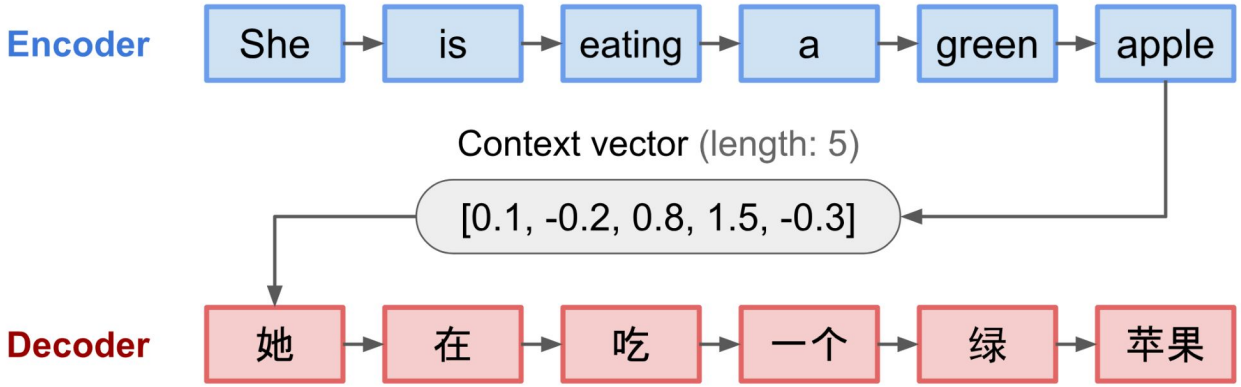
Attention and Transformer

Deep Learning

Aziz Temirkhanov
Lambda, HSE

Seq2Seq



- **Энкодер** обрабатывает входную последовательность и сжимает информацию в вектор контекста фиксированной длины (эмбеддинг, латентное представление или вектор смысла). Ожидается, что это представление является хорошей суммаризацией смысла всей вхожной последовательности.
- **Декодер** инициализируется этим вектором контекста, чтобы в дальнейшем преобразовать его в выход. Ранние работы использовали только последнее состояние энкодера как начальное для декодера.

Attention

Ранее, мы рассматривали нейросети как линейную комбинацию активаций, за которым шла нелинейность: $\mathbf{Z} = \varphi(\mathbf{X}\mathbf{W}),\ \text{where}\ \mathbf{X} \in \mathbb{R}^{m \times v}$

И задачей было найти **фиксированный W** для входных данных. Но что если нам придумать способ с более гибкими весами, которые будут зависеть от входных данных: $\mathbf{Z} = \varphi(\mathbf{X}\mathbf{W}(\mathbf{X}))$

Такого рода активации называются **вниманием (attention)**

Attention

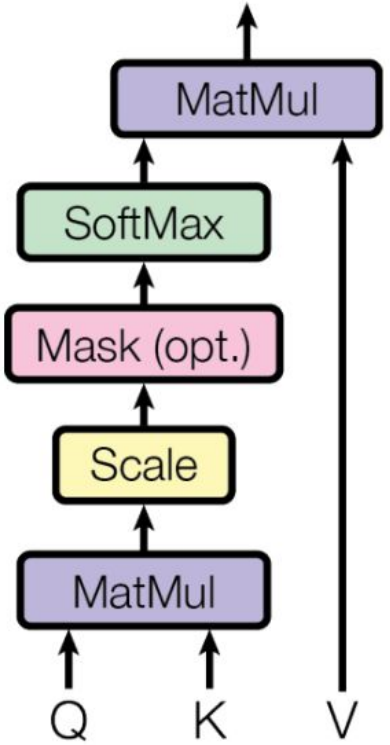
- Допустим $\mathbf{Q} = \mathbf{W}_q\mathbf{X},\ \mathbf{K} = \mathbf{W}_k\mathbf{X},\ \text{and}\ \mathbf{V} = \mathbf{W}_v\mathbf{X}.$

