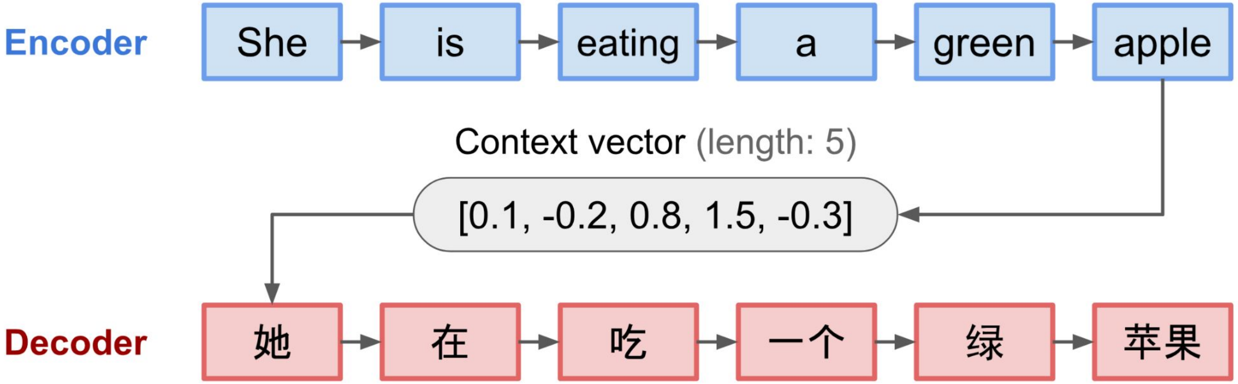
Attention and Transformer

Deep Learning

Aziz Temirkhanov
Lambda, HSE

Seq2Seq



- An **encoder** processes the input sequence and compresses the information into a context vector (also known as sentence embedding or “thought” vector) of a fixed length. This representation is expected to be a good summary of the meaning of the whole source sequence.
- A **decoder** is initialized with the context vector to emit the transformed output. The early work only used the last state of the encoder network as the decoder initial state.

Attention

Previously, we only considered NN with hidden activations as a linear combination of the input activations, followed by a nonlinearity: $\mathbf{Z} = \varphi(\mathbf{XW})$, where $\mathbf{X} \in \mathbb{R}^{m \times v}$

And our goal was to find a best **fixed W** for input data. But imagine if we could have more flexible model in which the weight is depends on the inputs: $\mathbf{Z} = \varphi(\mathbf{XW}(\mathbf{X}))$

This kind of multiplicative interaction is called **attention**

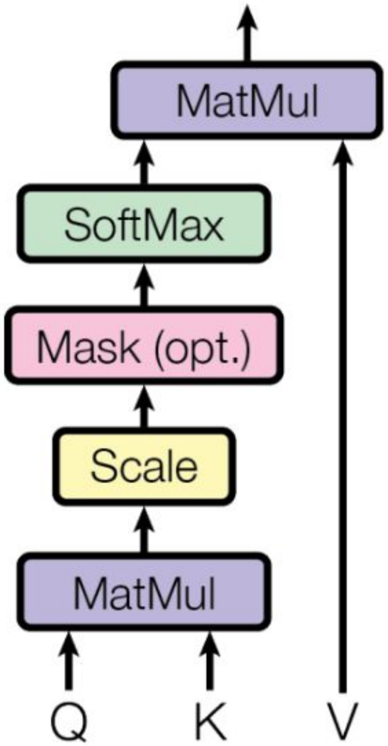
Attention

- Assume $\mathbf{Q} = \mathbf{W}_q\mathbf{X},\ \mathbf{K} = \mathbf{W}_k\mathbf{X},\ \text{and}\ \mathbf{V} = \mathbf{W}_v\mathbf{X}.$
- Then: $\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^{\mathsf{T}}}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v}$

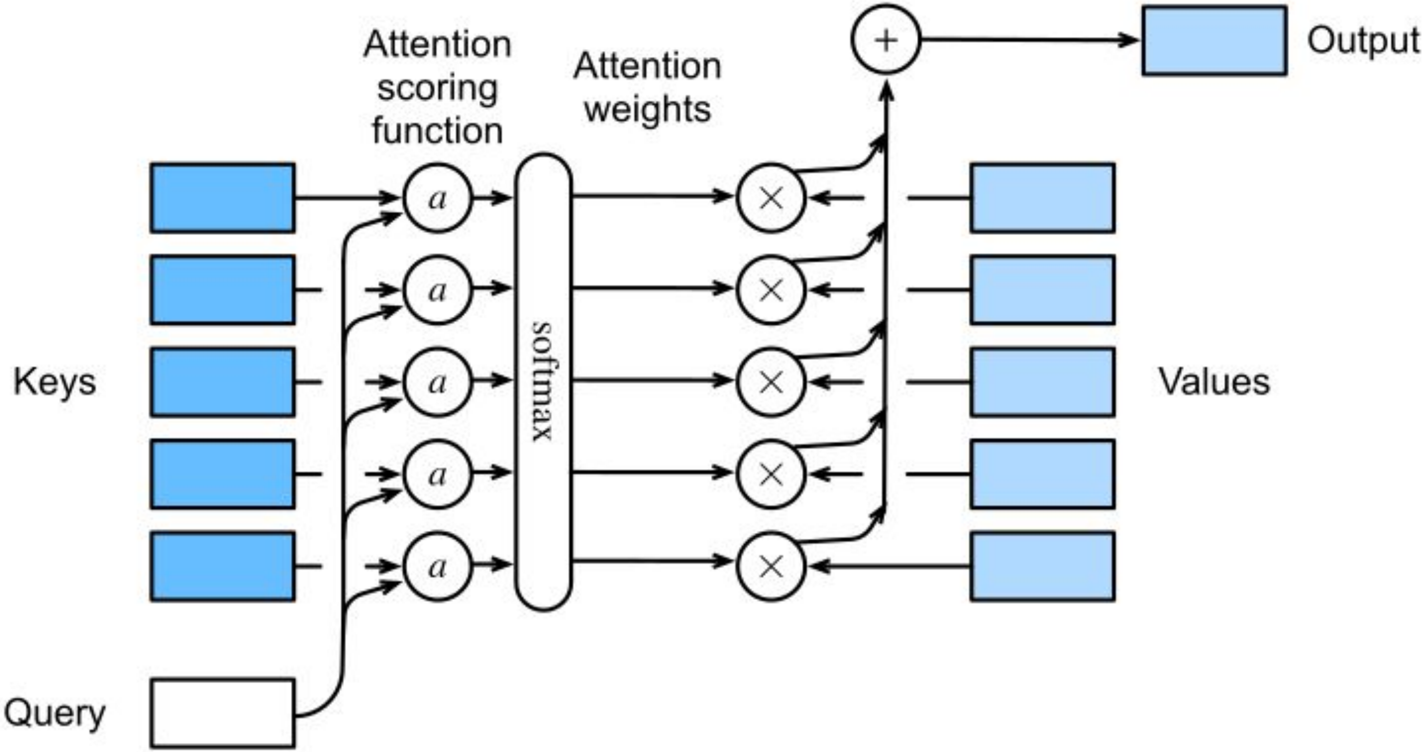
But what does it mean?

