Procesamiento de Lenguaje Natural

Clase 13 – Transformers

Ph.D. Rubén Manrique

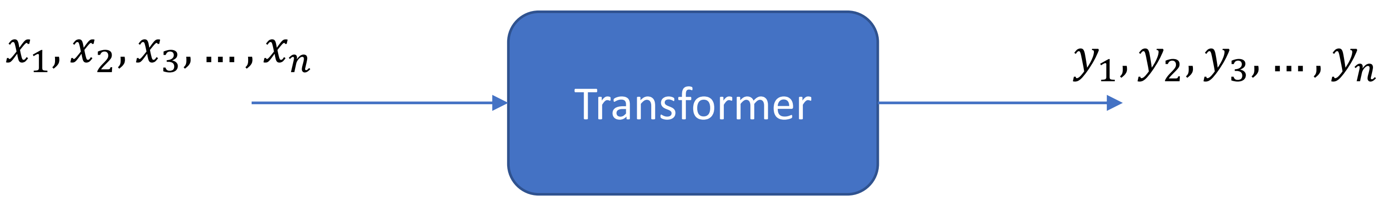
rf.manrique@uniandes.edu.co

Maestría en Ingeniería de Sistemas y Computación

- Chapter 9/11: Transformers

Transformers

- An approach to sequence processing that eliminates recurrent connections and returns to architectures reminiscent of the fully connected networks.

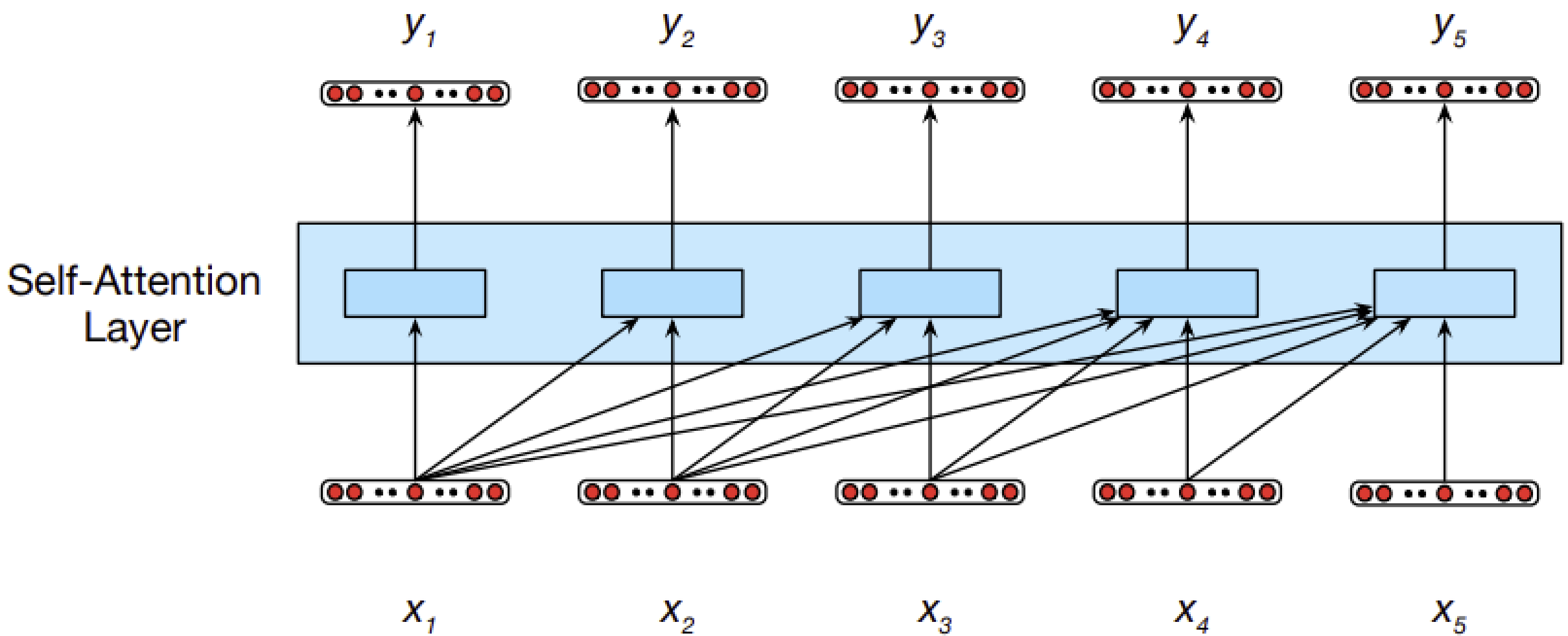


Simple feedforward networks
+
Self-attention layers

Self-attention (I)

- Self-attention allows a network to directly extract and use information from arbitrarily large contexts.

- There is not need to pass it through intermediate recurrent connections as in RNNs.



- The model has access to all of the inputs up to an including the one under consideration.
- No access to information about inputs beyond the current one.
- The computation of y_3 is based on a set of **comparisons** between the input x_3 and its preceding elements x_1 and x_2, and to x_3 itself.

Self-attention (II): Ability to compare

- The attention-based approach is the ability to compare an item of interest to a collection of other items in way that reveals their relevance in the current context.
- The simplest form of comparison between elements in a self-attention layer is a dot product.

$$score(x_i, x_j) = x_i \cdot x_j$$

- The larger the value the more similar the vectors that are being compared.

Self-attention (II): Compute the output

- Suppose you want to compute y_3, so you compare the different inputs:
  - $\boldsymbol{x_3} \cdot \boldsymbol{x_1}$
  - $\boldsymbol{x_3} \cdot \boldsymbol{x_2}$
  - $\boldsymbol{x_3} \cdot \boldsymbol{x_3}$

a set of comparisons to relevant items in some context

- How to evaluate the proportion of relevance of each input?
  - Create a vector of weights $\boldsymbol{\alpha}$

$$\boldsymbol{\alpha_{ij}} = softmax\left(score(x_i, x_j)\right) \ \forall\, \boldsymbol{j} \leq \boldsymbol{i}$$

Normalization to provide a probability distribution

$$y_i = \sum_{j \leq i} \boldsymbol{\alpha_{ij}}\, \boldsymbol{x_j}$$

weighted sum using this distribution

We need extra parameters….

- Transformers include additional parameters in the form of a set of weight matrices that operate over the input embeddings.
- Input embeddings roles:
  - Query: As the current focus of attention when being compared to all of the other preceding inputs.
  - Key: In its role as a preceding input being compared to the current focus of attention.
  - Value: as a value used to compute the output for the current focus of
  - attention.
- Transformers introduce three sets of weights:
  - W^Q, W^K, W^V
  - These weights will be used to compute linear transformations of each input x.
  - The resulting values being used in their respective roles in subsequent calculations.

We need extra parameters….