- Тогда: $\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^\mathsf{T}}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v}$

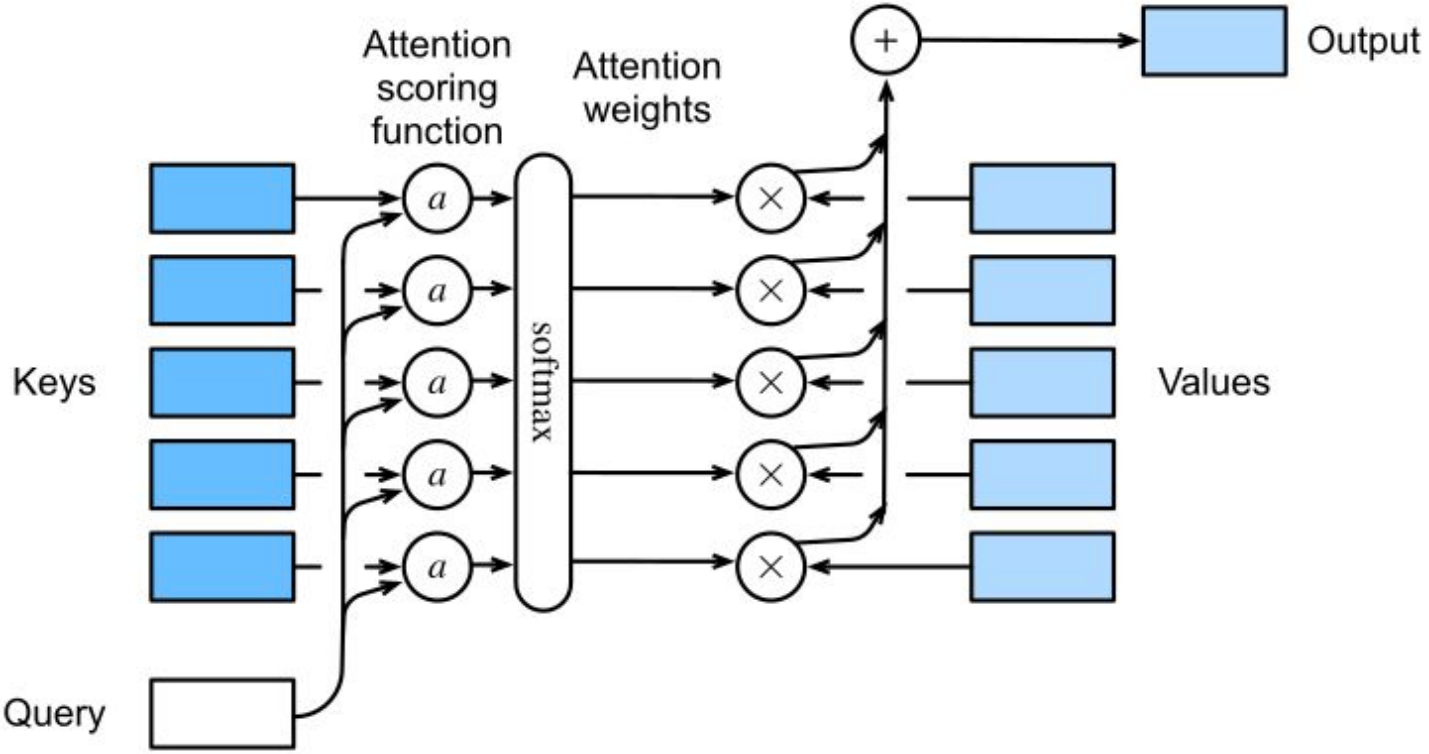
Но что это означает?

