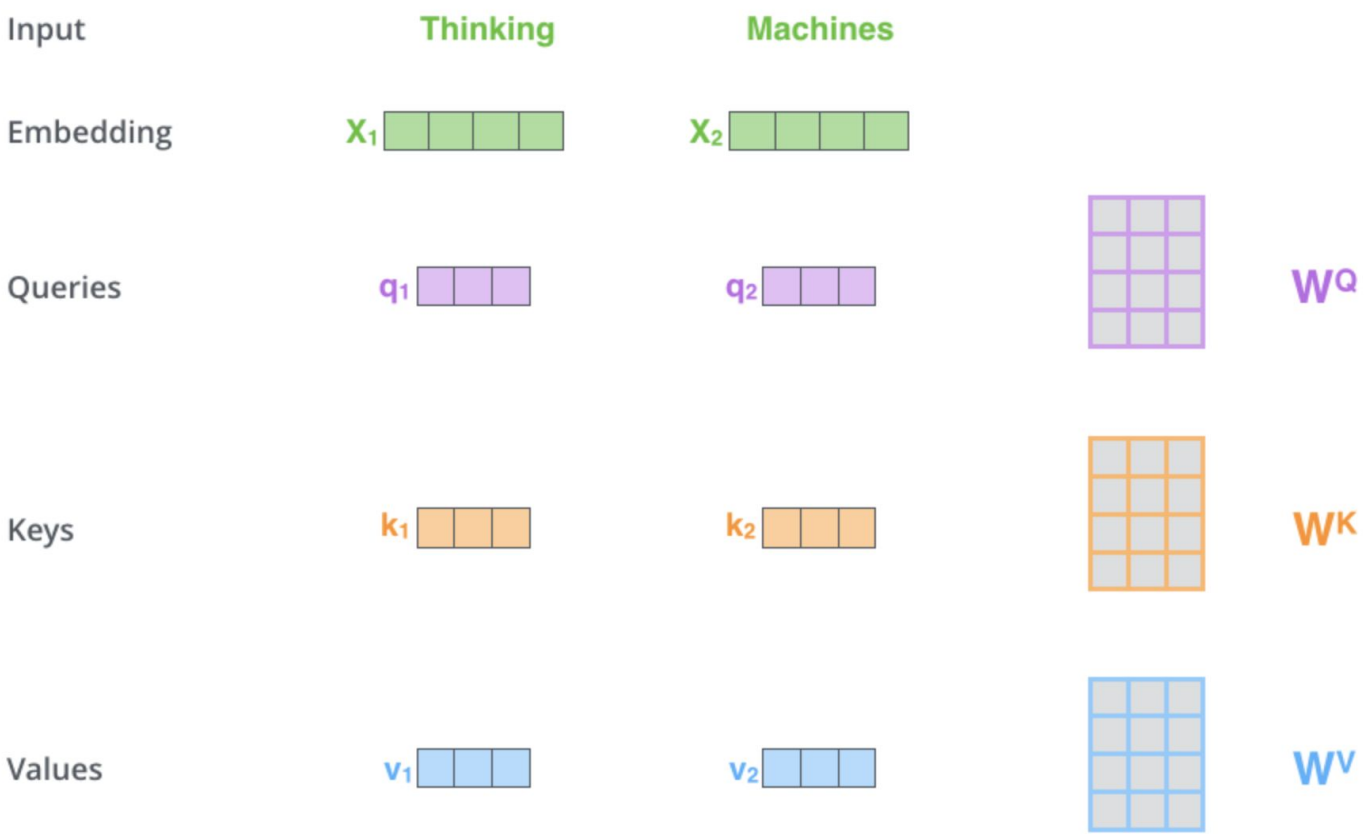
- What does “it” in this sentence refer to?
- We want self-attention to associate “it” with “animal”

- 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



# Self-Attention: detailed explanation

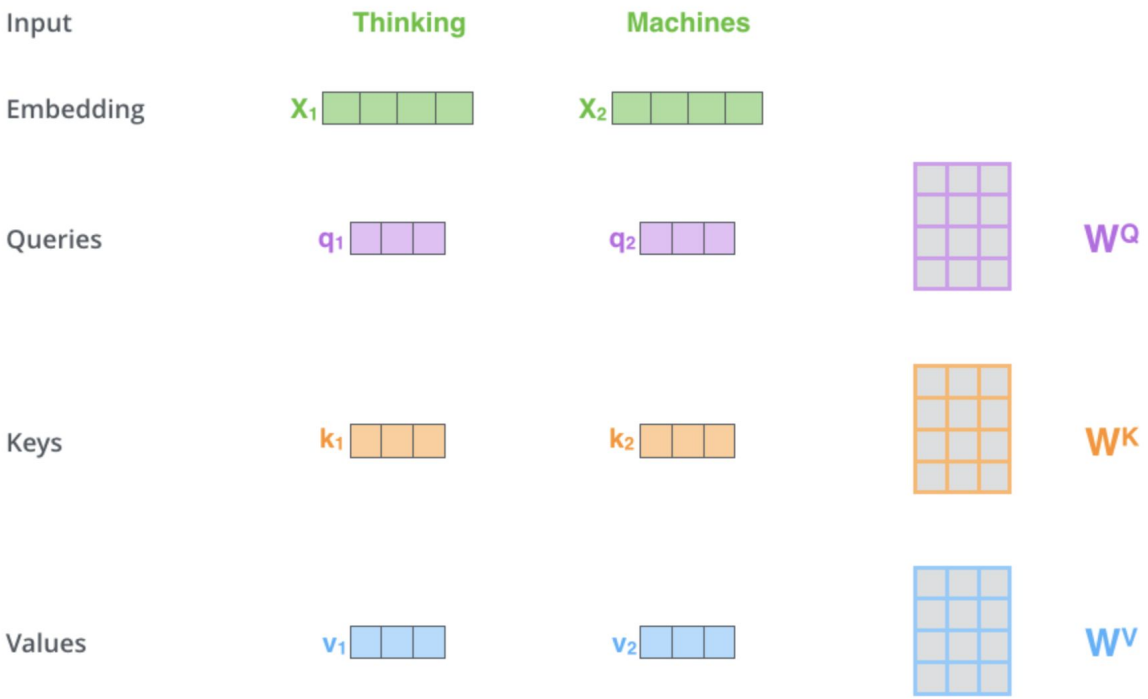


# Self-Attention: detailed explanation

**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”, and “value” vectors?

# Self-Attention: detailed explanation

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

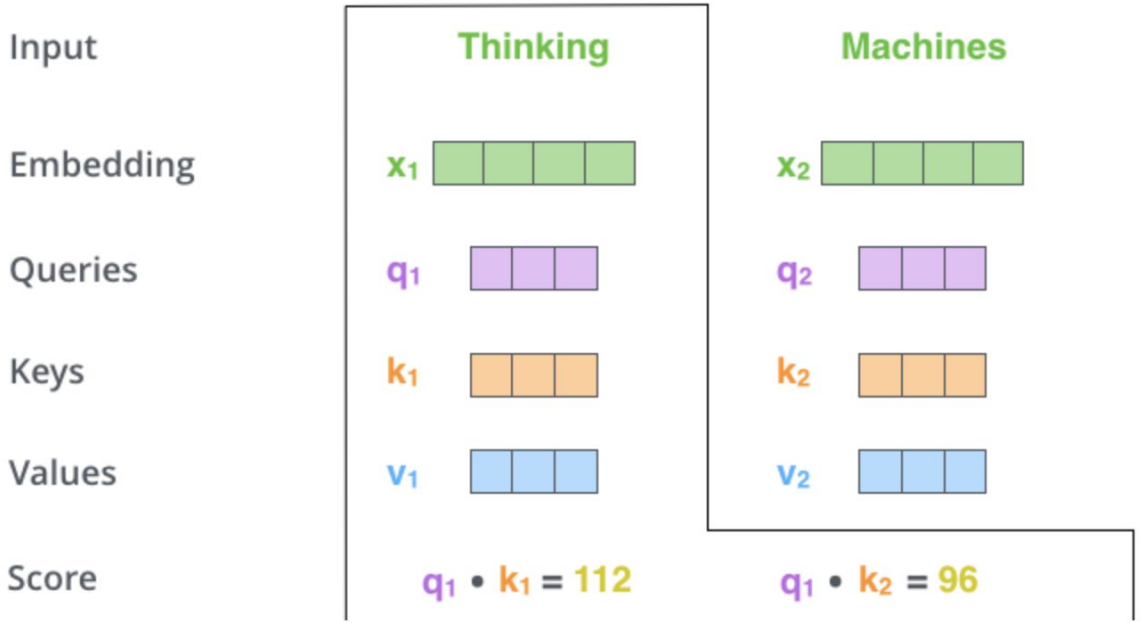
They’re abstractions that are useful for calculating and thinking about attention.