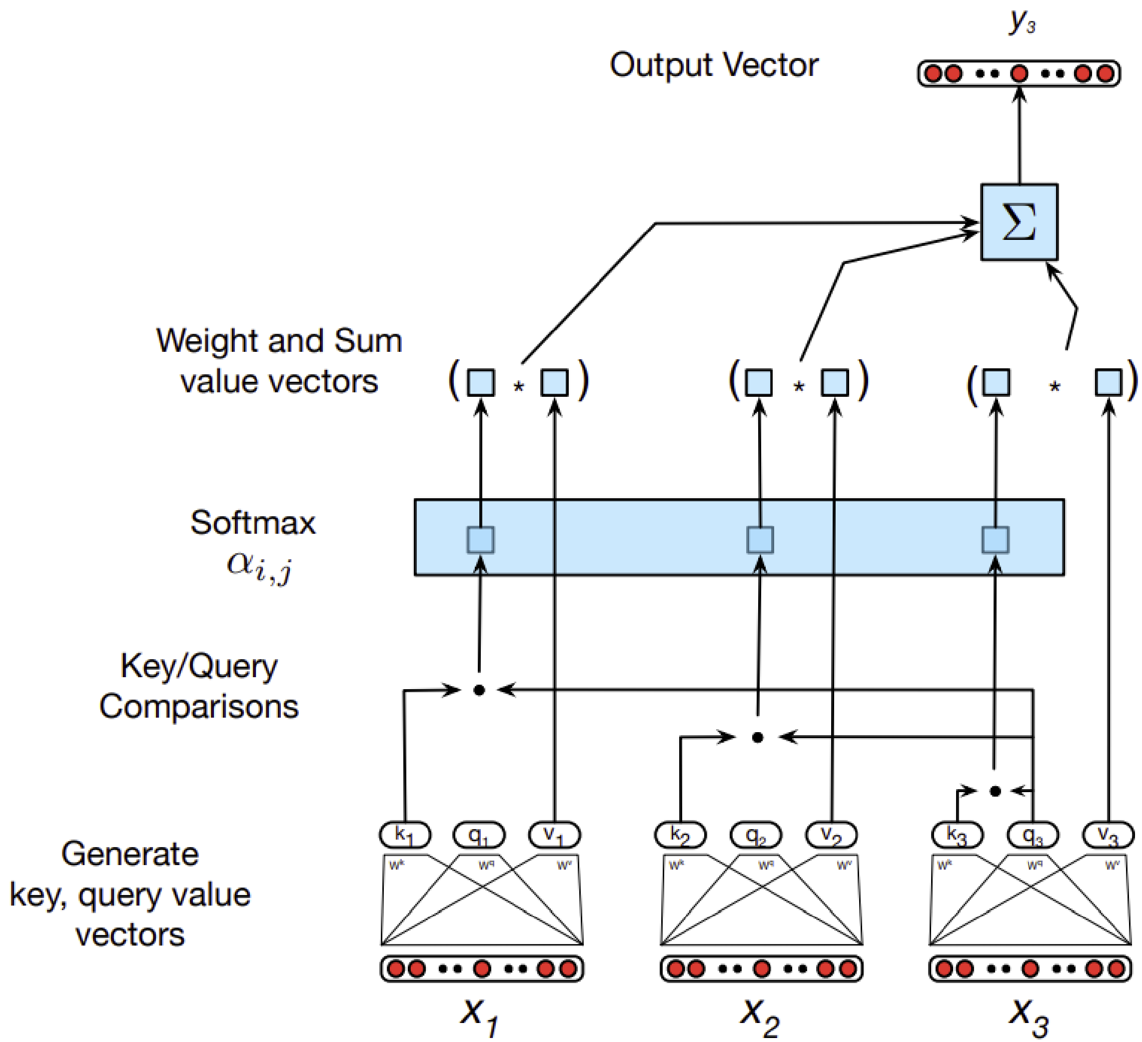
- Linear transformations:
  - $q_i = W^Q x_i, \quad W^Q \in \mathbb{R}^{d_q \times d_m}, \; d_m{=}1024, d_q = 64$
  - $k_i = W^K x_i, \quad W^K \in \mathbb{R}^{d_k \times d_m}, \; d_m{=}1024, d_k = 64$
  - $v_i = W^V x_i, \quad W^V \in \mathbb{R}^{d_v \times d_m}, \; d_m{=}1024, d_v = 64$

$$\boldsymbol{score(x_i, x_j) = q_i \cdot k_j}$$

$$\boldsymbol{\alpha_{ij}} = softmax\left(score(x_i, x_j)\right) \; \forall \, \boldsymbol{j} \le \boldsymbol{i}$$

$$y_i = \sum_{\boldsymbol{j \le i}} \boldsymbol{\alpha_{ij}\, v_j}$$

Vaswani et al., 2017.
https://arxiv.org/abs/1706.03762



Calculation of the value of the third element of a sequence using causal self-attention.

Score normalization

- Practical consideration about $\boldsymbol{\alpha_{ij}}$
  - Dot product can produce large values.
  - Exponentiating such large values can lead to numerical issues.
    - Effective loss of gradients during training.
- A typical approach is to divide the dot product by the square root of the dimensionality of the query and key vectors.

$$score(x_i, x_j) = \frac{q_i \cdot k_j}{\sqrt{d_k}}$$

Single matriz packing…

- Produce matrices containing all the key, query and value vectors:

$Q = W^Q X$

$K = W^K X$

$V = W^V X$

- Can we then calculate all in a single step?

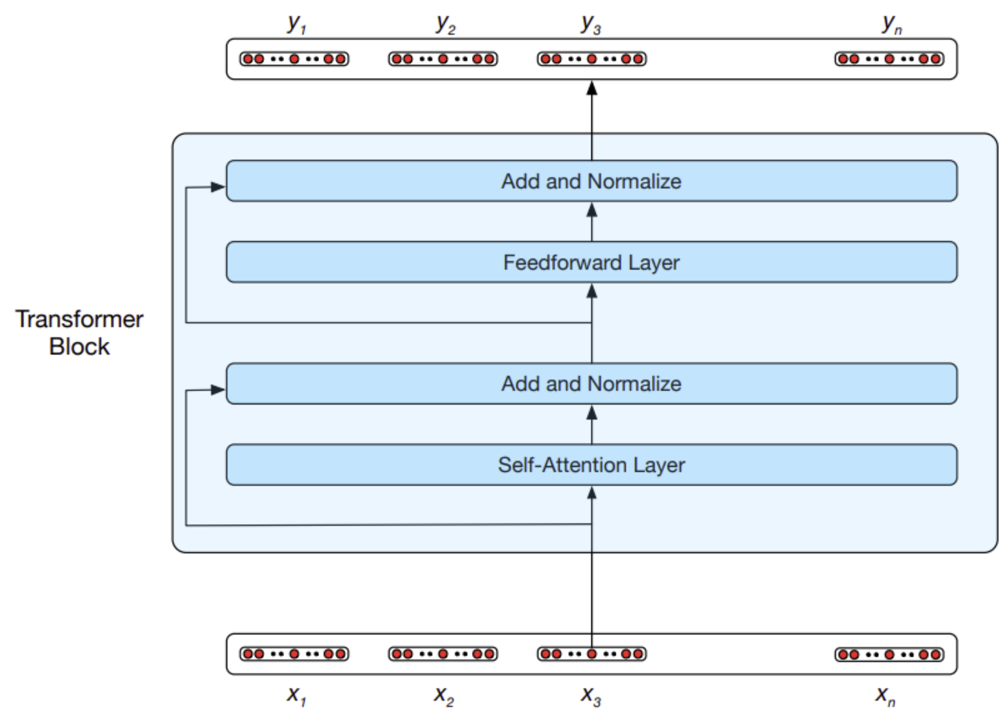
$$SelfAttention(Q, K, V) = softmax\left(\frac{\boldsymbol{QK^T}}{\sqrt{d_k}}\right)V$$

Problem: QK^T results in a score for each query value to every key value, including those that follow the query.

Solution: the elements in the upper-triangular portion of the comparisons matrix are zeroed out.

Transformer block

- We just see the self attention layer.
- A transformer block also contains:
  - Normalization layers: Batch normalization accelerates training, in some cases by halving the epochs or better, and provides some regularization, reducing generalization error.
  - Feedforward layers.