Attention

- Key, Value и Query terms (Ключ, Значение, Запрос) пришли к нам из поисковых систем. Представьте себе поисковый движок, для примера
- Когда мы ищем результат для некоторого **запроса (query)**, поисковик будет сравнивать его отображение с множеством **ключей** (**keys**), которые связаны со страницами, а затем предоставлять вам лучшее совпадение **значений** (**values**)



Attention

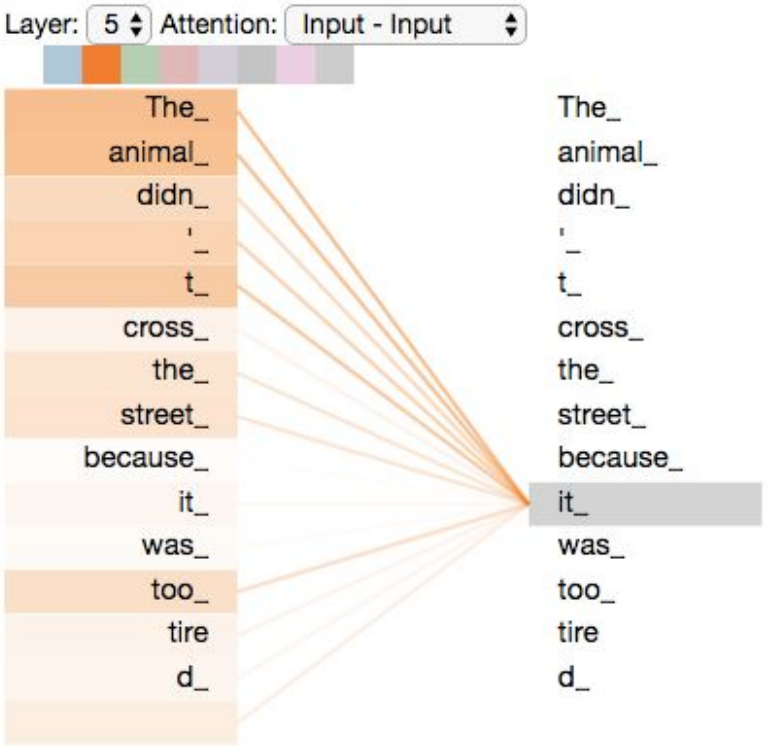

Attention scoring function
Attention weights
Output
Keys
Values
Query
a
softmax
+

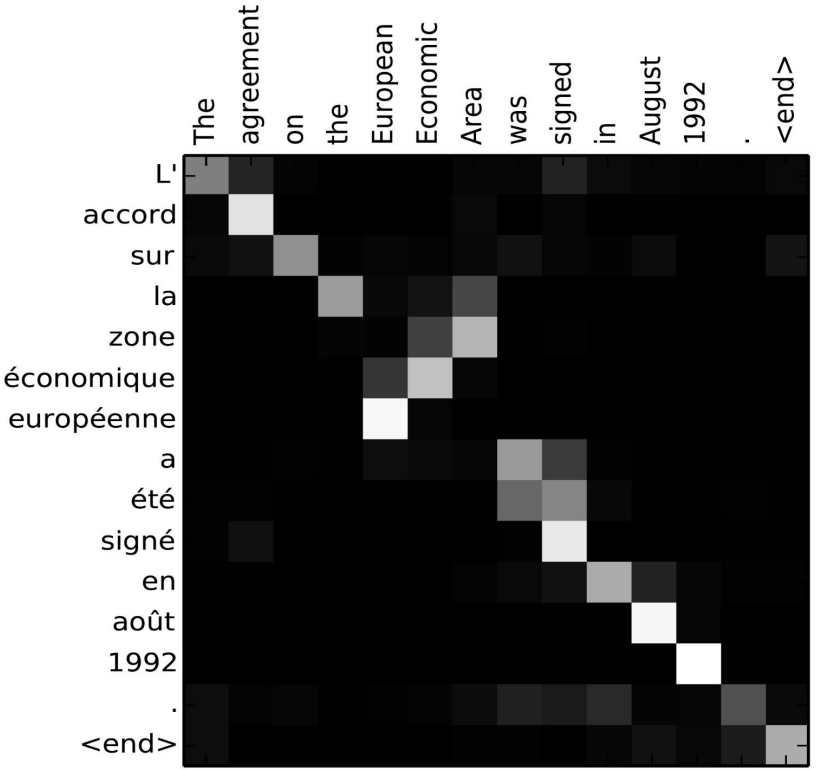
Attention

Как это работает?