# Self-Attention: detailed explanation

**STEP 2:**

calculate a score

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



# Self-Attention: detailed explanation

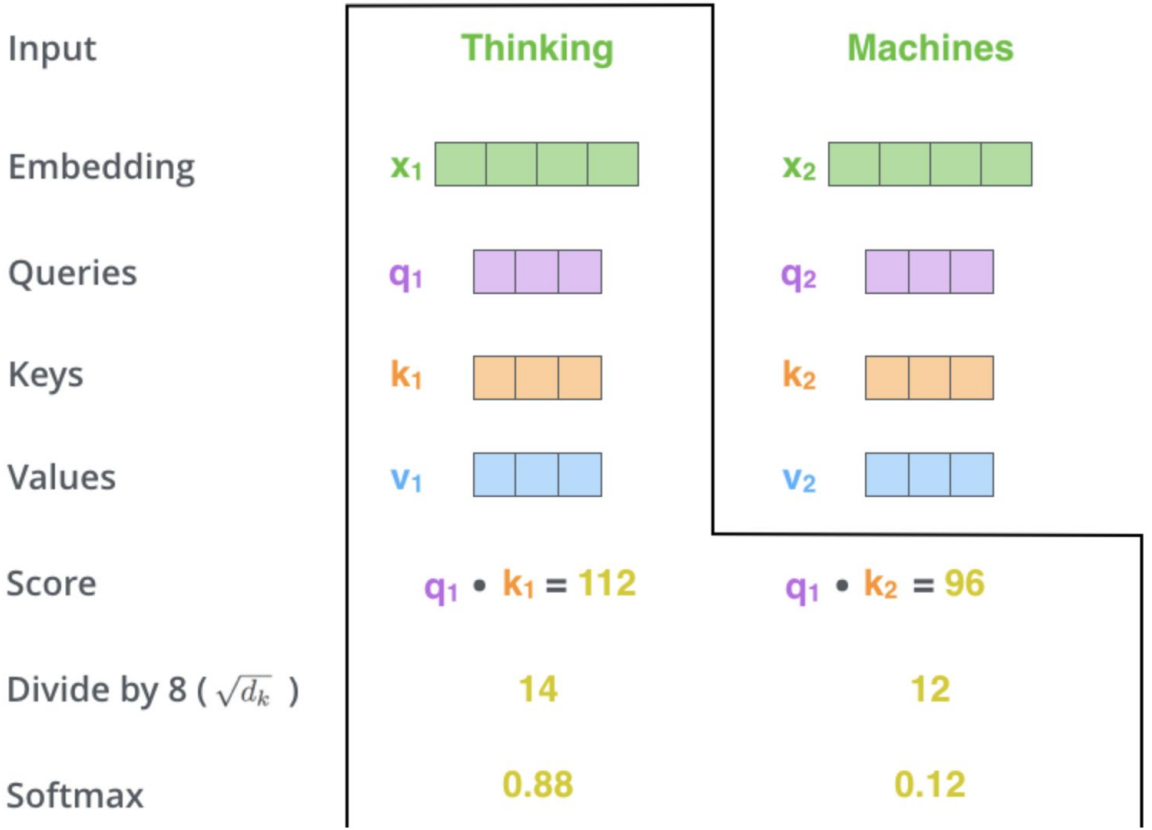
**STEP 3:**

divide the scores by 8

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

**STEP 4:**

softmax



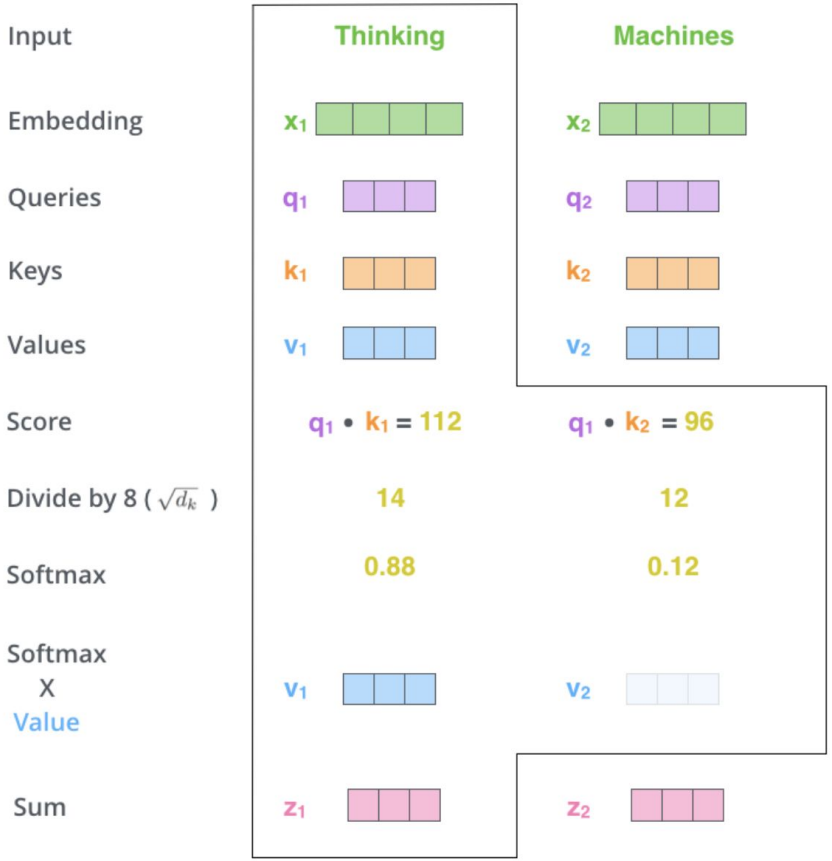
# Self-Attention: detailed explanation

**STEP 5:**

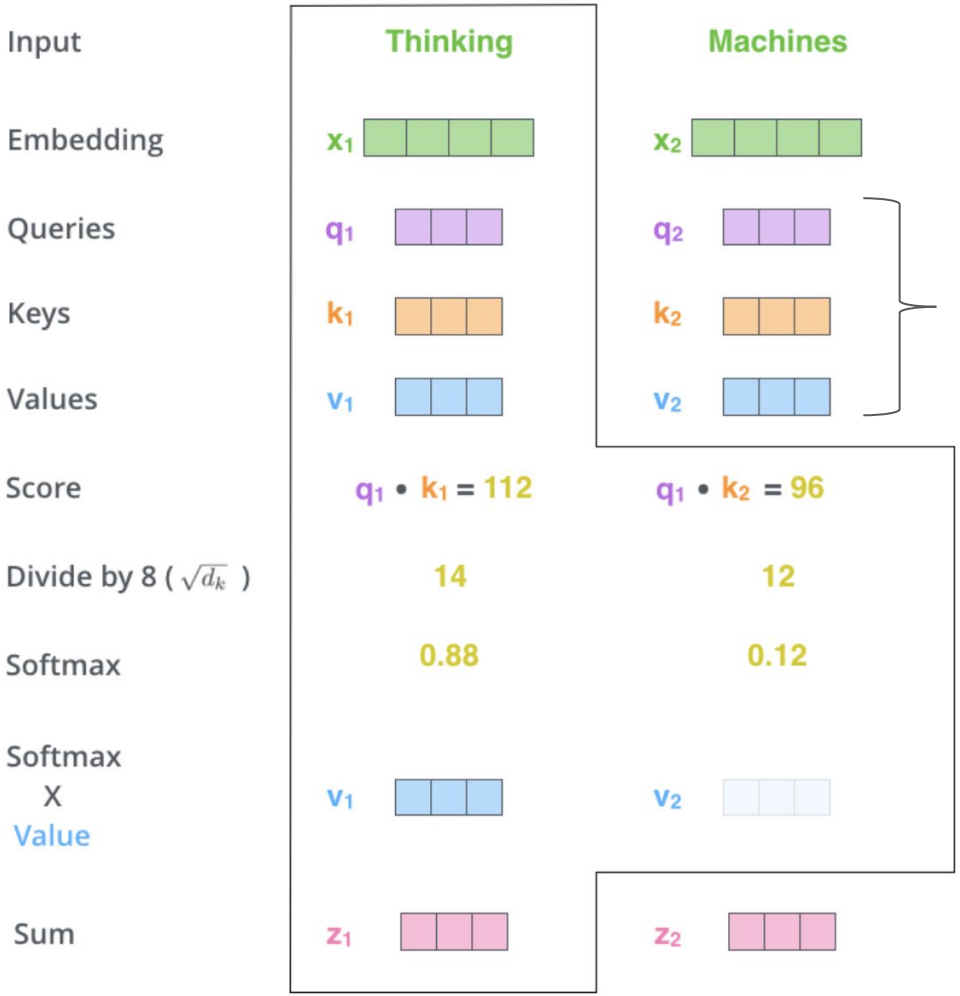
multiply each value vector by the softmax score

**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

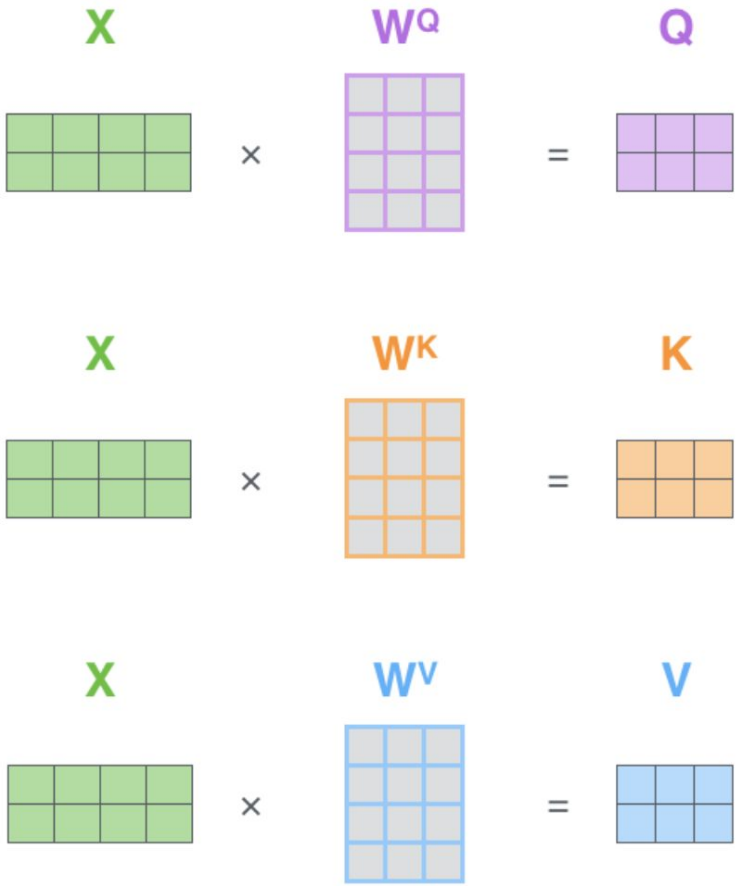
**STEP 5:** multiply each value vector by the softmax score