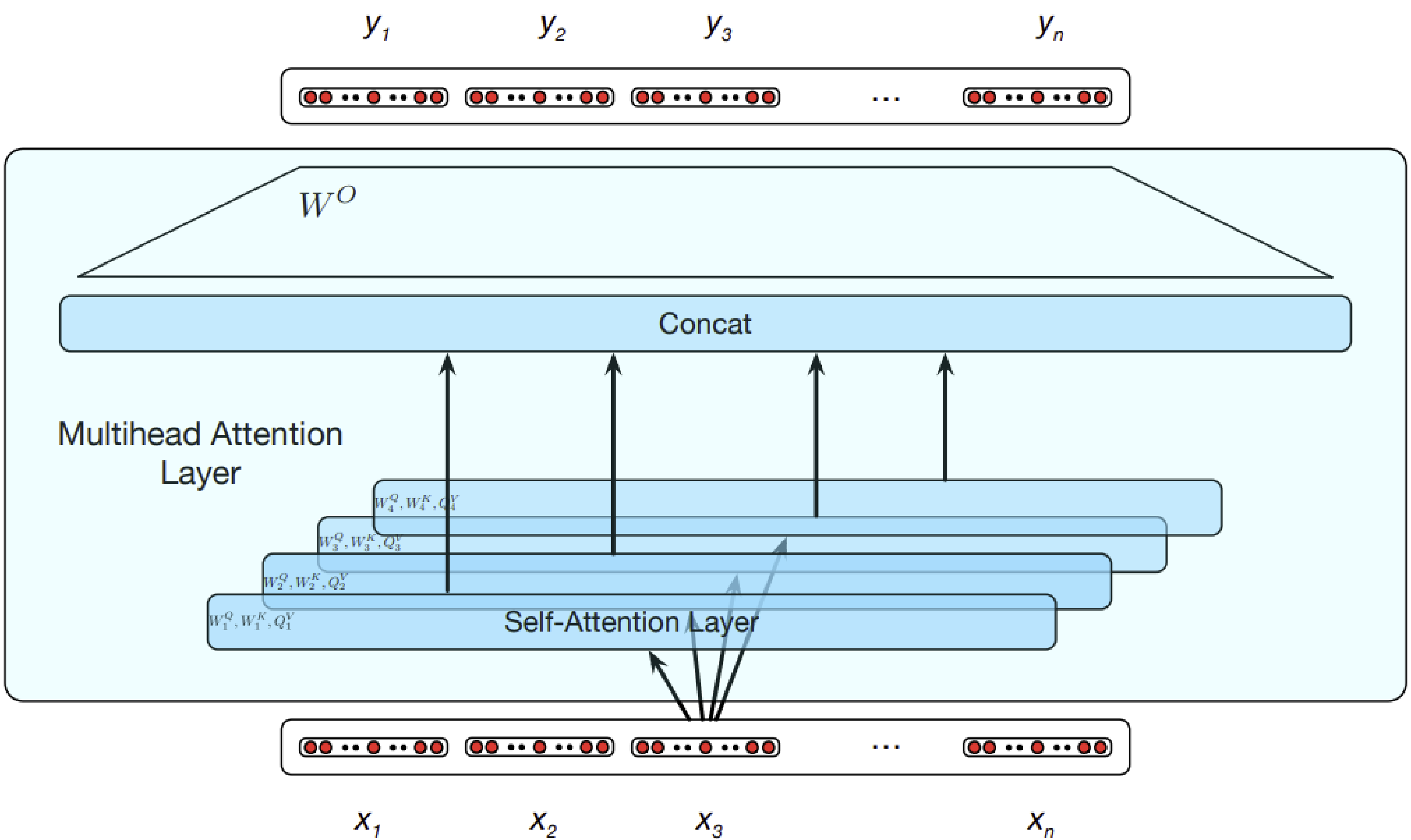


Multihead Attention Layers

- The different words in a sentence can relate to each other in many different ways simultaneously.

- Transformers address this issue with multihead self-attention layers.

- These are sets of self-attention layers, called heads, that reside in parallel layers at the same depth in a model, each with its own set of parameters.



$$MultiHeadAttn(Q,K,V) = W^O(head^1 \oplus head^2 \ldots \oplus headh)$$

Ojo Concatenación!

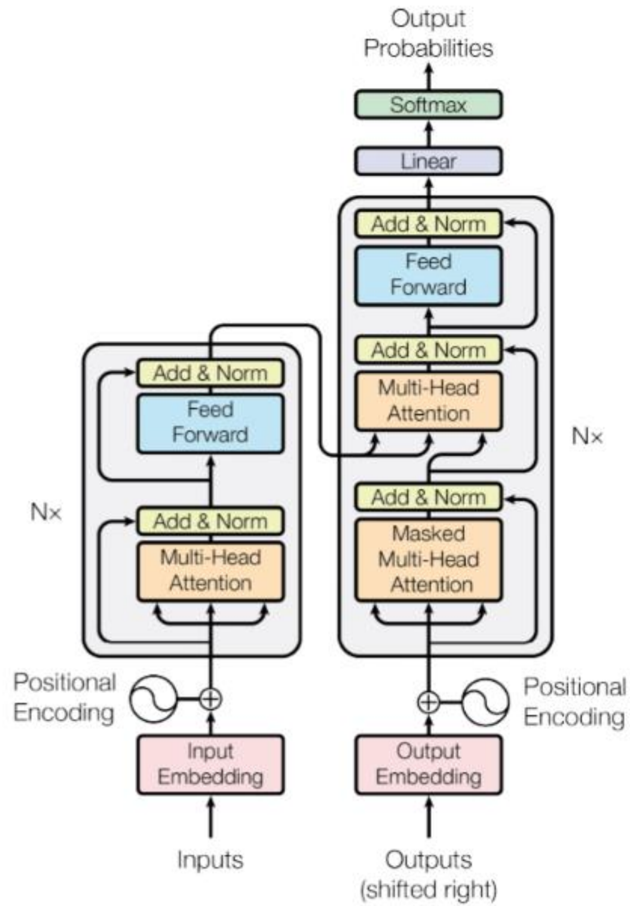
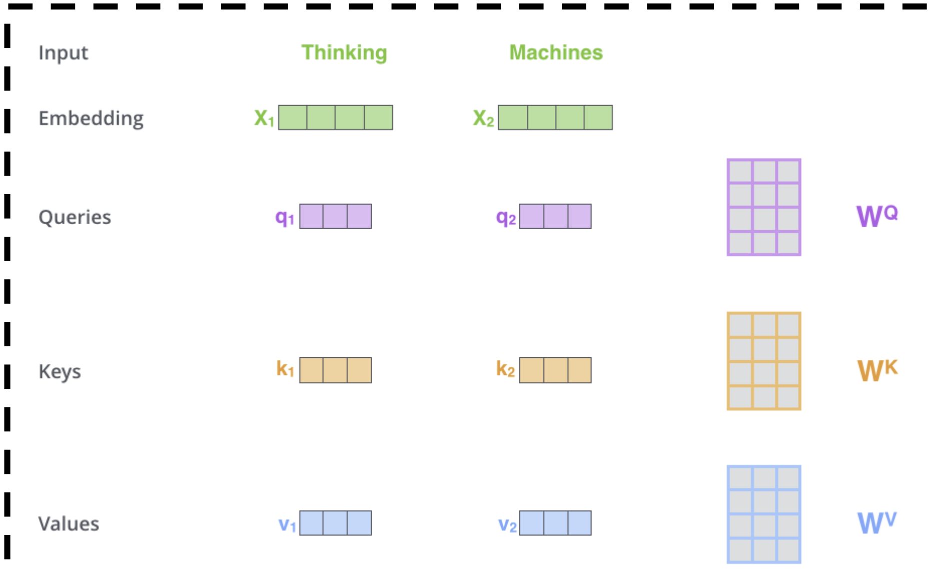
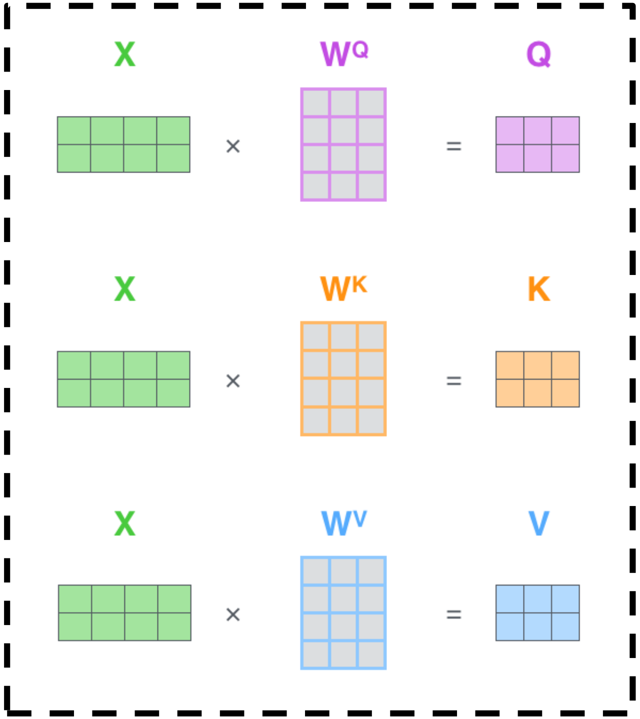


Figure 1: The Transformer - model architecture.