Attention

- Key, Value and Query terms is derived from retrieval systems. Imagine a search engine, for example
- When searching for some **query**, search engine will map it against the set of **keys** (page titles, html headers, tags, etc.) associated with any web page, then present you the best matched result (**values**)



Attention



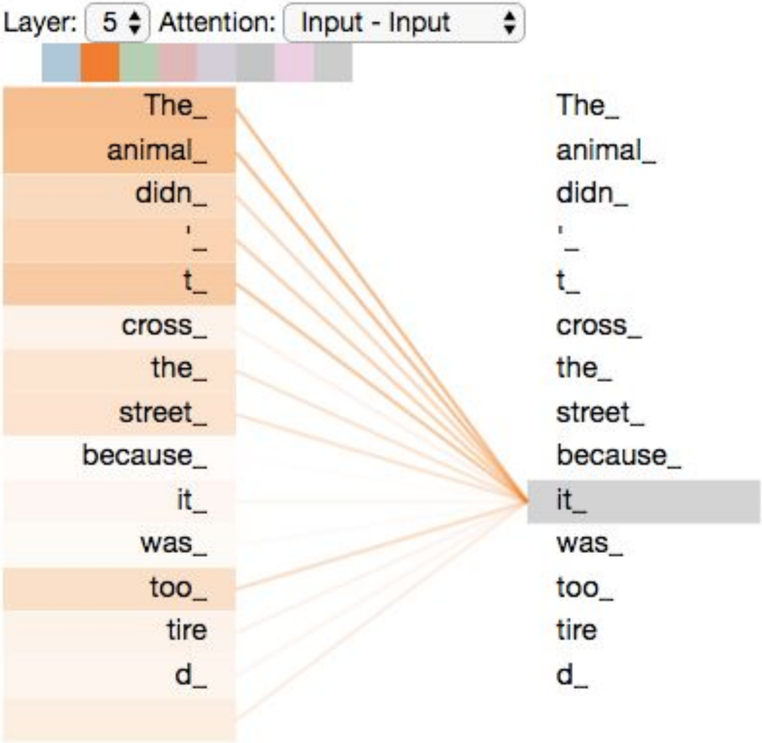
Attention scoring function
Attention weights
Output
Keys
Values
Query
softmax
a

Attention

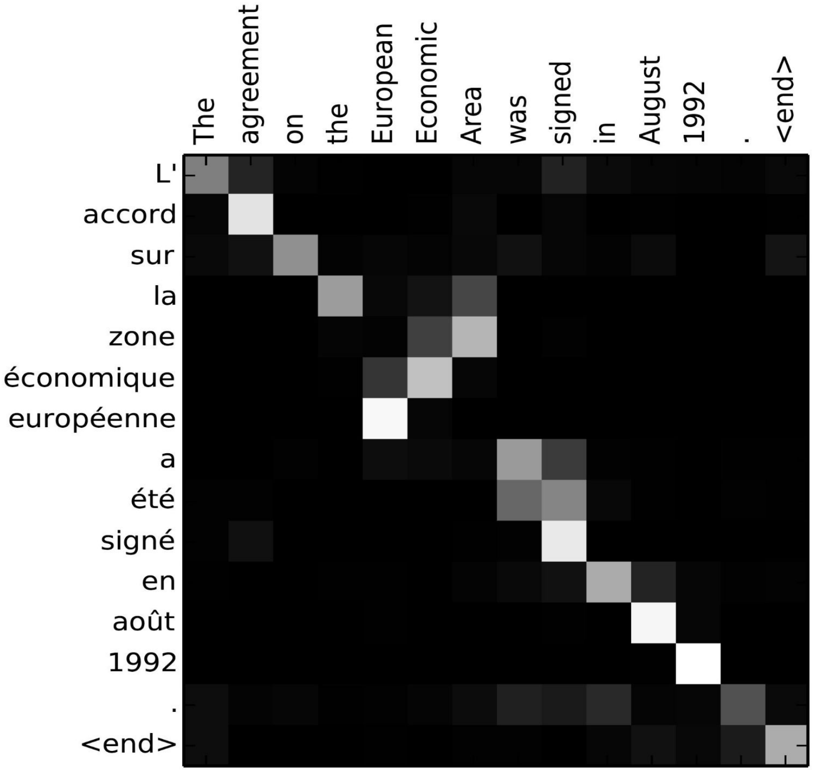
But why does it work?