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

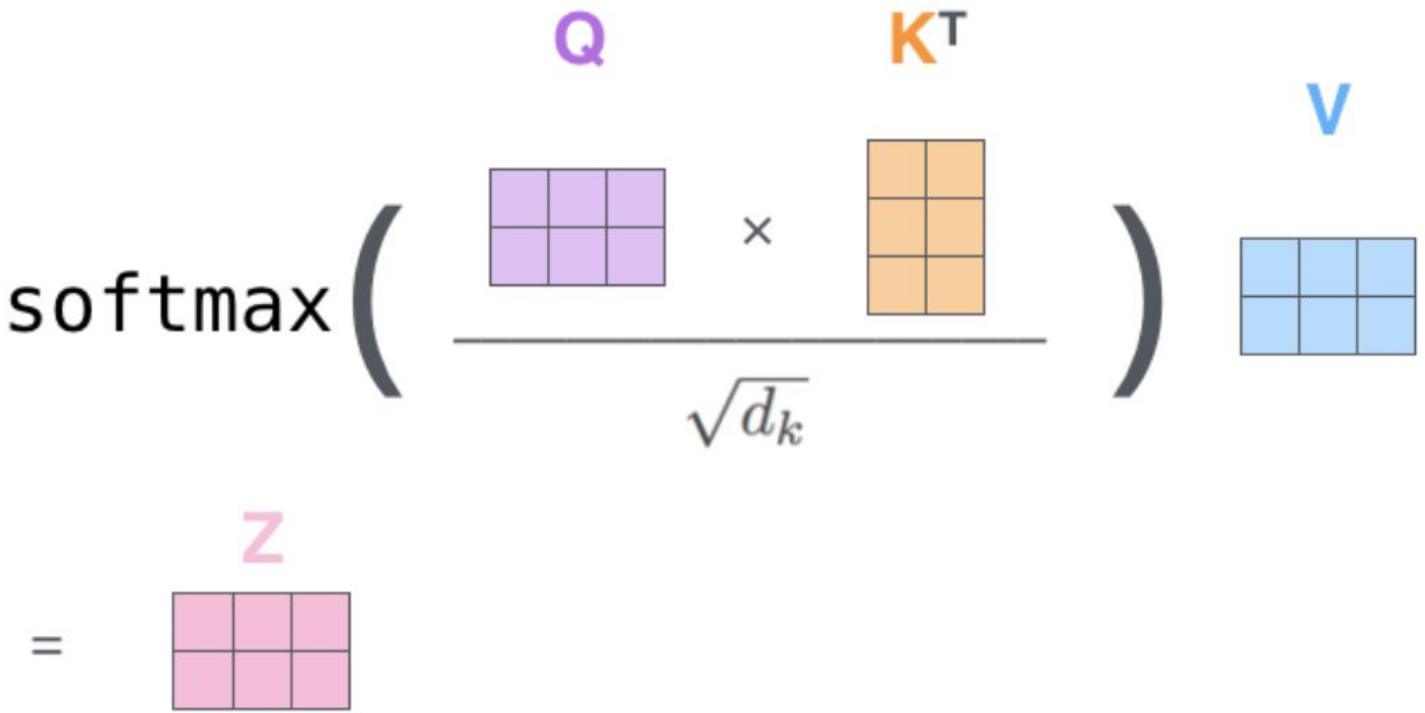
# Self-Attention: Matrix Calculation

Pack embeddings into matrix **X**

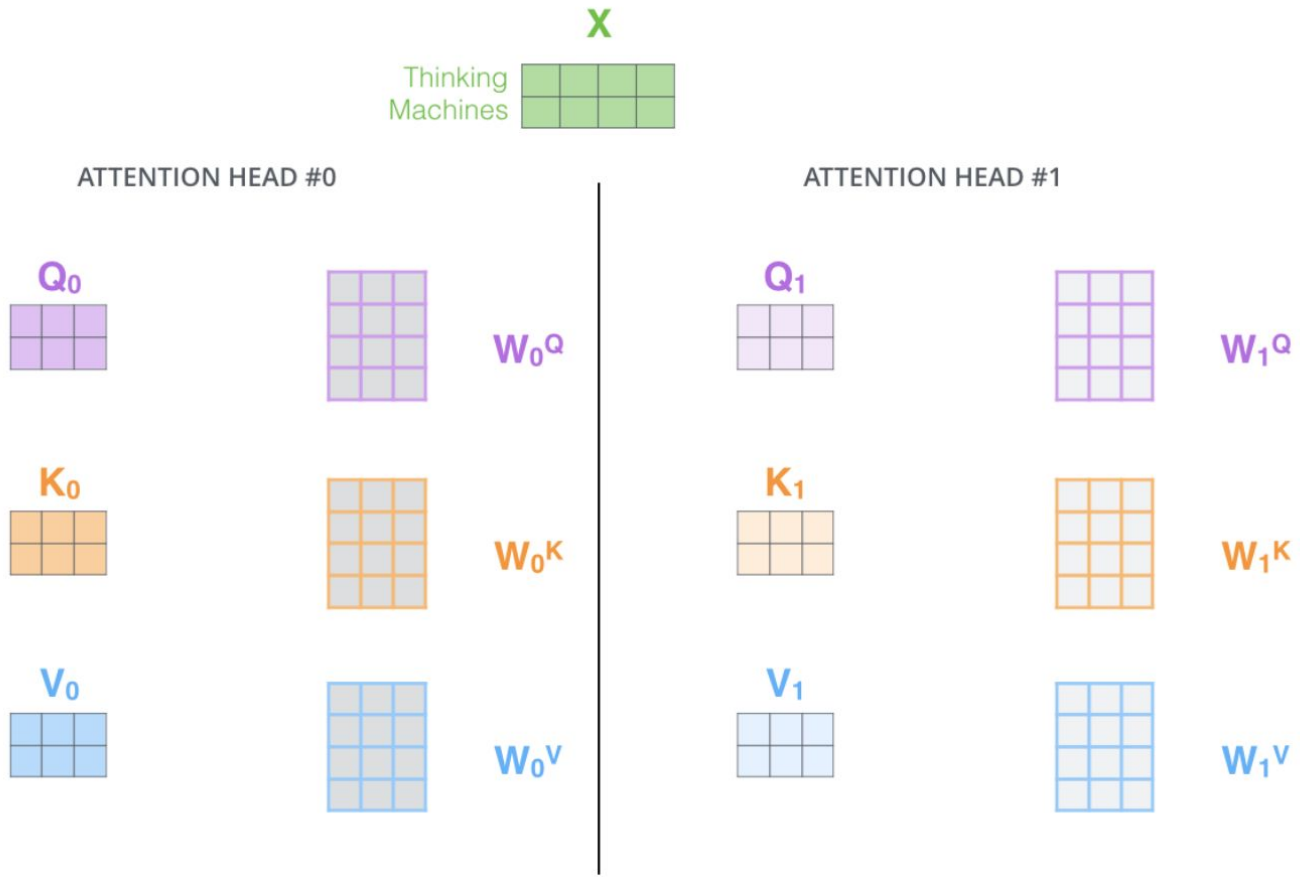
Multiply **X** by weight matrices we've trained (**Wk, Wq, Wv**)



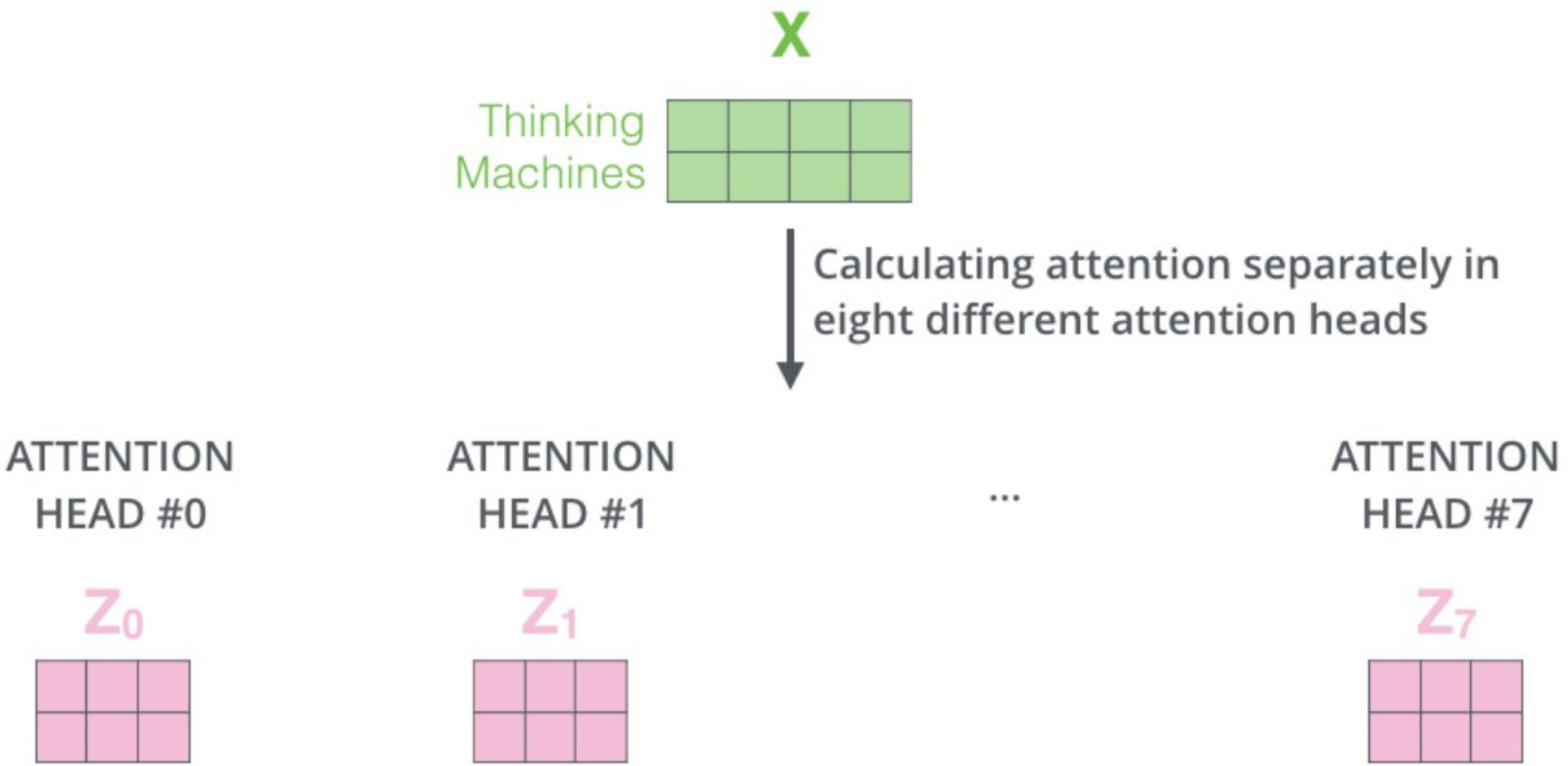
# Self-Attention: Matrix Calculation



# Multi-Head Attention



# Multi-Head Attention

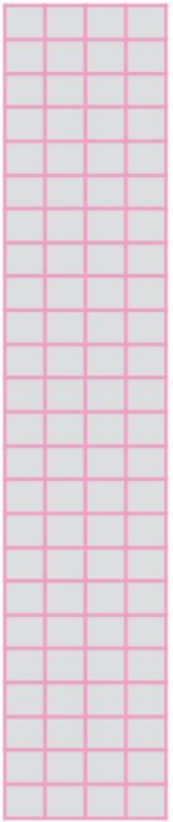


# Multi-Head Attention

1) Concatenate all the attention heads

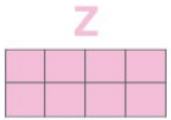

2) Multiply with a weight matrix $W^O$ that was trained jointly with the model

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

=



1) This is our input sentence*

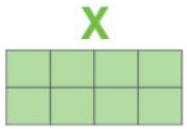
2) We embed each word*

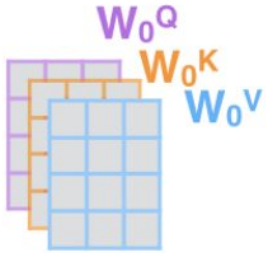
3) Split into 8 heads. We multiply X or R with weight matrices

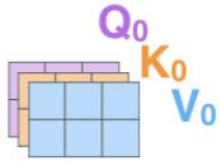
4) Calculate attention using the resulting Q/K/V matrices

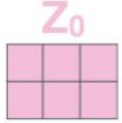
5) Concatenate the resulting Z matrices, then multiply with weight matrix $W^O$ to produce the output of the layer