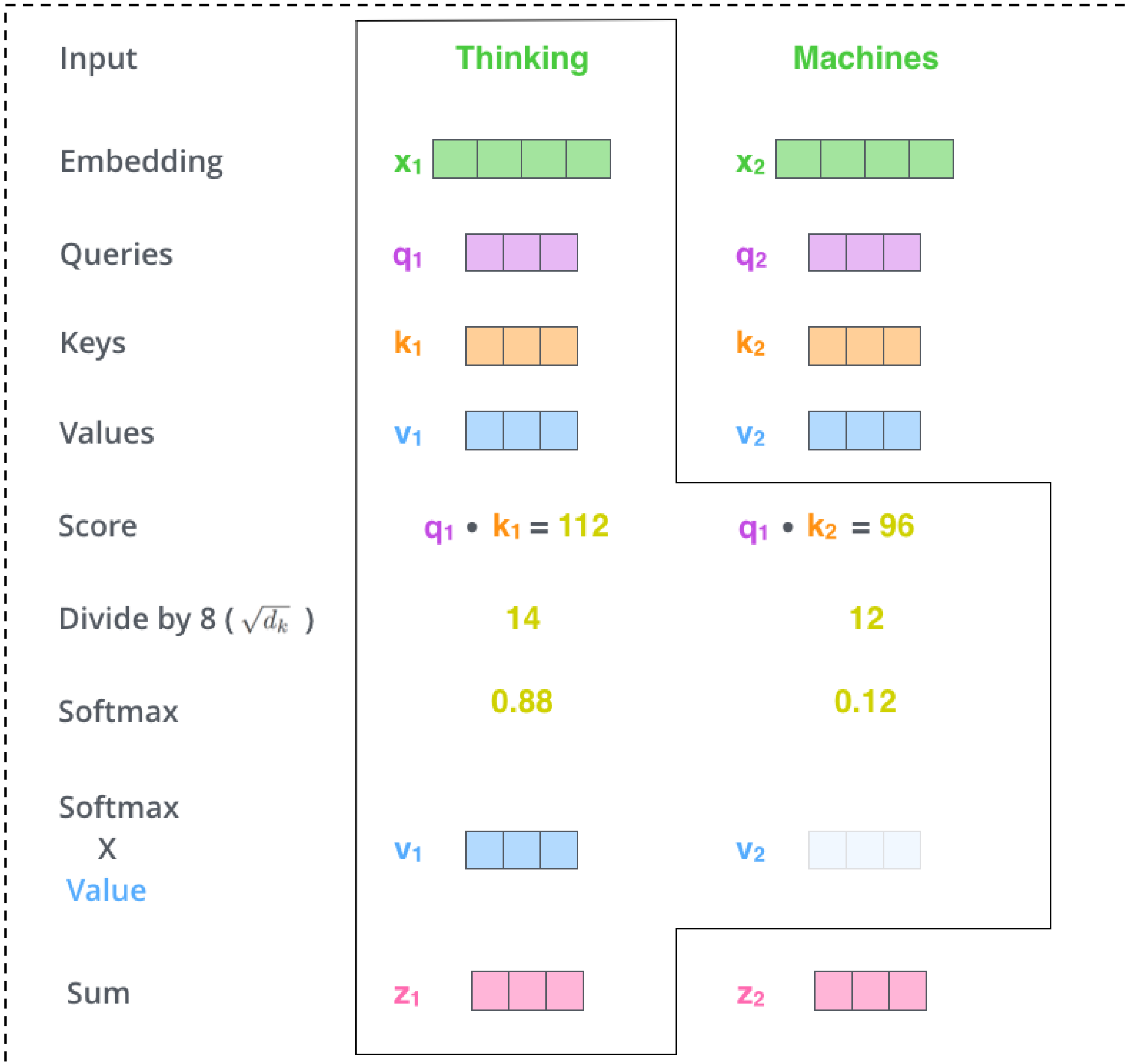
Self attention review (I)

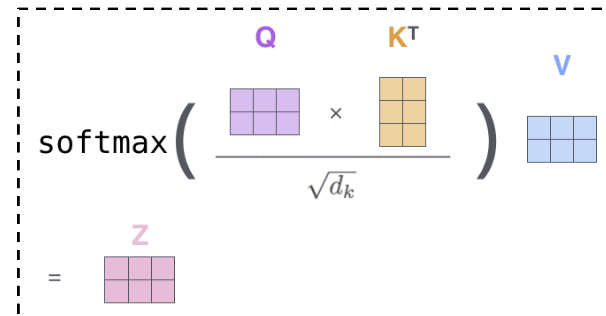




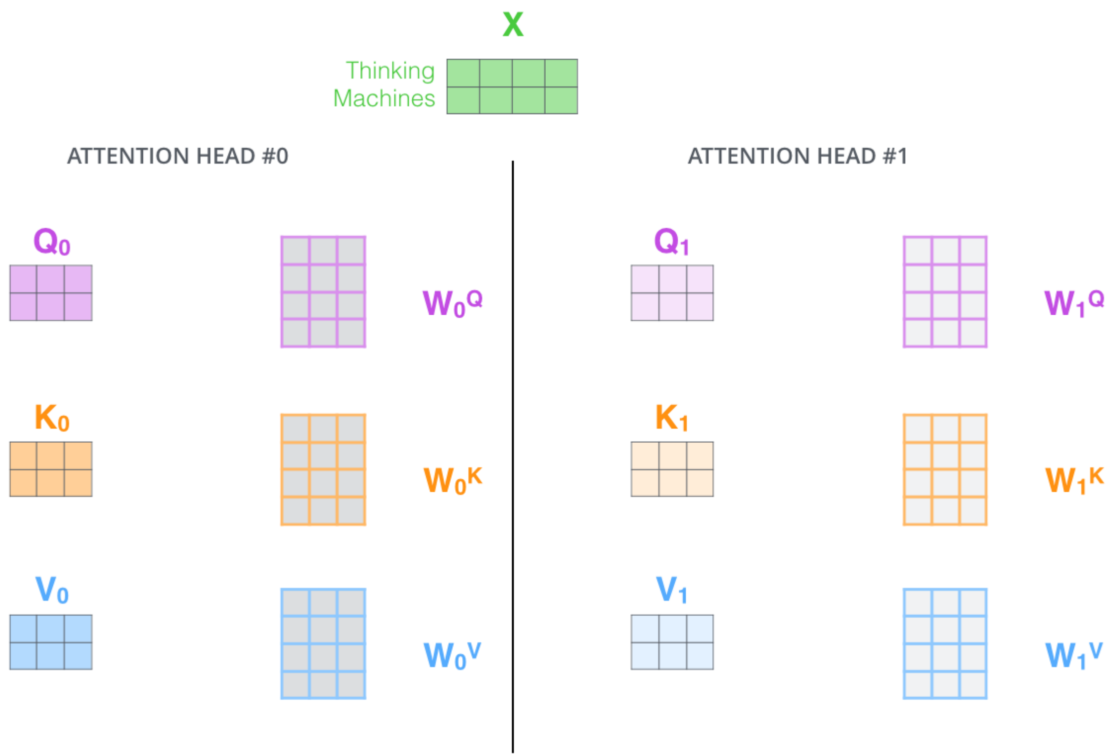
http://jalammar.github.io/illustrated-transformer/

Self attention review (II)