1. Project inputs at a common space (obtaining **K** and **Q**)
2. Choose a similarity measure (dot product in out case)
    a. The more those vectores alike, the lesser the angle between them, so normed dot product is closer to 1
3. Obtain a similarity matrix by matching **keys** against a **query**
4. Normalize these vectors
5. Hardmax/softmax of step 4 then multiplies to a vector **V**

Attention

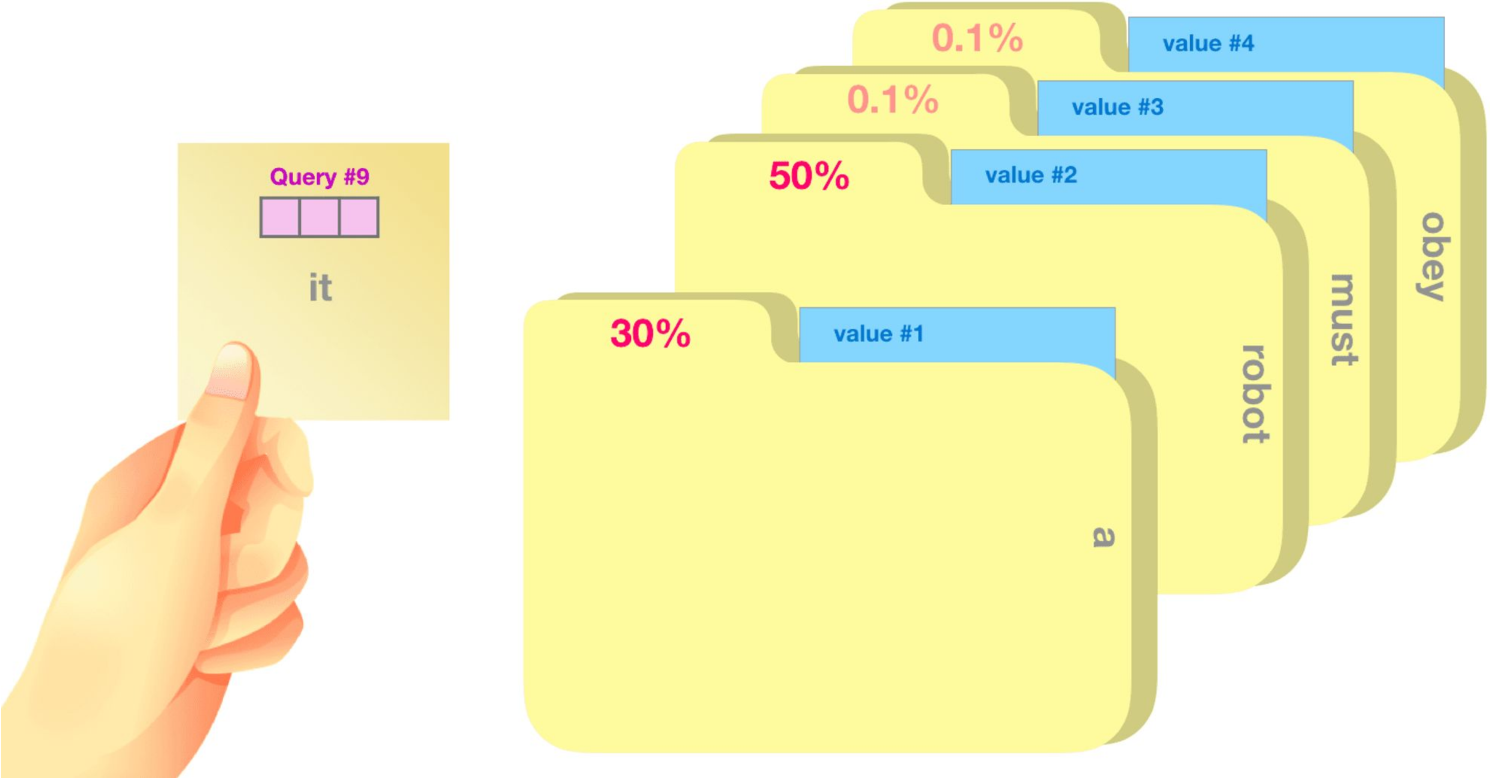


Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