Thinking Machines

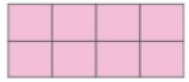


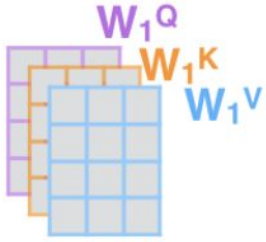



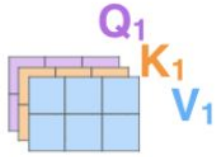



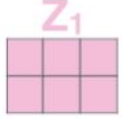





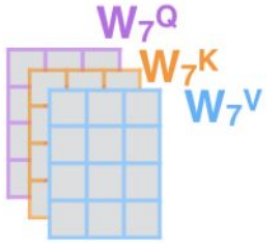


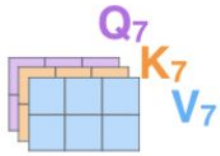



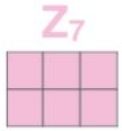



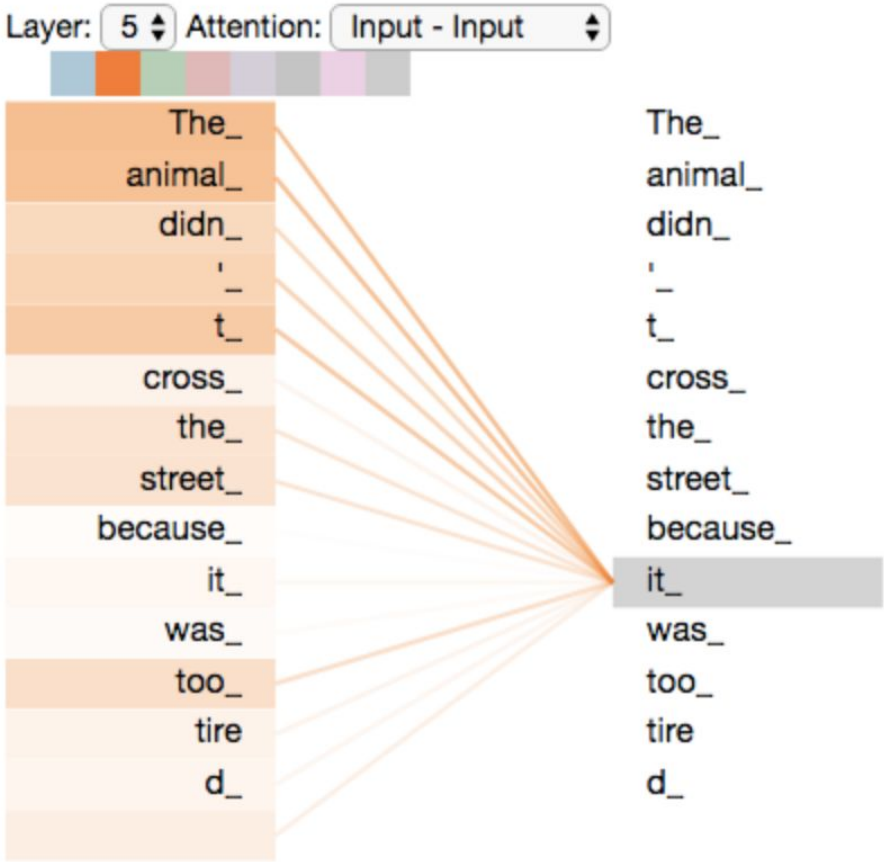

* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

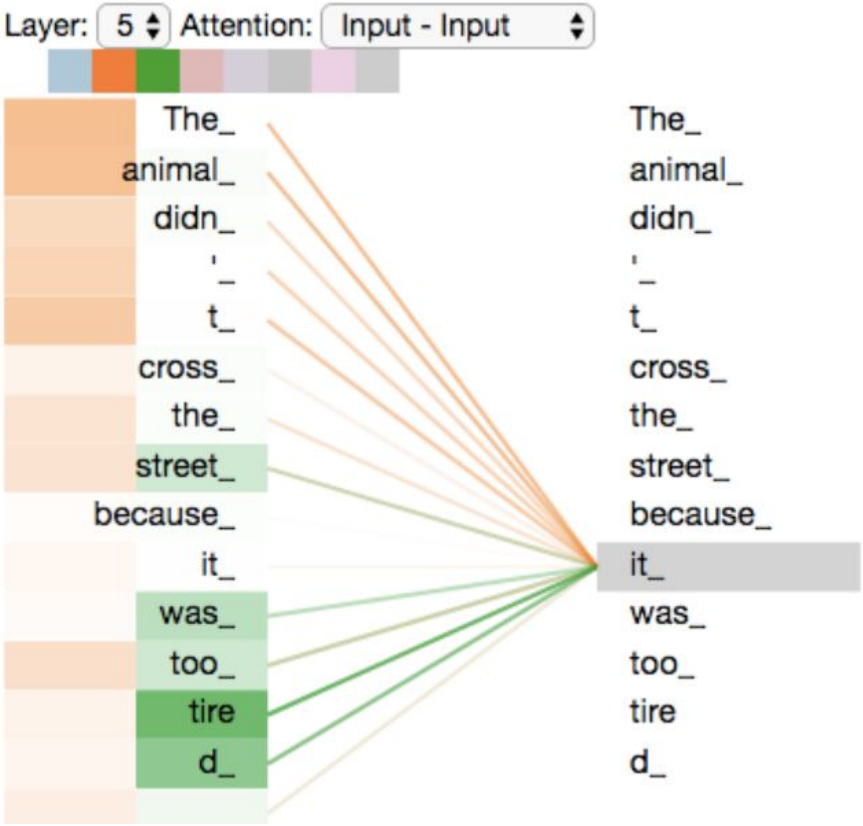



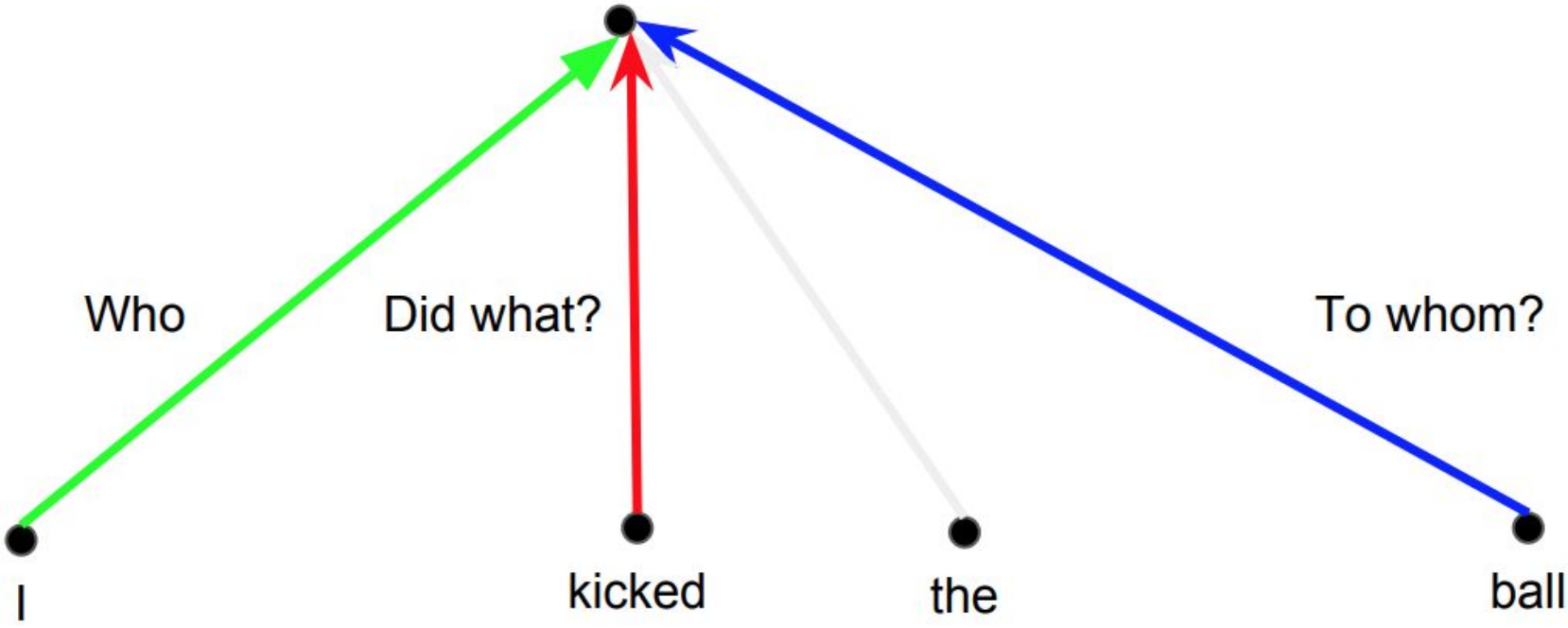



# Multi-Head Attention

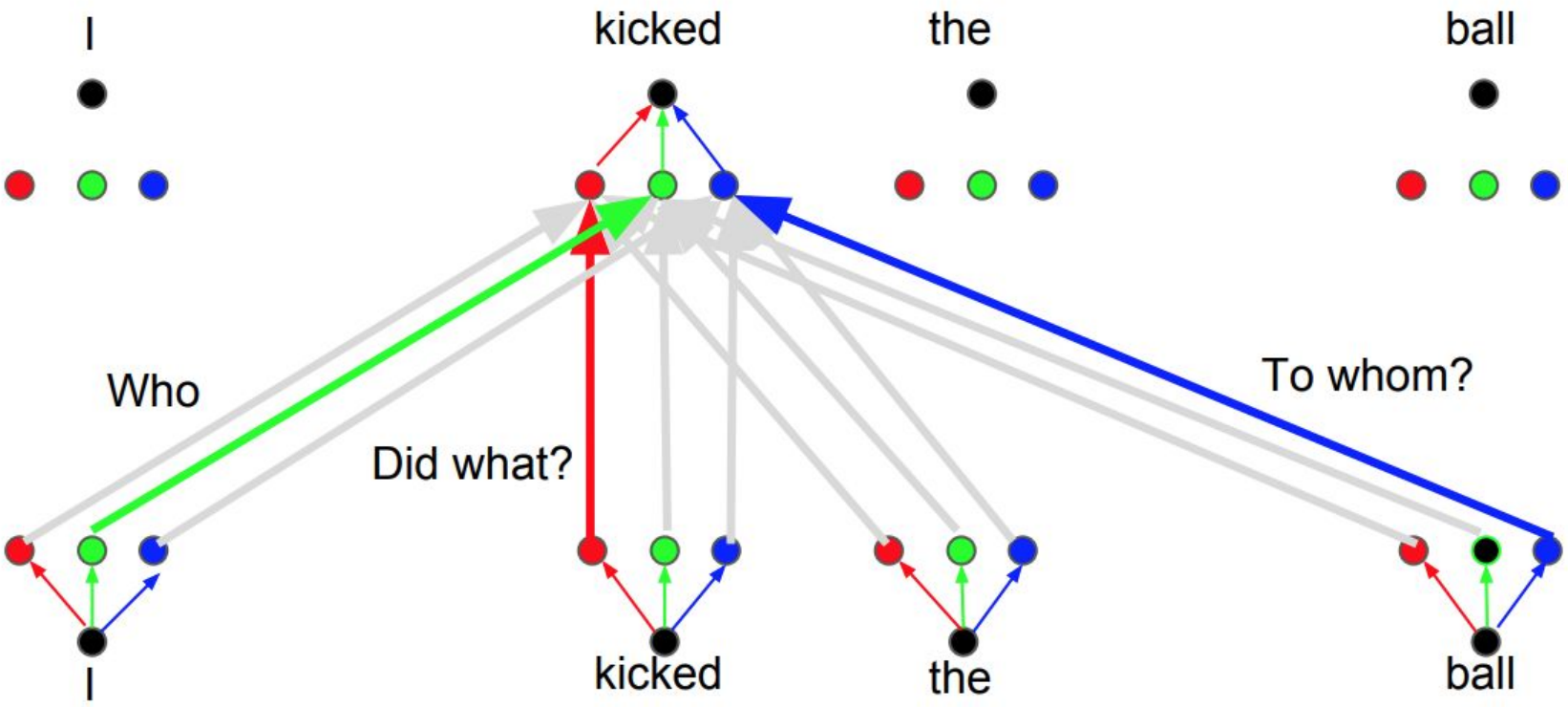


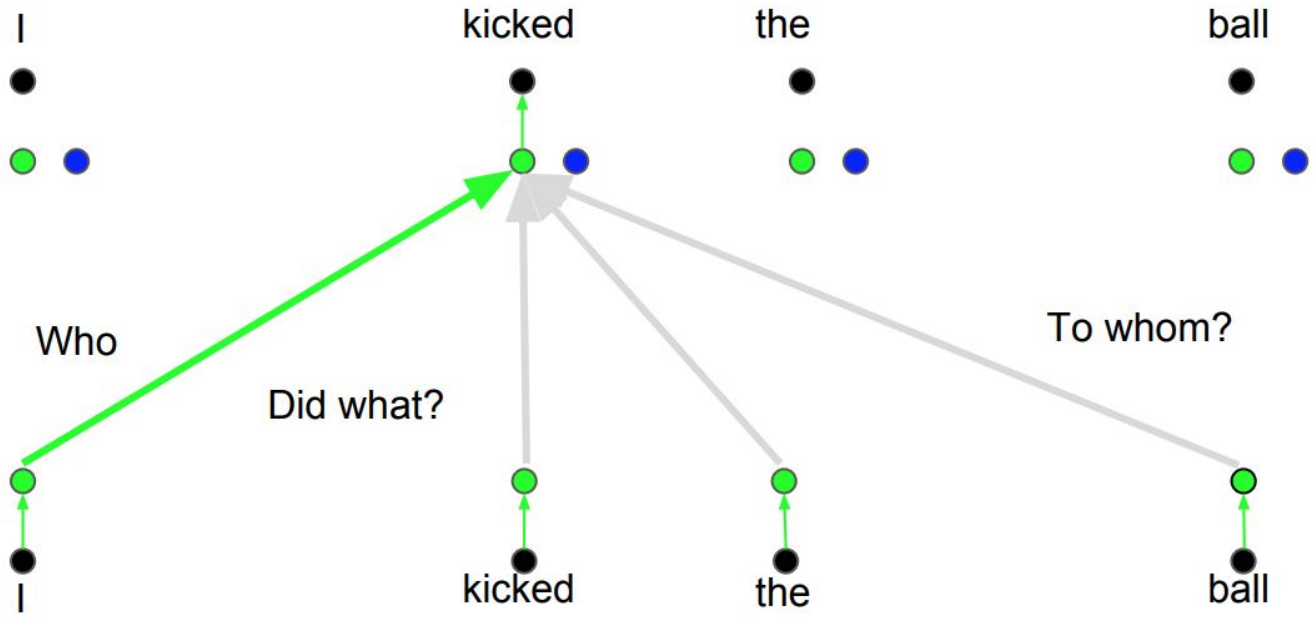
# Why Multi-Head Attention?