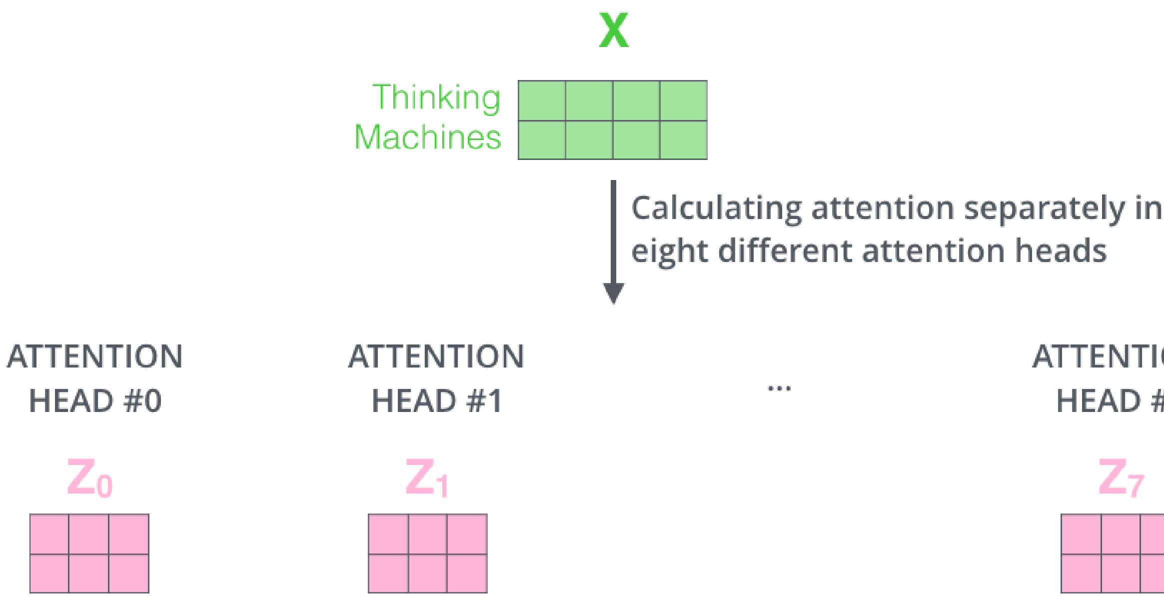




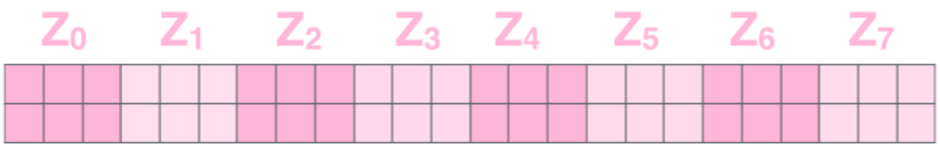
Self attention review (III): Multihead



Self attention review (IV): Multihead

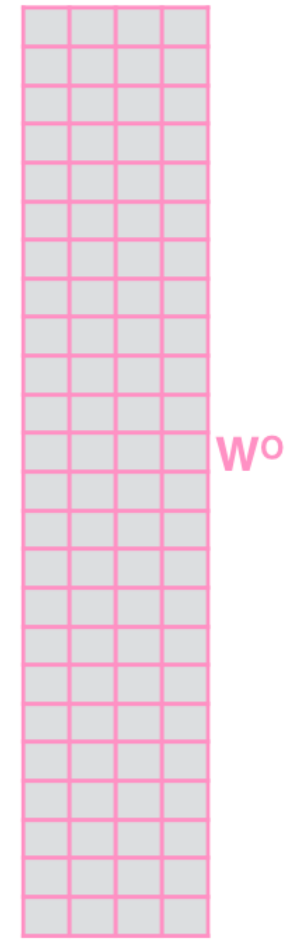


1) Concatenate all the attention heads



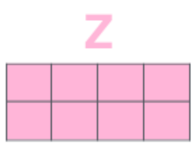
2) Multiply with a weight matrix W^O that was trained jointly with the model