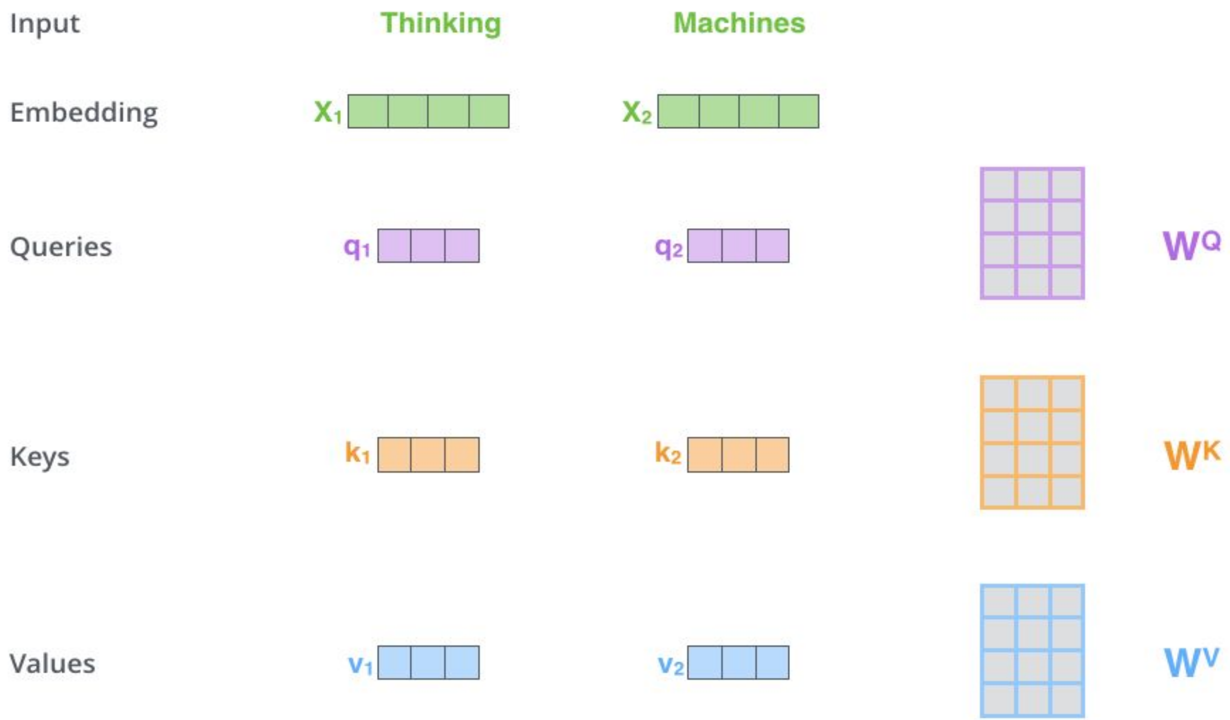


The
agreement
on
the
European
Economic
Area
was
signed
in
August
1992
.
<end>
L'
accord
sur
la
zone
économique
européenne
a
été
signé
en
août
1992
.
<end>

Attention

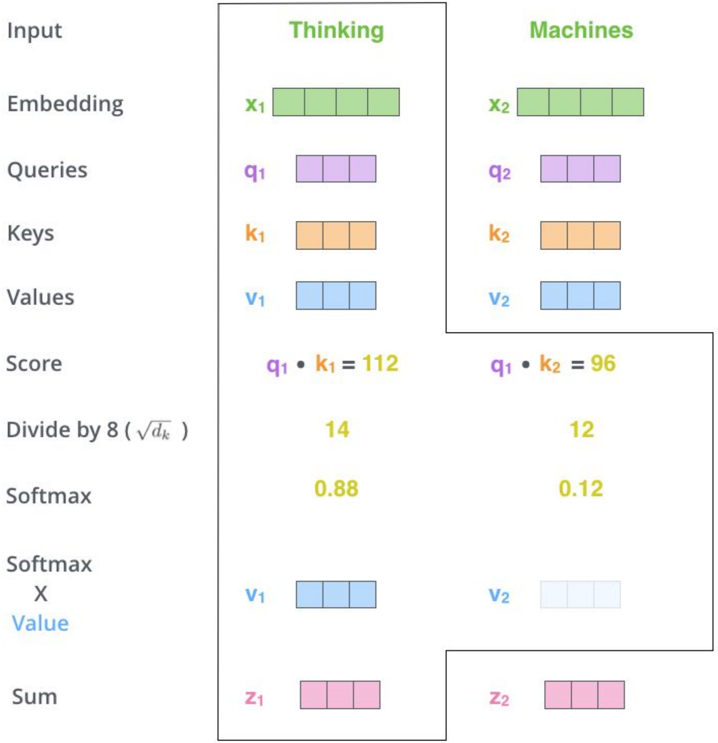

Query #9
it
0.1%
value #4
obey
0.1%
value #3
must
50%
value #2
robot
30%
value #1
a

Self-Attention



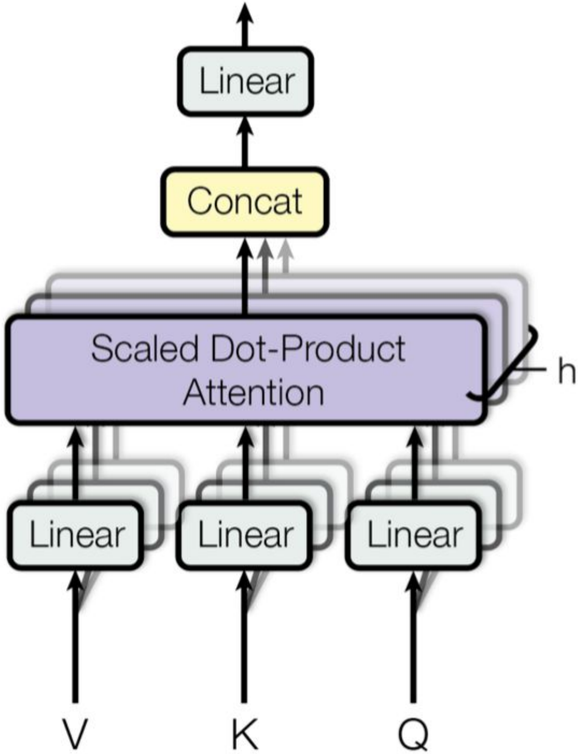
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Self-Attention