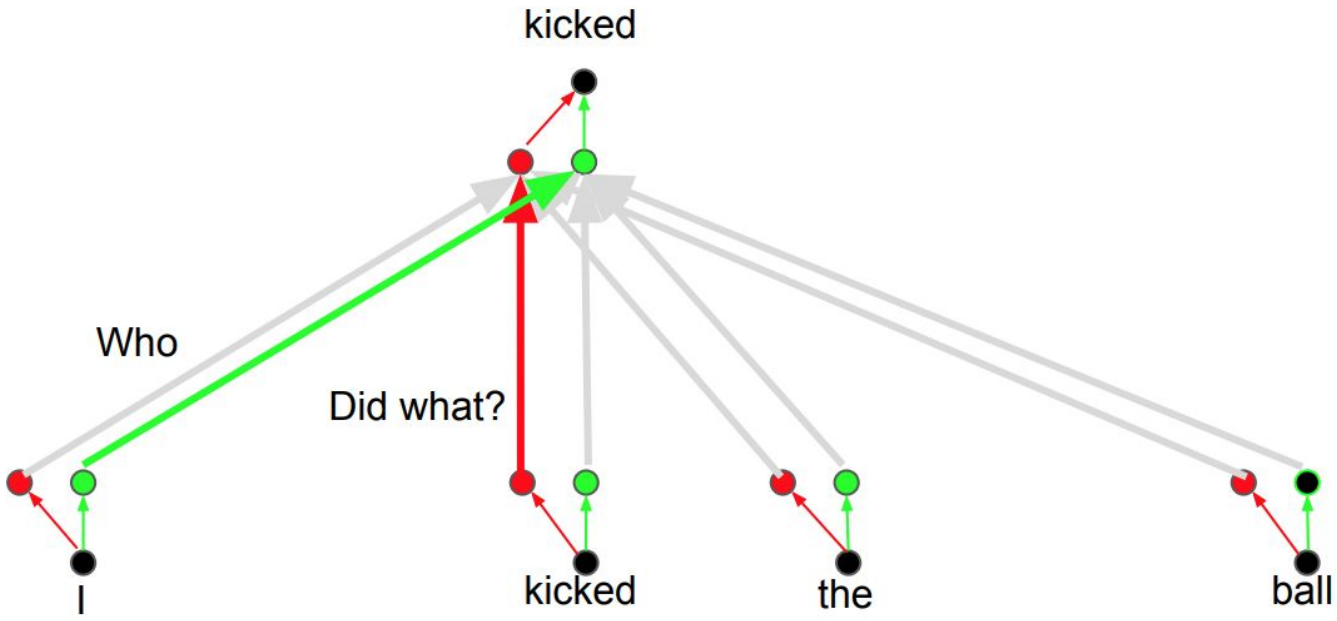
# Why Multi-Head Attention?



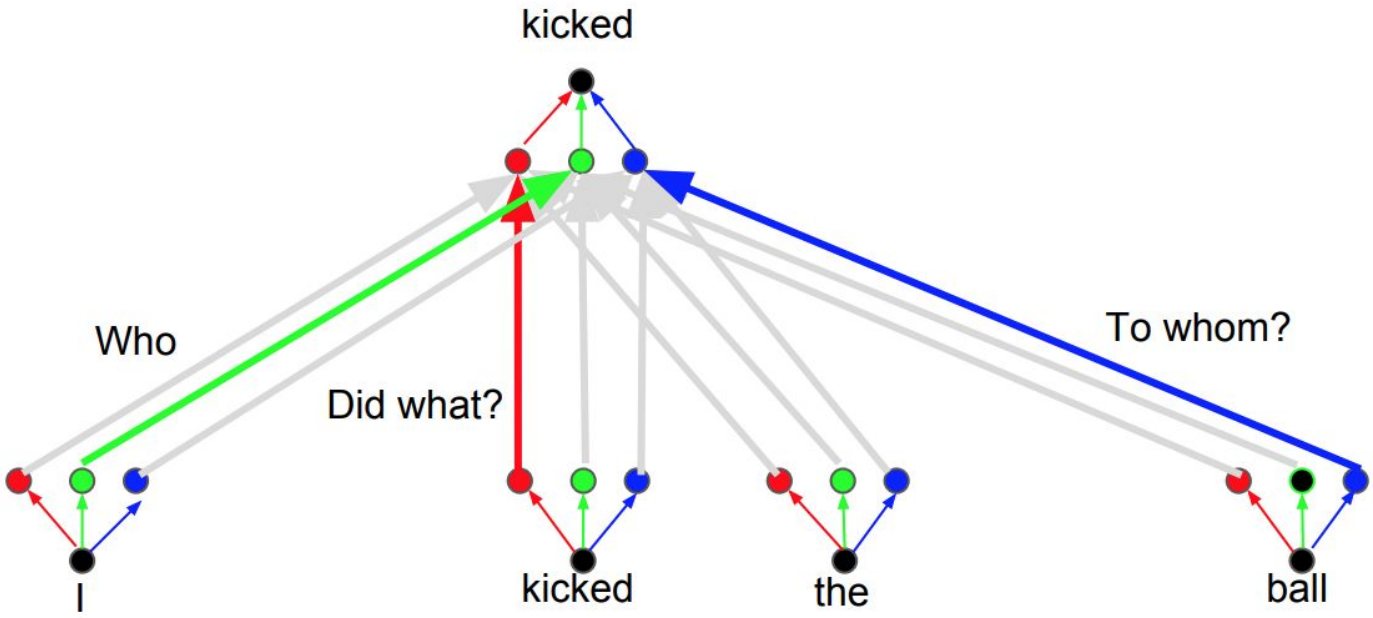
# Attention head: Who



# Attention head: Did What?

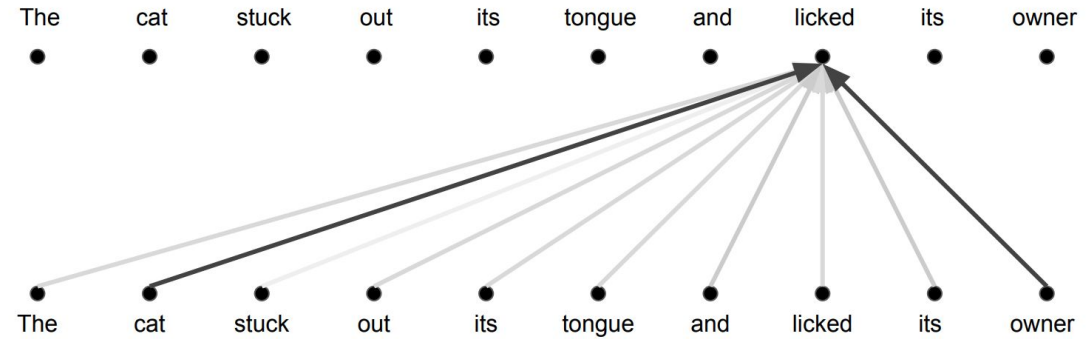


# Attention head: To Whom?



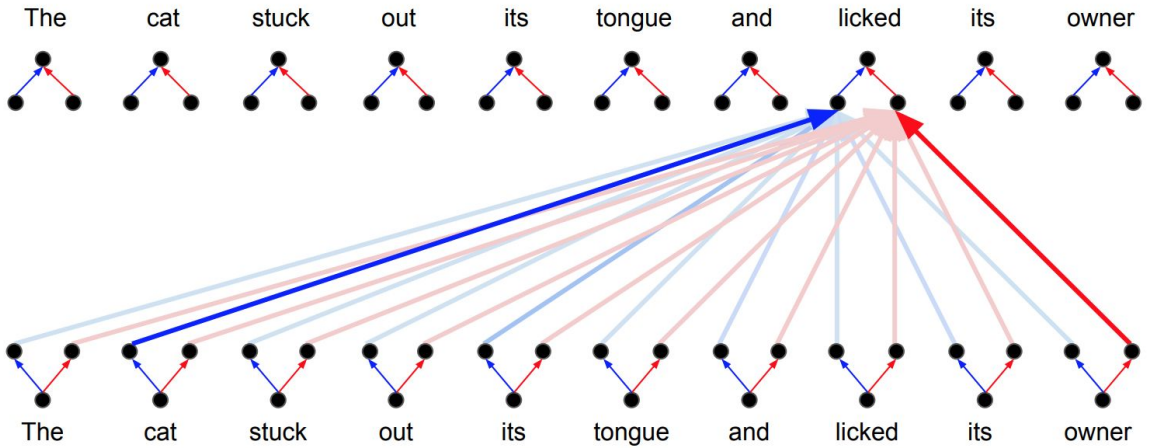
# Attention vs. Multi-Head Attention

**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** |

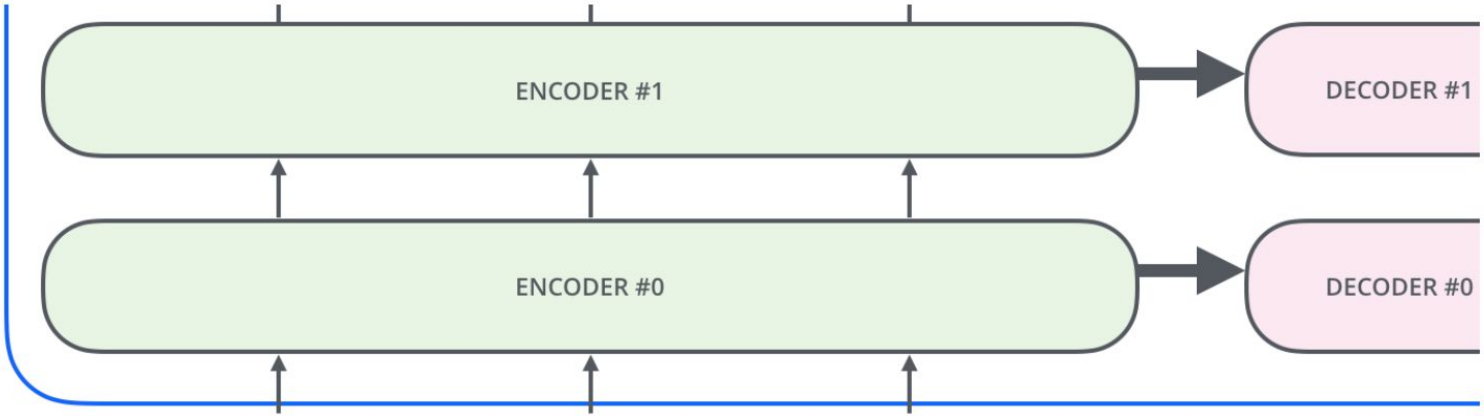
*Transformer models trained >3x faster than the others.

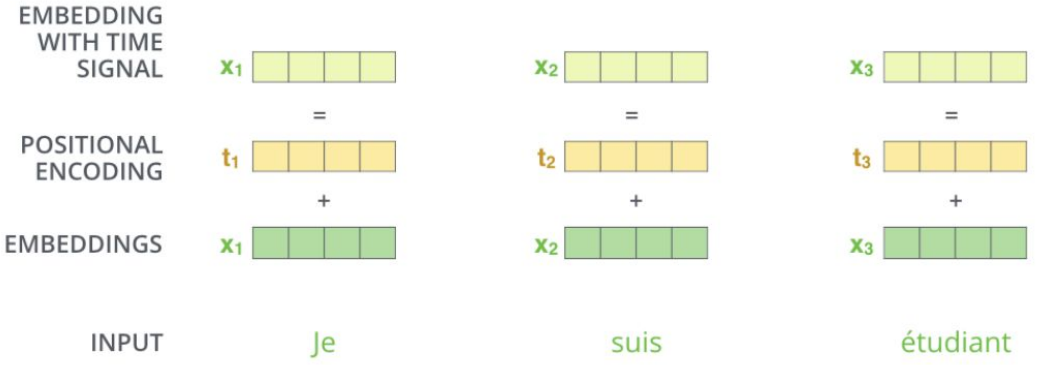
# Research Challenges

- 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

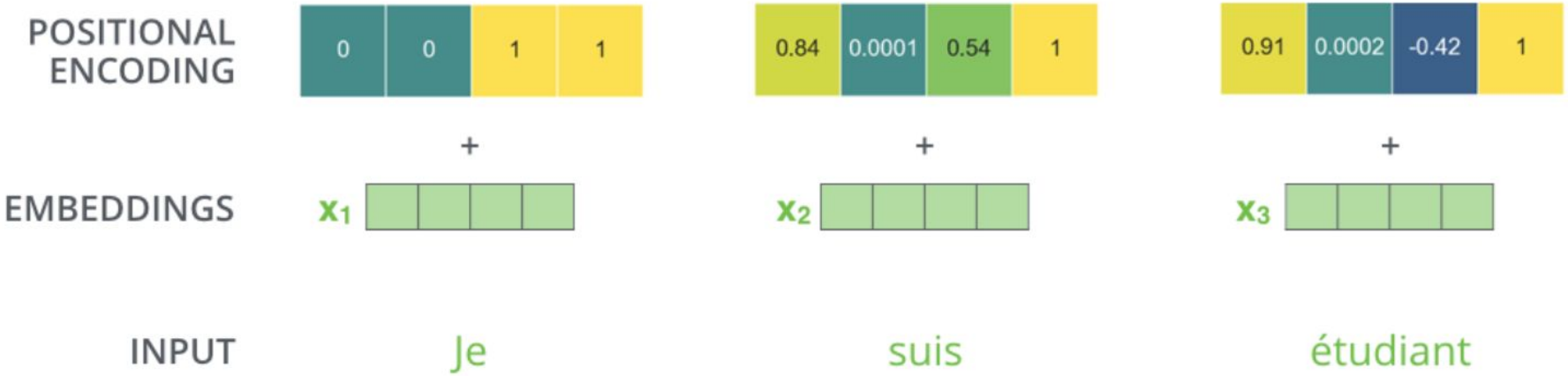
$$PE_{(pos,\ 2i)} = sin(pos / 10000^{2i\ /\ d_{\text{model}}})$$

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

- *pos* is the position
- *i* is the dimension.