X

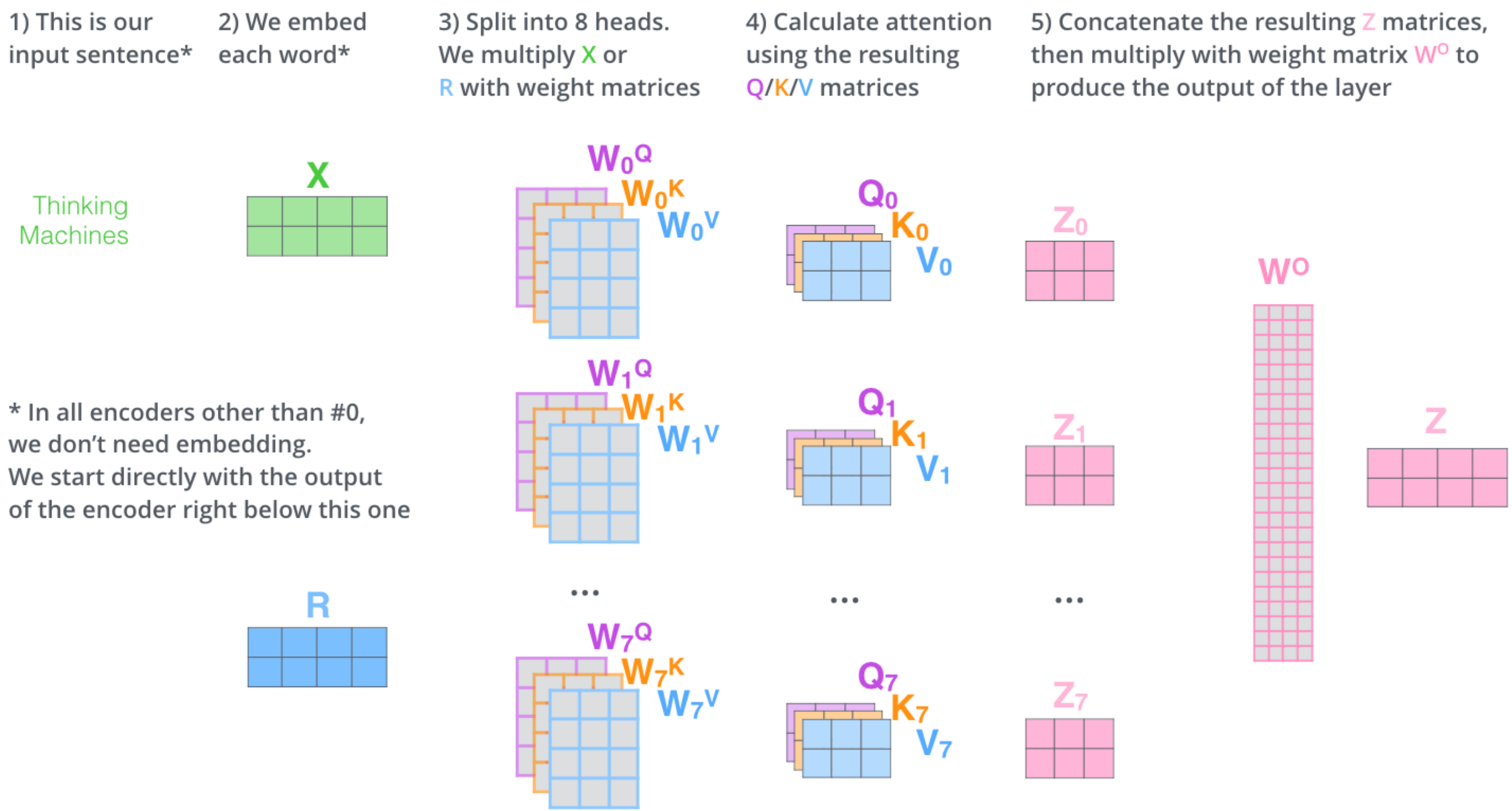


3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

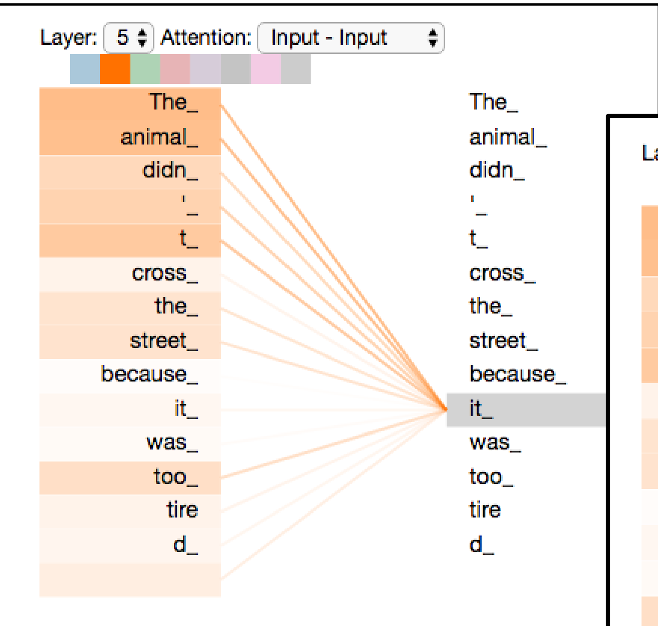
=



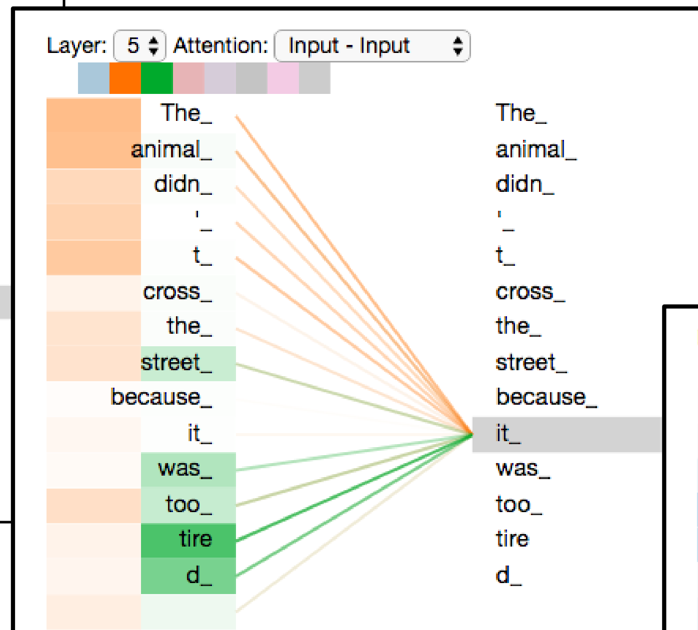
Self-attention: Summary



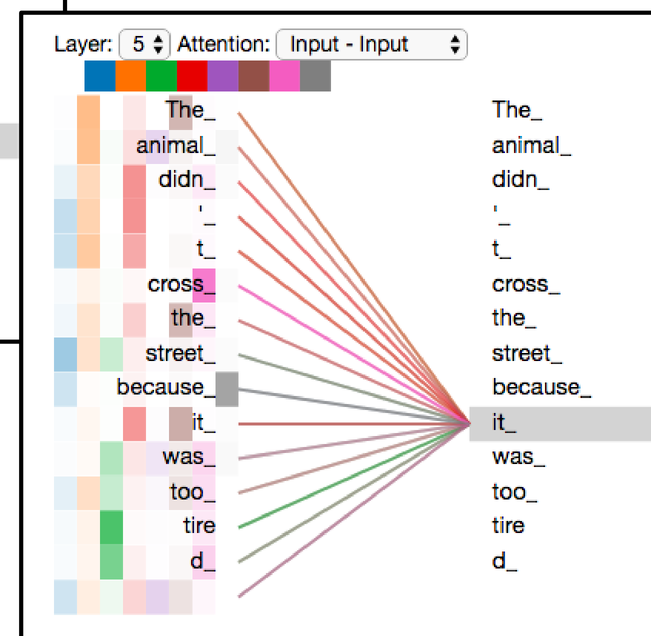
One head



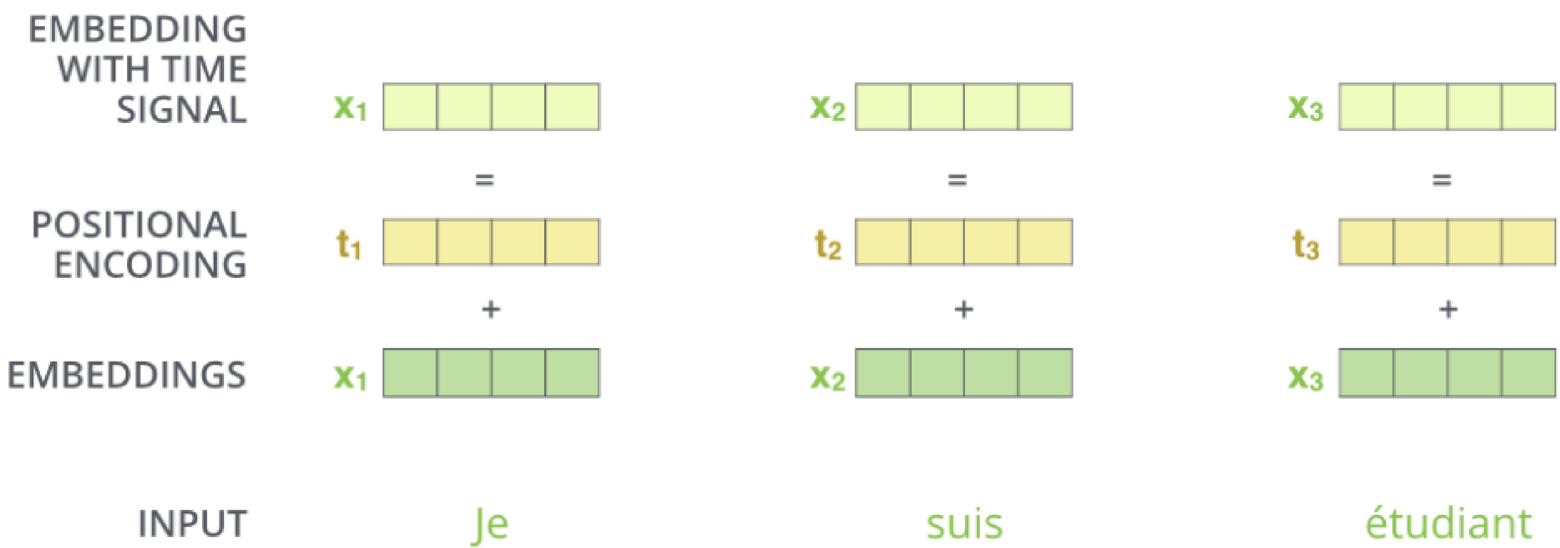
Two heads



All heads



Representing The Order of The Sequence Using Positional Encoding



But the Transformer architecture ditched the recurrence mechanism in favor of multi-head self-attention mechanism. Avoiding the RNNs' method of recurrence will result in massive speed-up in the training time. As each word in a sentence simultaneously flows through the Transformer's encoder/decoder stack, The model itself doesn't have any sense of position/order for each word.

Using Positional Encoding

- The first idea that might come to mind is to assign a number to each time-step within the [0, 1] range in which 0 means the first word and 1 is the last time-step.
  - Problem: you can’t figure out how many words are in the sentence.

- Another idea is to assign a number to each time-step linearly. That is, the first word is given “1”, the second word is given “2”, and so on.
  - Problem: values could get quite large, but also our model can face sentences longer than the ones in training.

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

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

Geometric progression.