- For a vector **X**, compute **Q, K,** and **V** by multiplying learnable matrices $\mathbf{W}_{\mathbf{Q/K/V}}$
- For a fixed $\boldsymbol{q_i}$, compute attention **score** by matching it against **K**
- Assemble result: retrieve a **value** with respect to its **score**

Multi-Head Self-Attention



$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)W^O$$
$$\text{where } \mathrm{head}_i = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

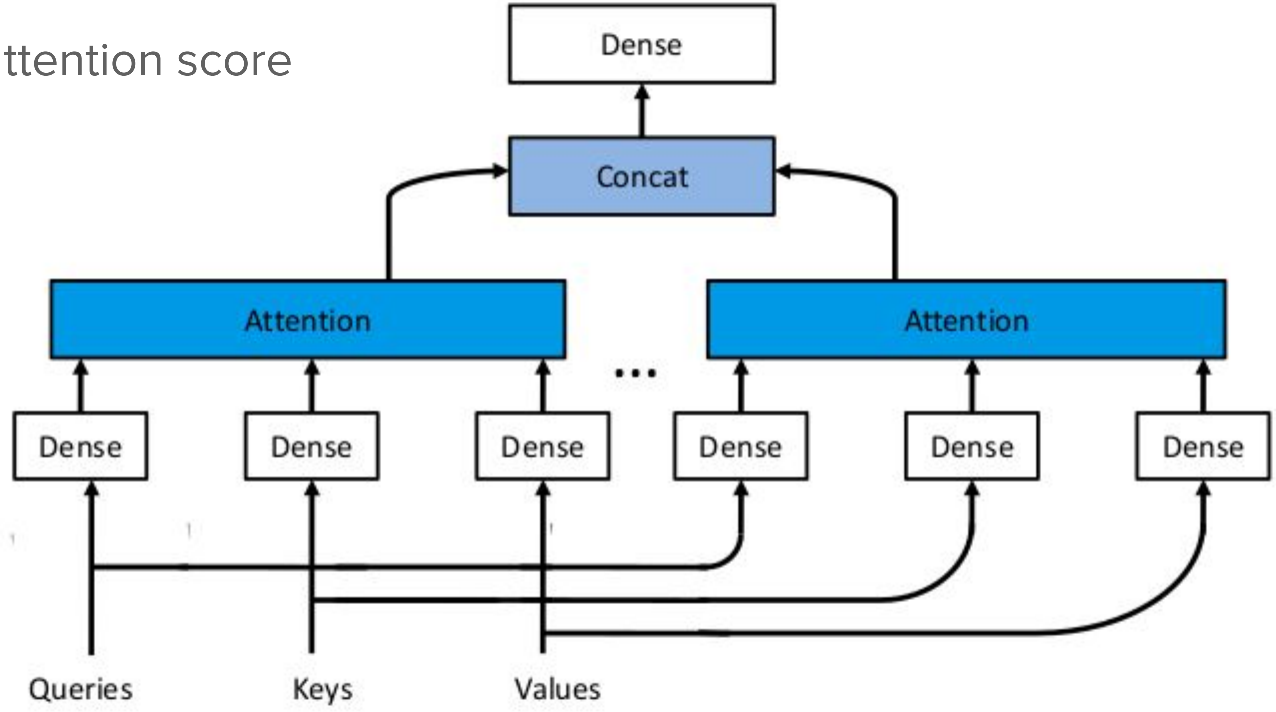
Where the projections are parameter matrices:

$$W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k},$$
$$W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k},$$
$$W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v},$$
$$W_i^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}.$$

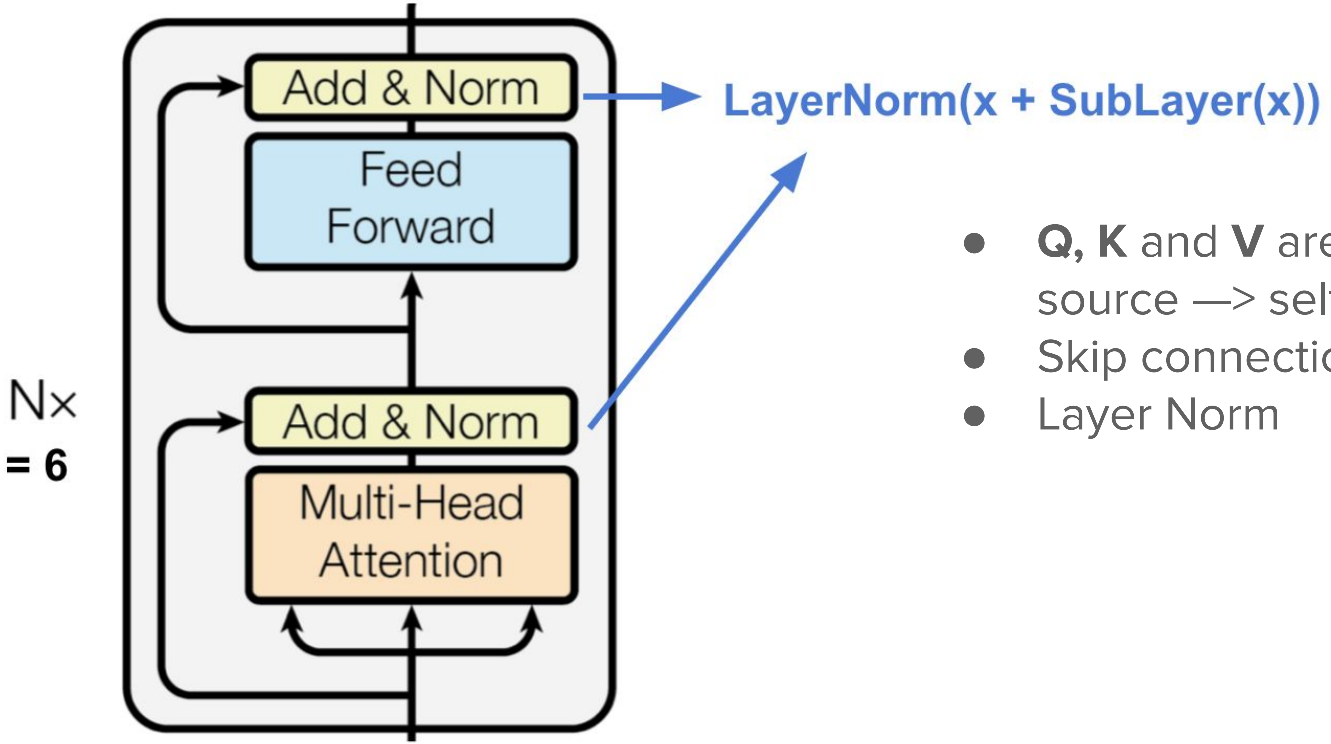
$$\boldsymbol{h} = \mathrm{MHA}(\boldsymbol{q}, \{\boldsymbol{k}_j, \boldsymbol{v}_j\}) = \mathbf{W}_o \begin{pmatrix} \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_h \end{pmatrix} \in \mathbb{R}^{p_o}$$