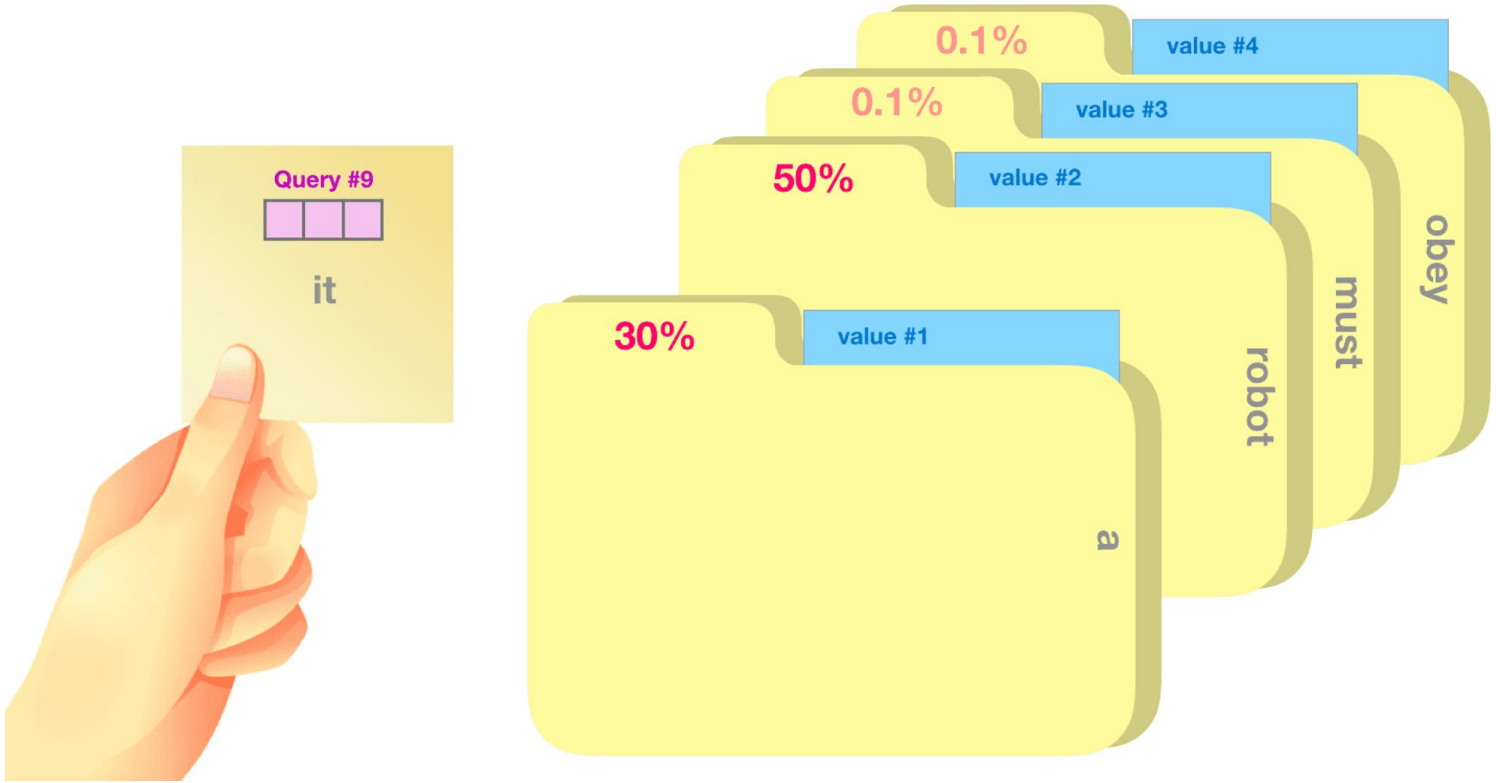
1. Сначала данные проецируются в одно и то же пространство (получаем **K** и **Q**)
2. Выбирается метрика похожести (dot product/скалярное произведение в нашем случае)
    - Чем больше наши вектора похожи друг на друга (запрос на ключ), тем меньше угол между ними, а значит, нормированная скалярное произведение стремиться к 1
3. Получаем матрицу схожести с помощью скалярного произведения запроса и всех ключей
4. Нормализуем (делим на квадратный корень из величины ключей)
5. Интерпретируем эти веса с помощью софтмакс (получаем вектор вероятностей), затем умножаем на вектор значений (выбираем куда смотреть)