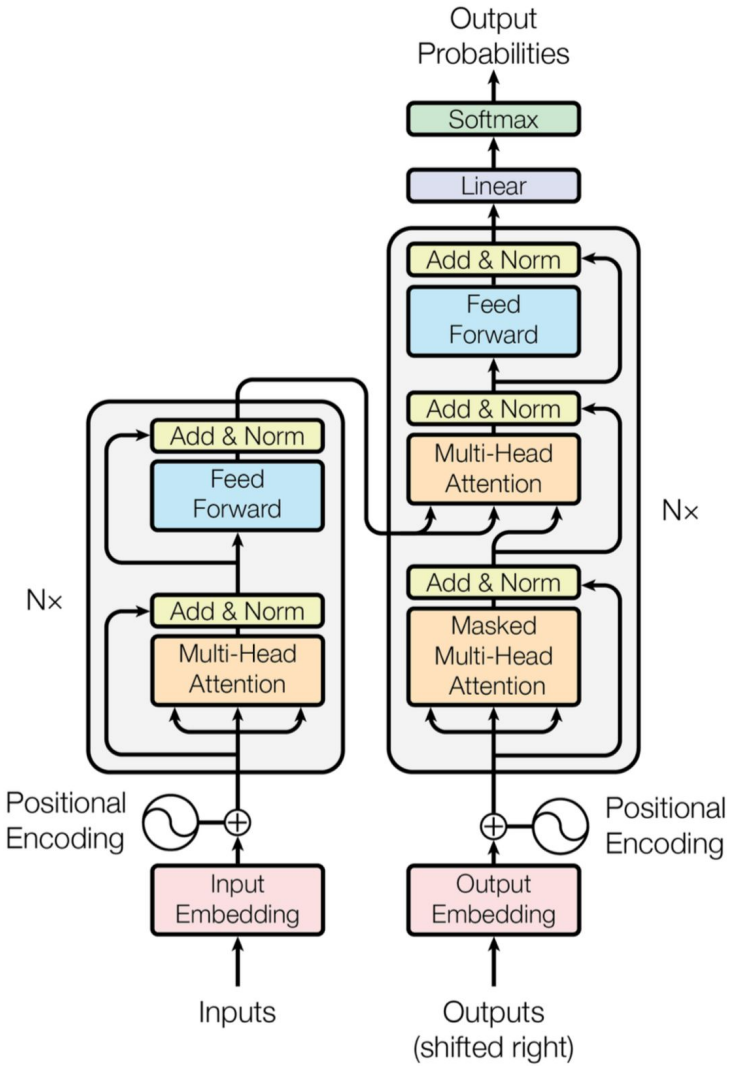
Each dimension of the positional encoding corresponds to a sinusoid.
The wavelengths form a geometric progression from 2pi to 2pi * 10 000