MHA

- Easy to parallel
- Compute different attention score for each head



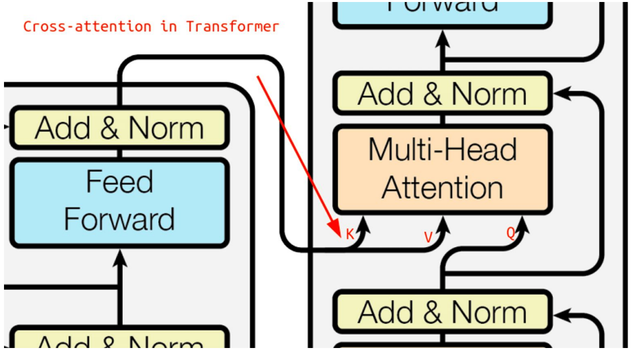
Encoder

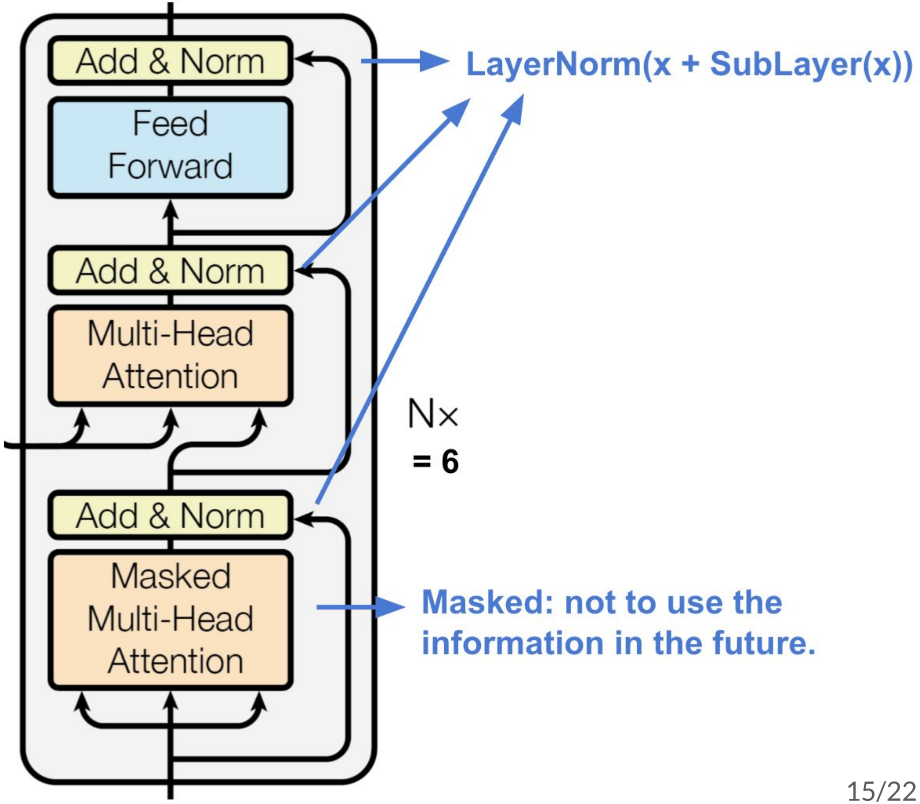


- **Q, K** and **V** are derived from the same source —> self-attention
- Skip connection
- Layer Norm

Decoder

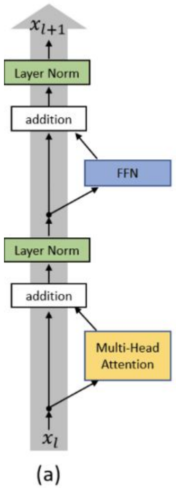
- Cross-Attention: **K** and **V** are coming from encoder, **Q** is from previous layer
- Masked MHA: assign large negative number to any token from the future