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

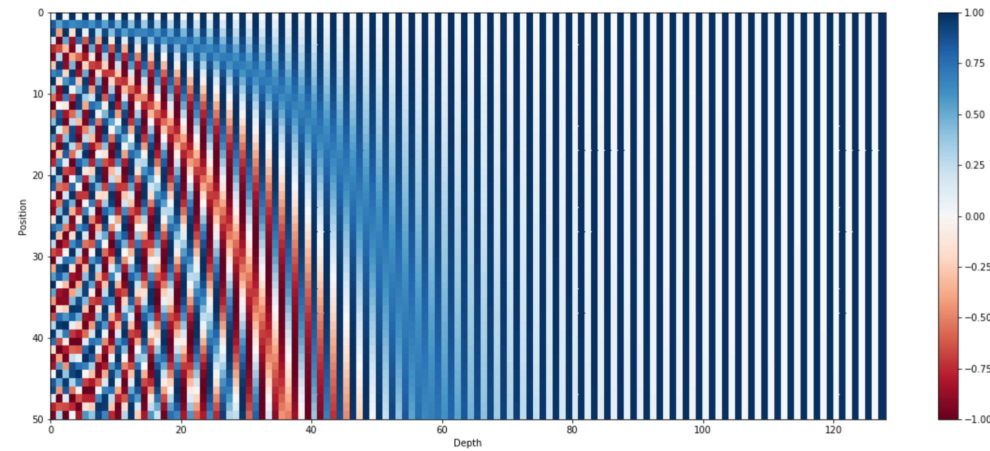
Using Positional Encoding

- Let t be the desired position in an input sentence, $p_t \in \mathbb{R}^d$ the positional encoding and d the encoding dimension. Then $f: \mathbb{N} \to \mathbb{R}^d$ will be the function that produces the output vector p_t.

Intuition

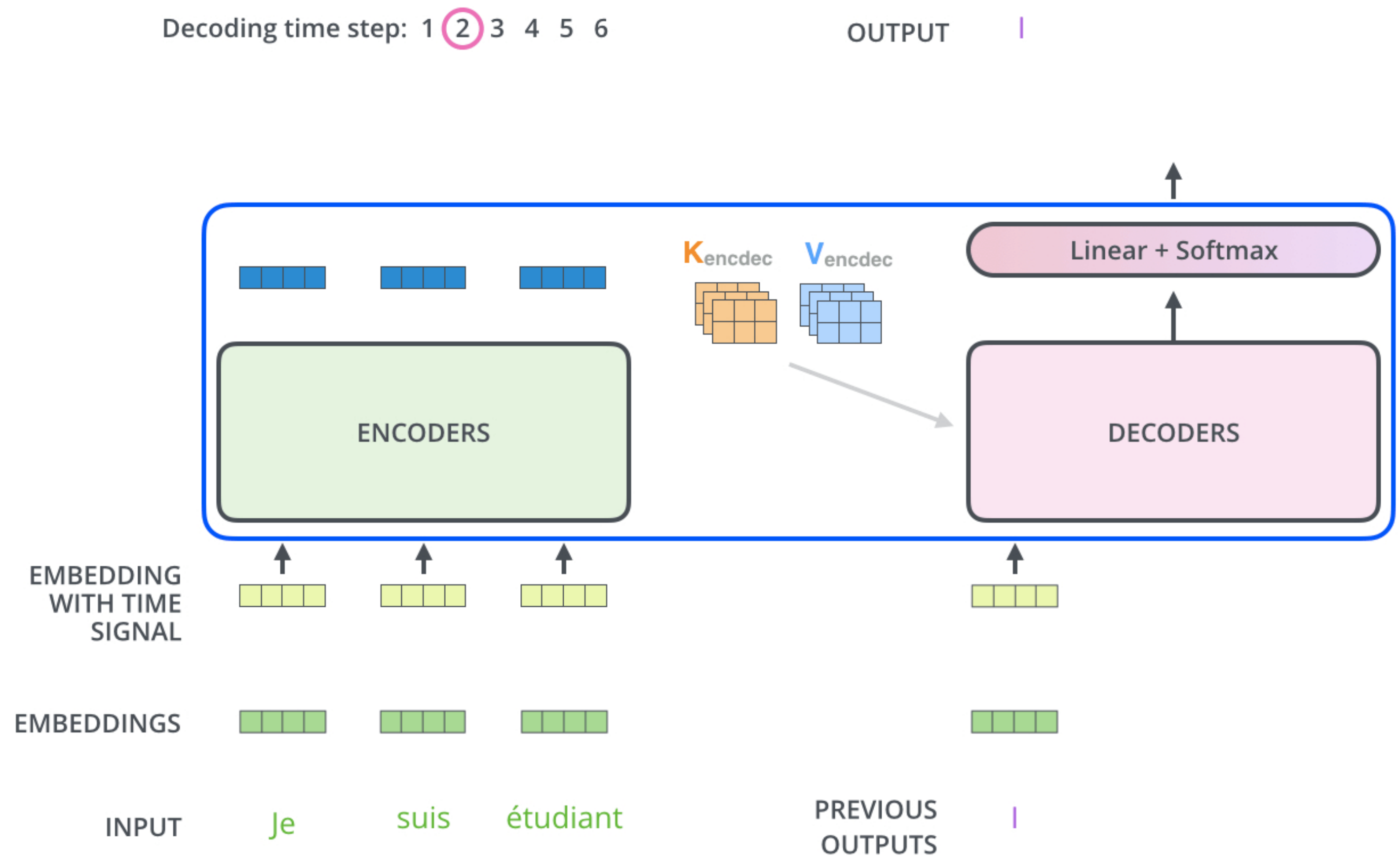
0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

LSB bit is alternating on every number, the second-lowest bit is rotating on every two numbers, and so on.