# Positional Encoding

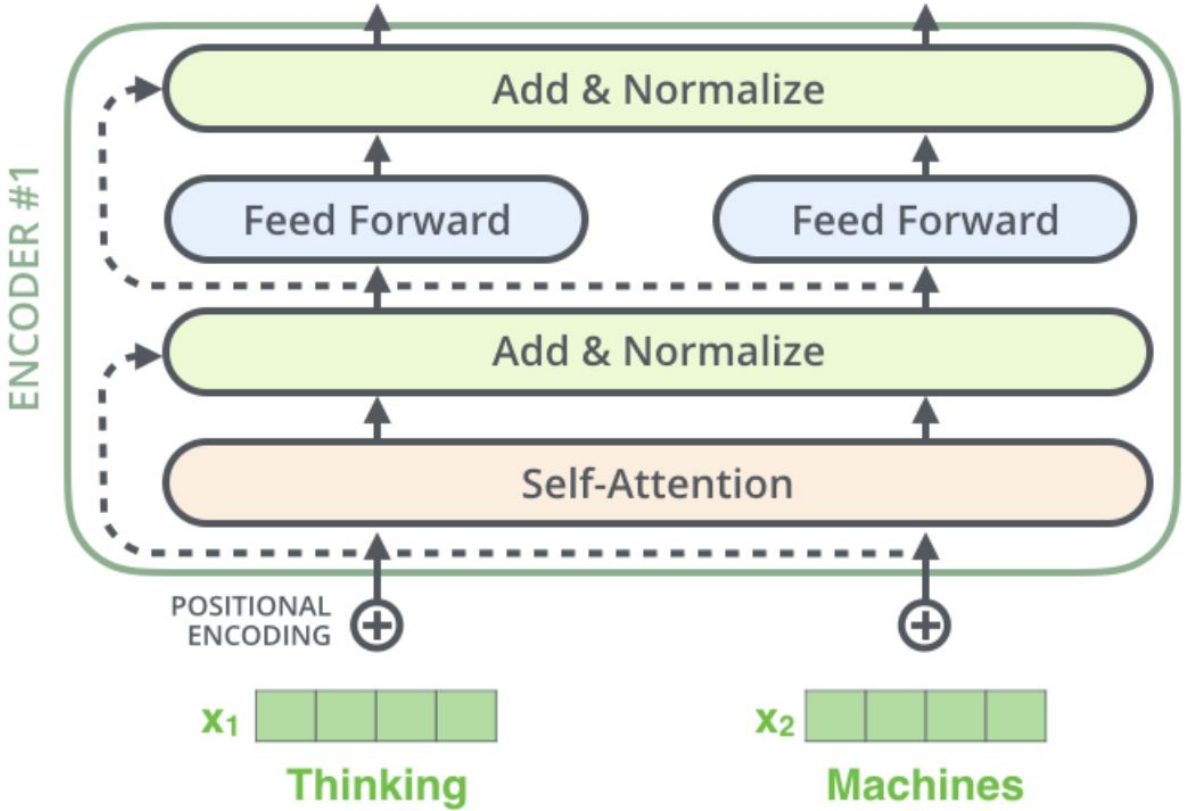


# Layer Normalization

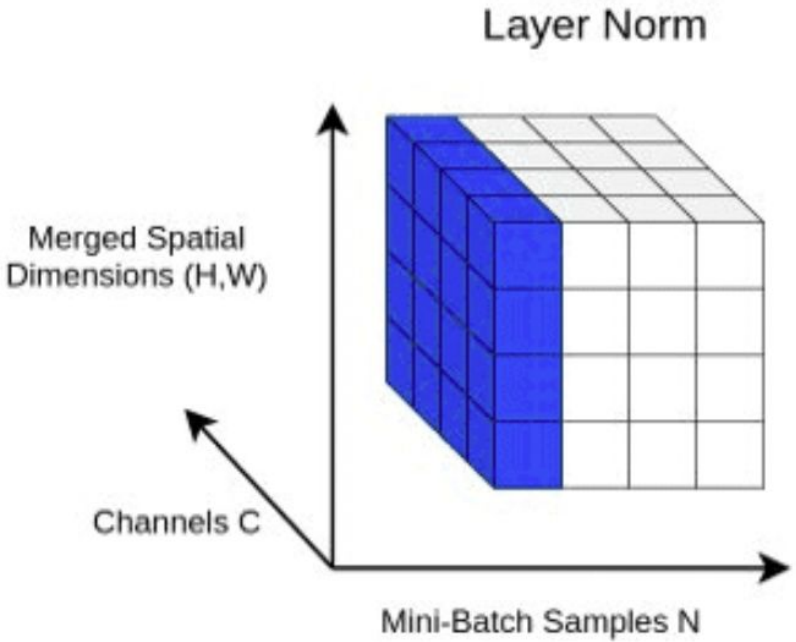
# The Transformer: recap

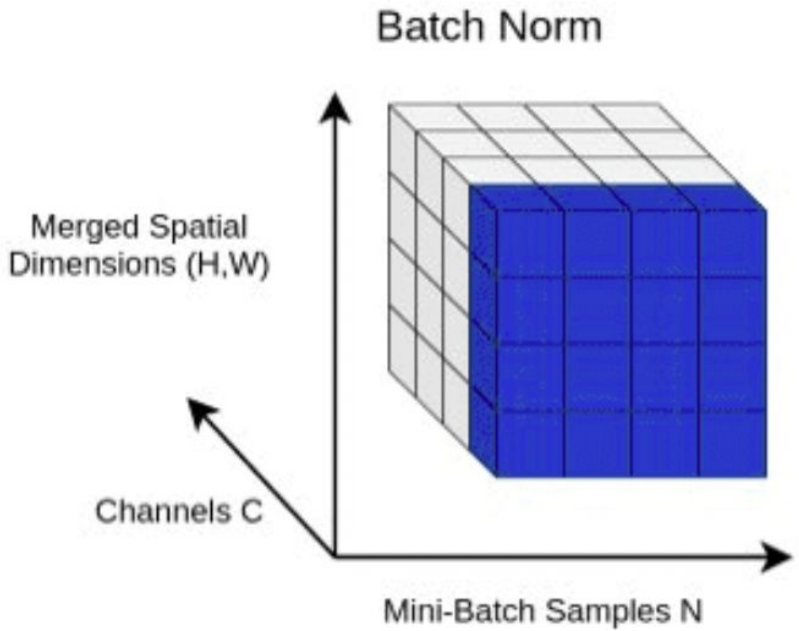


# Layer Normalization

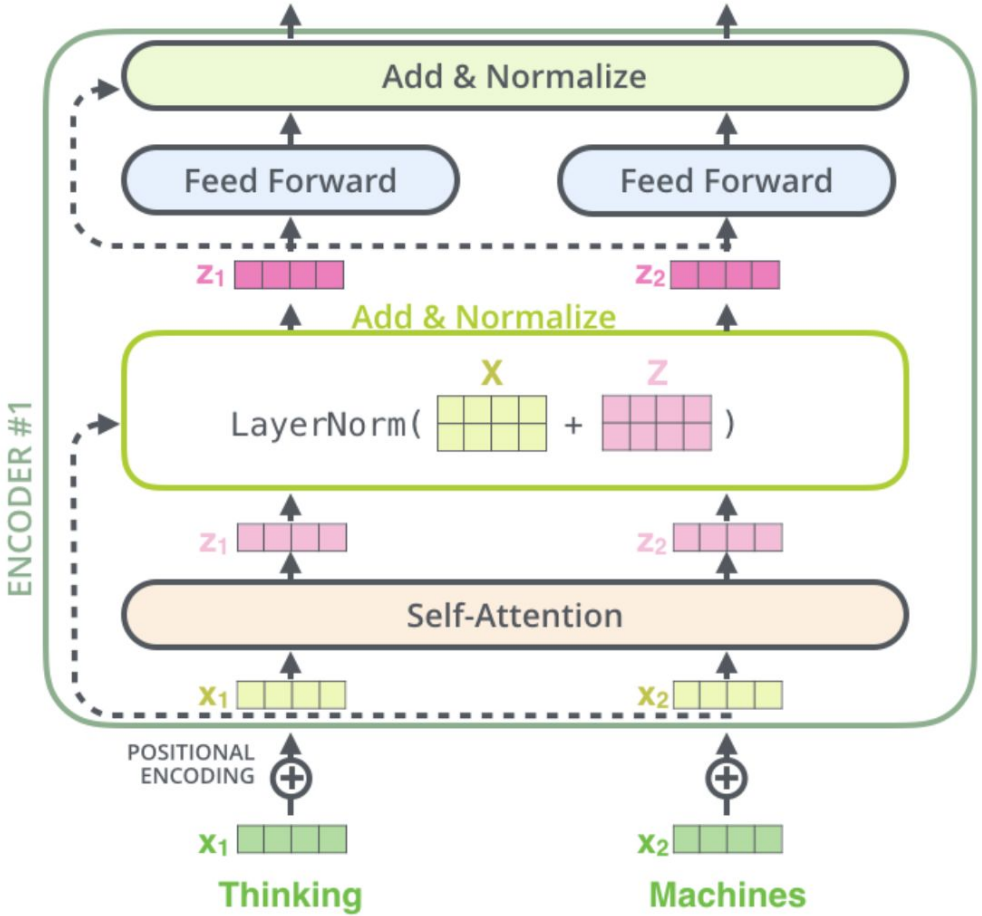


# Layer Norm vs. Batch Norm

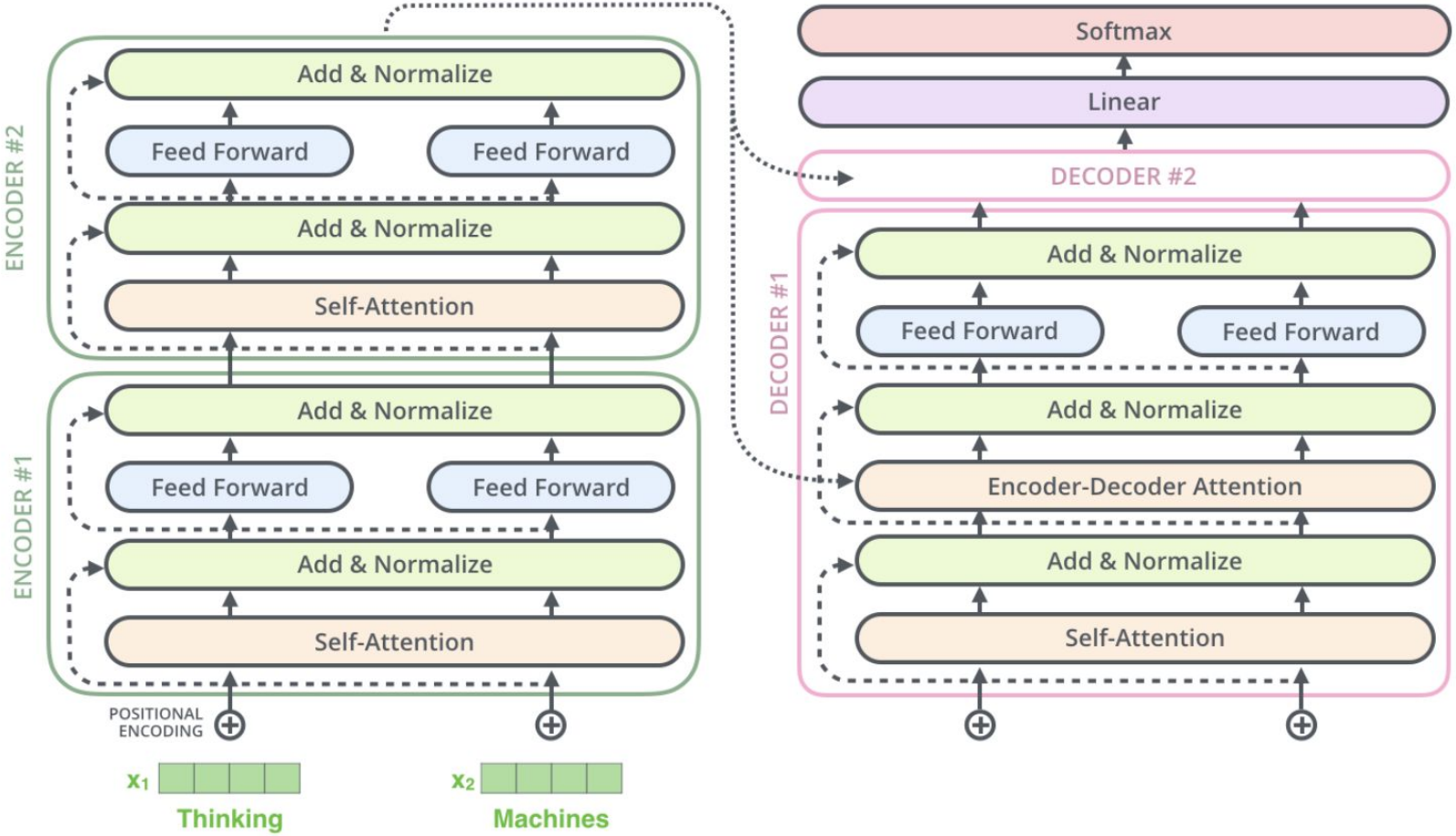


# Layer Normalization

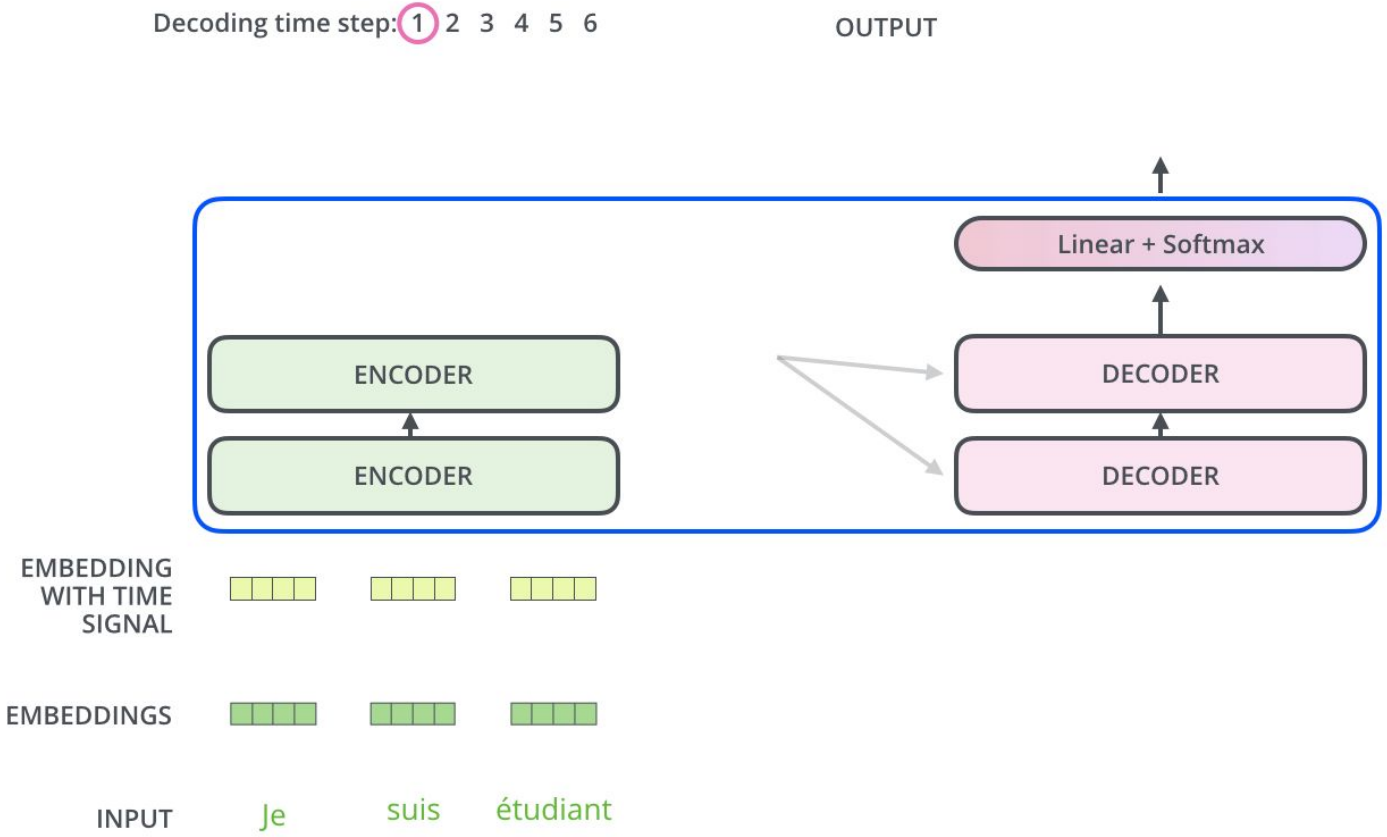


# Layer Normalization

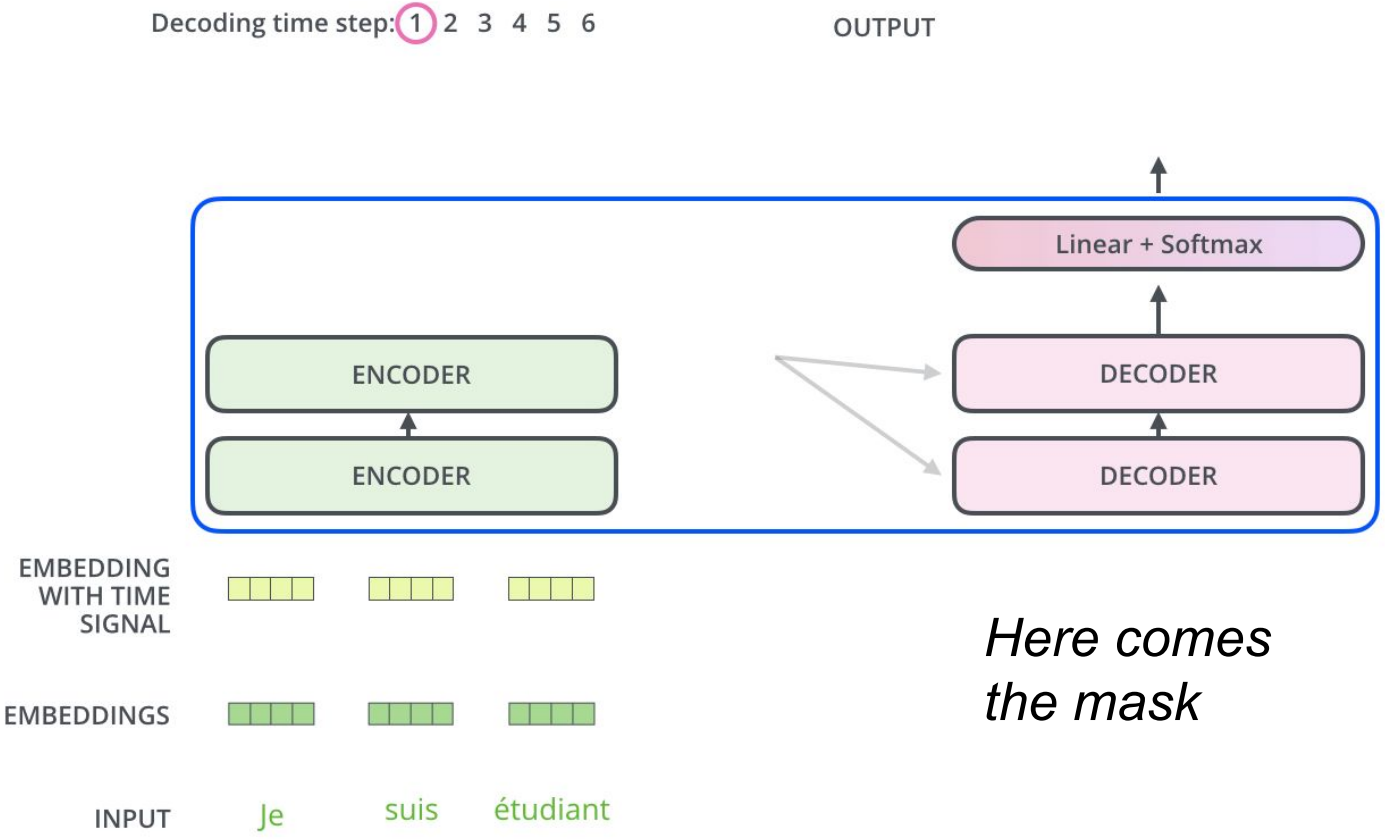


# The Decoder

# The Decoder Side

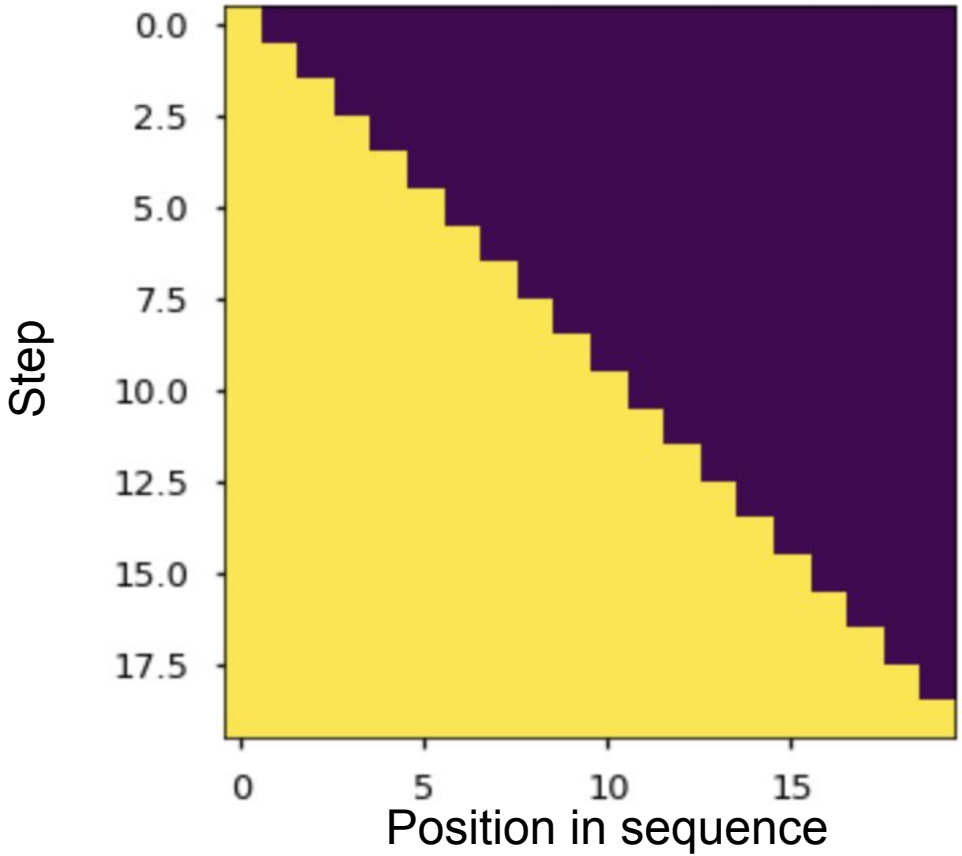