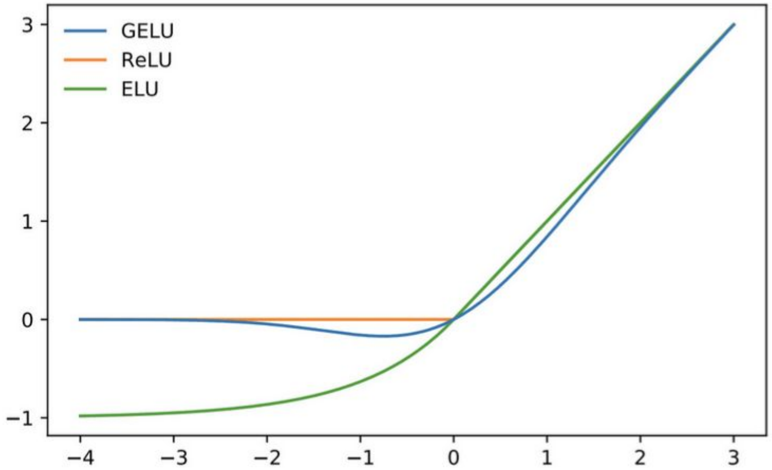




Feed Forward

- Feed Forward is a simple two-layer dense NN
- Using GELU activations
- Layer Normalization in PreLN regime: first, apply layer norm, then, concat the residuals

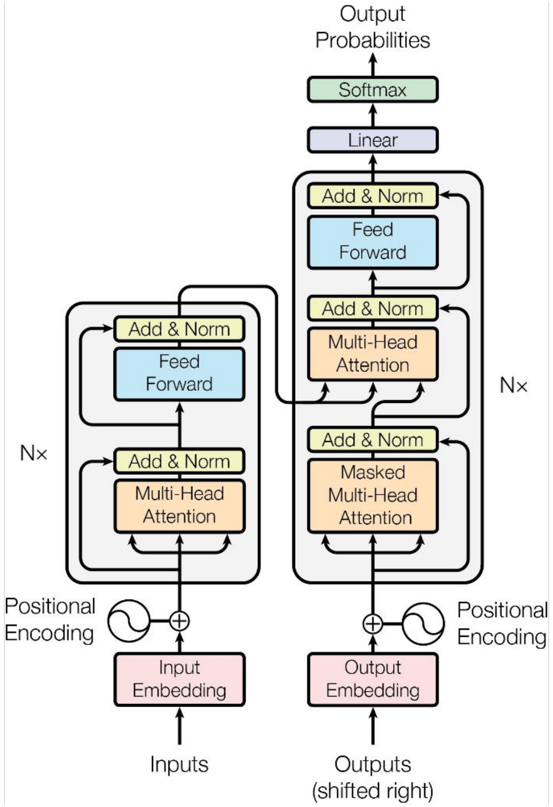




Transformer

- Combining all together, we obtaining the well known encoder-decoder transformer architecture
- One can use decoder part only, training the model is self-supervised regime (GPT)

- 6 enc-layers + 6 dec-layers
- H = 512
- H in FF = 4*512
- 8 heads
- 65M params



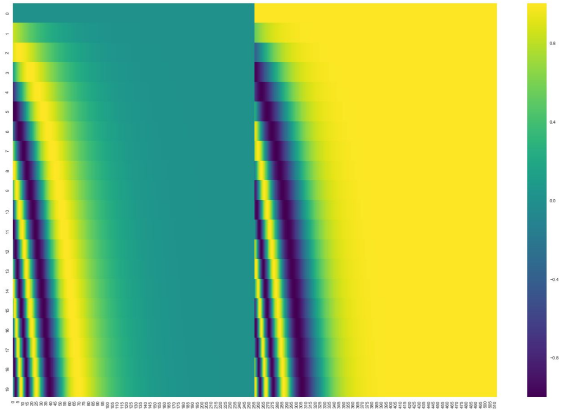
One Small Problem

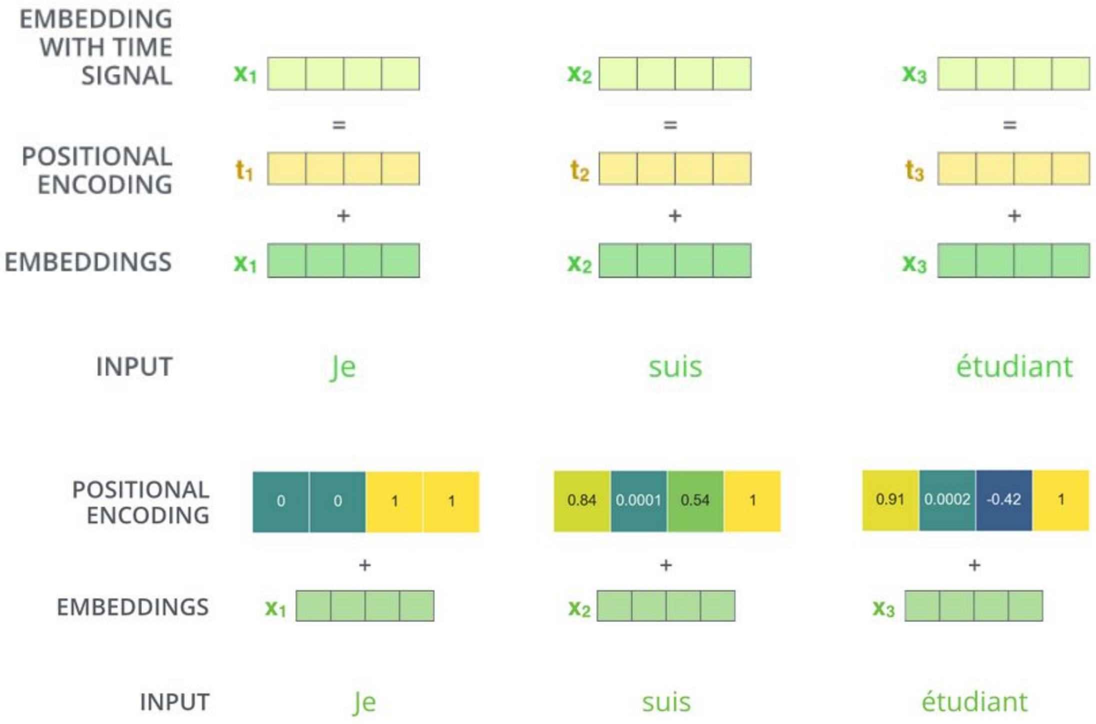
- We talked about attention and how it is affects the decision-making process via looking at certain parts of text
- But one problem is still present: attention is permutation invariant, and hence ignorance the input word ordering
- To solve this issue, let us introduce a **positional embedding**, and concat it with input sequence