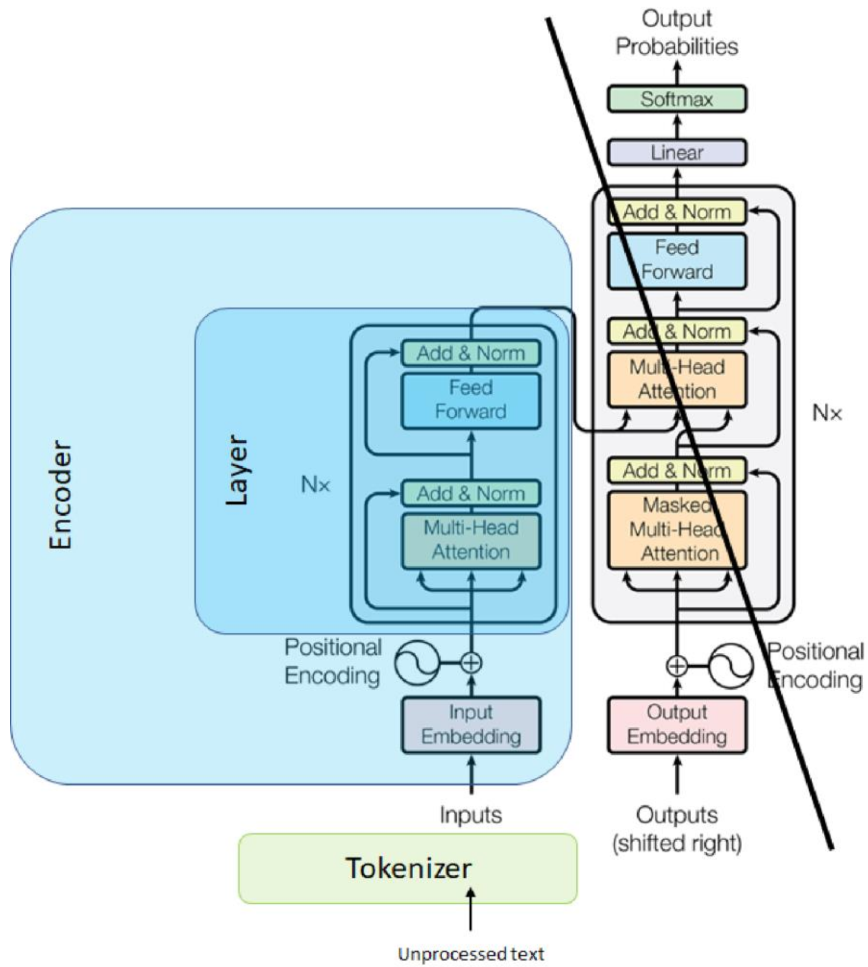


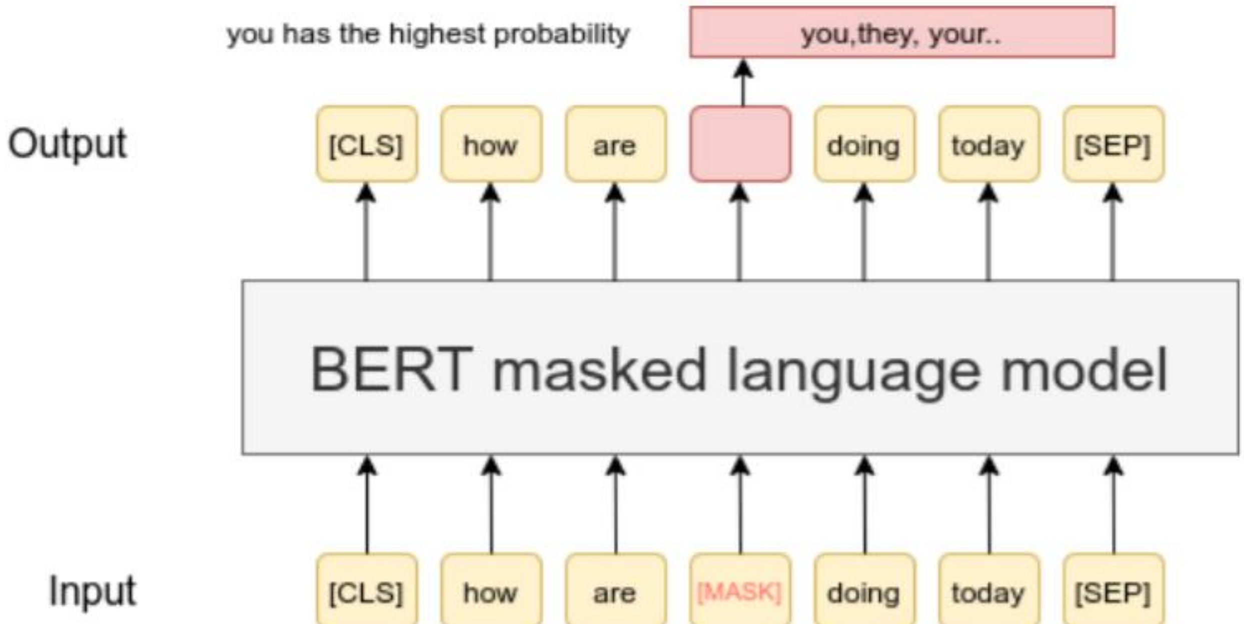
The 128-dimensional positional encoding for a sentence with the maximum length of 50.

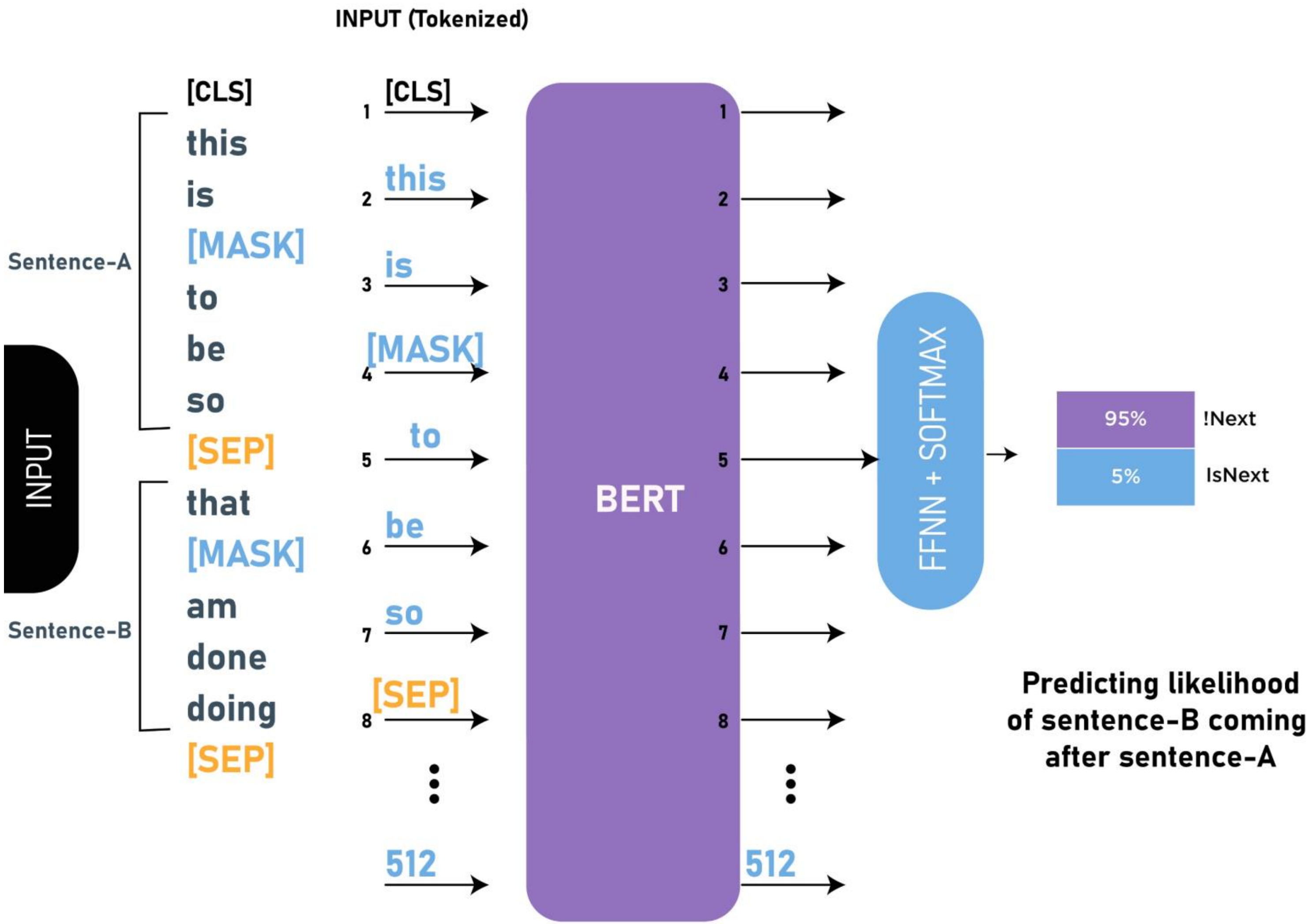
And the decoder?



<u>In the decoder, the self-attention layer is only allowed to attend to earlier positions in the output sequence.</u>


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Positional Encoding
Output Embedding
Outputs (shifted right)
Encoder
Layer
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Positional Encoding
Input Embedding
Inputs
Tokenizer
Unprocessed text


you has the highest probability
you,they, your..
Output
[CLS]
how
are
doing
today
[SEP]
BERT masked language model
Input
[CLS]
how
are
[MASK]
doing
today
[SEP]


INPUT (Tokenized)
INPUT
Sentence-A
Sentence-B
[CLS]
this
is
[MASK]
to
be
so
[SEP]
that
[MASK]
am
done
doing
[SEP]
1 [CLS]
2 this
3 is
4 [MASK]
5 to
6 be
7 so
8 [SEP]
512
BERT
1
2
3
4
5
6
7
8
512
FFNN + SOFTMAX
95% !Next
5% IsNext
Predicting likelihood of sentence-B coming after sentence-A

Original paper

- https://ai.googleblog.com/2017/06/accelerating-deep-learning-research.html
  - Tensor2Tensor
- https://arxiv.org/pdf/1706.03762.pdf

Gracias por la atención

¿Tiene alguna pregunta?