# The Decoder Side

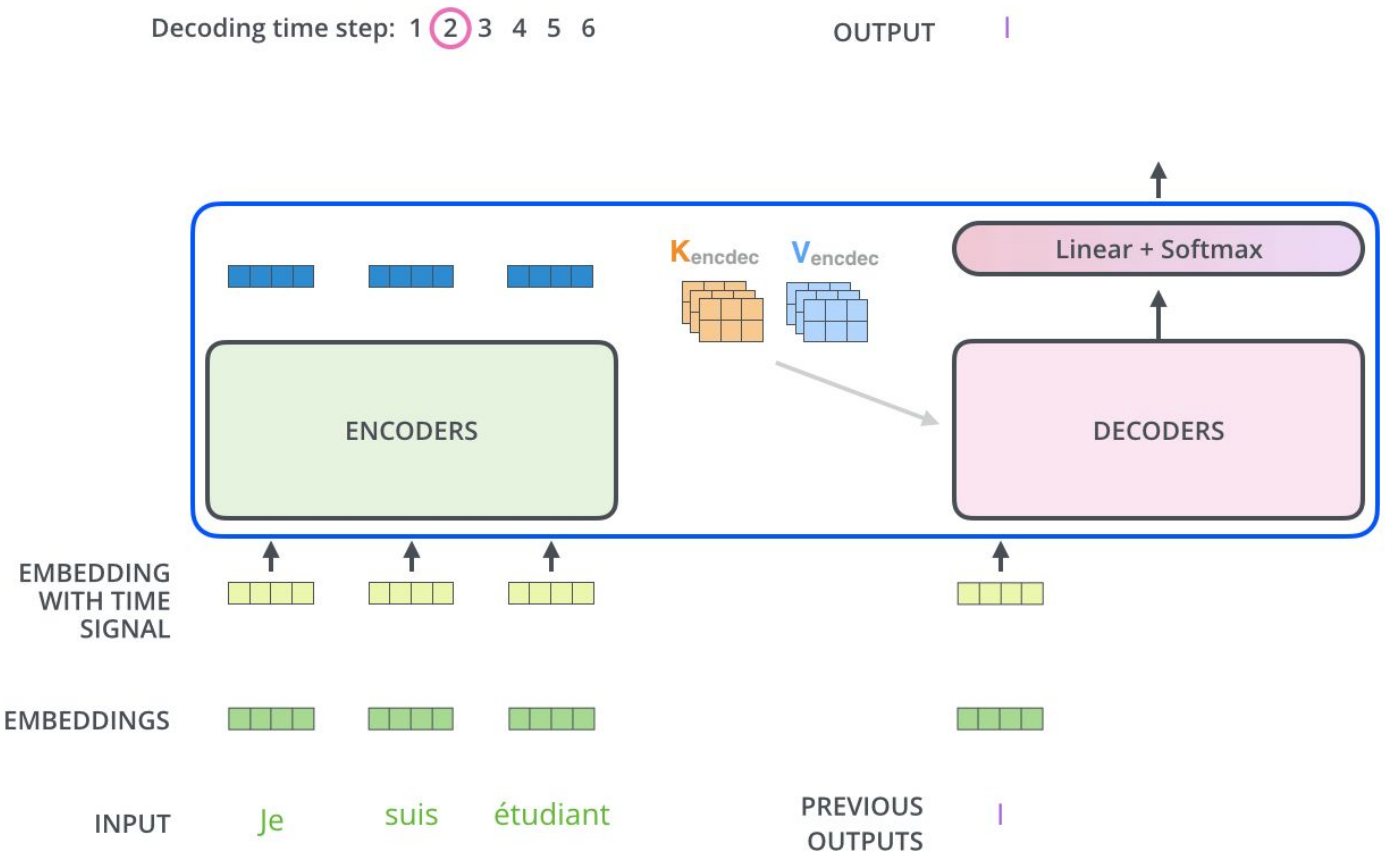



*Here comes the mask*

# The masked decoder input



# The Decoder Side



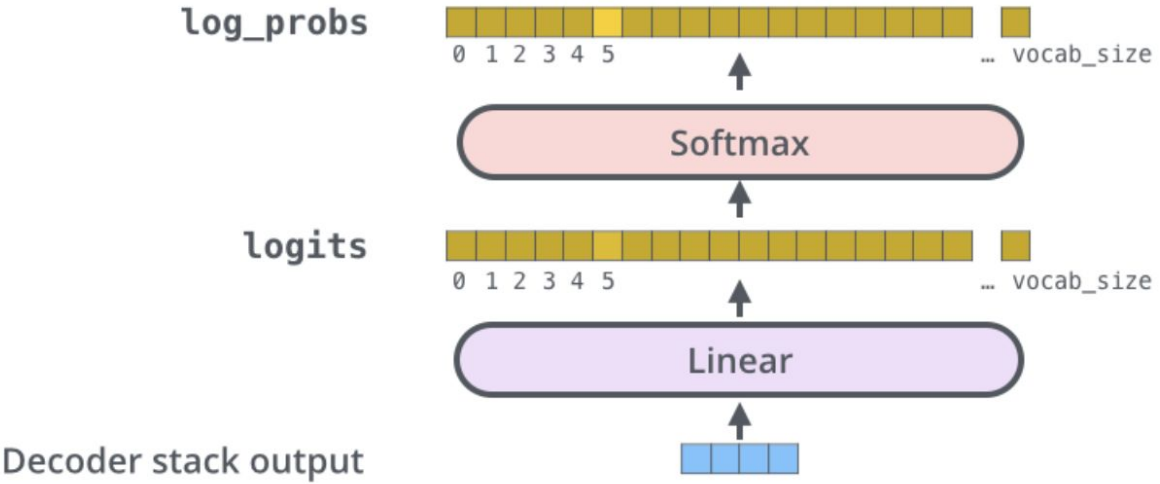
# Final Linear and Softmax Layer

Which word in our vocabulary is associated with this index?

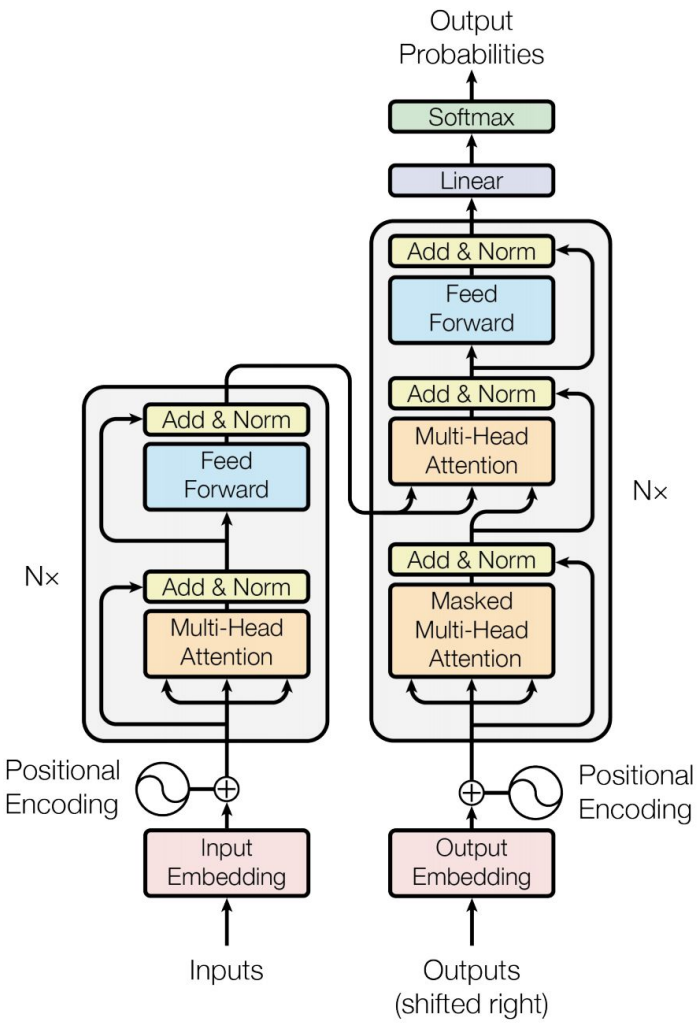
am

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

5



# The Transformer



# Outro

- Transformer is a very powerful architecture based on self-attention mechanism
- Physical analogues can help you