Positional Encoding

Variant from the initial paper:

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





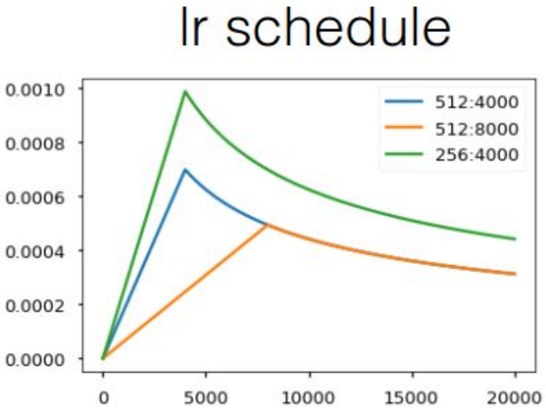
A real example of positional encoding with a toy embedding size of 4

Training details

- Cross-Entropy Loss $NLL(y_{1:M}) = -\sum_{t=1}^{M} \log p(y_t|t_{t-1})$
- Teacher Forcing for decoder
- Adam Optimizer
- lr with warm-up scheduling

$$lr = d_{model}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup^{-1.5})$$

- label smoothing $y_{ls} = (1-\alpha) \cdot y_{hot} + \alpha/K$
- Residual Dropout to the output of each sub-layer and to the sum of word embeddings and positional encoding
- BPE
- Model Averaging (average over last k checkpoints)



Output Generation

Greedy search: $y_t = \text{argmax}_{y \in \mathcal{Y}} P(y|y_1, \ldots, y_{t-1}, C)$

Result of greedy search

Time step	1	2	3	4
A	**0.5**	0.1	0.2	0.0
B	0.2	**0.4**	0.2	0.2
C	0.2	0.3	**0.4**	0.2
<eos>	0.1	0.2	0.2	**0.6**

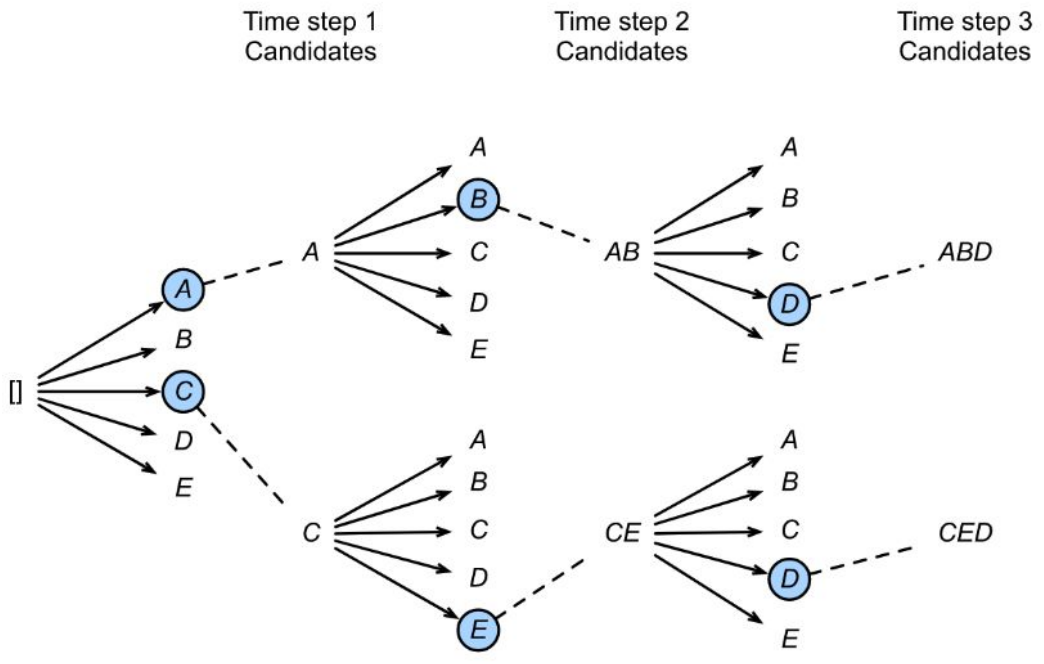
$$P = 0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$$

Better output sequence

Time step	1	2	3	4
A	**0.5**	0.1	0.1	0.1
B	0.2	0.4	**0.6**	0.2
C	0.2	**0.3**	0.2	0.1
<eos>	0.1	0.2	0.1	**0.6**

$$P = 0.5 \times 0.3 \times 0.6 \times 0.6 = 0.054$$

Beam Search



- At each step choose k best candidates (k=~5)
- Stop when k candidates with <END> token have been generated
- Choose the best one:

$$\boxed{\frac{1}{L^{\alpha}}} \log P(y_1, \ldots, y_L | C)$$

length penalty,
alpha = 0.75