Attention

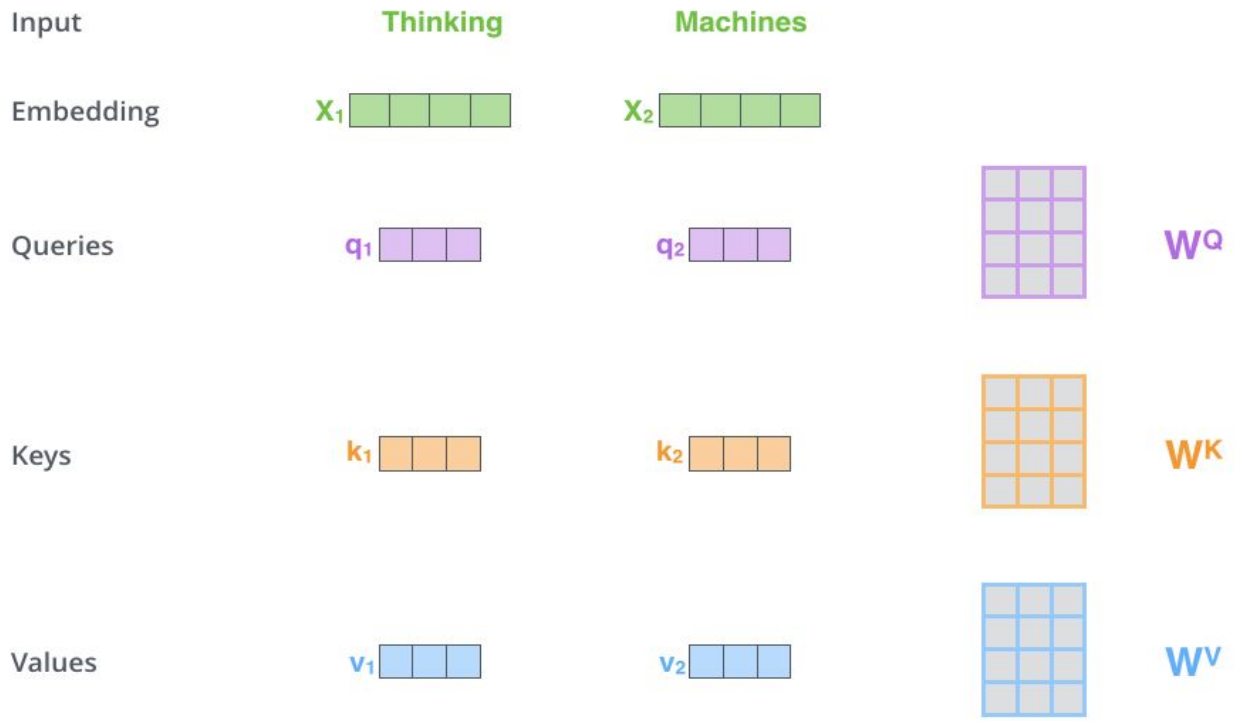

Layer: 5 Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_


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

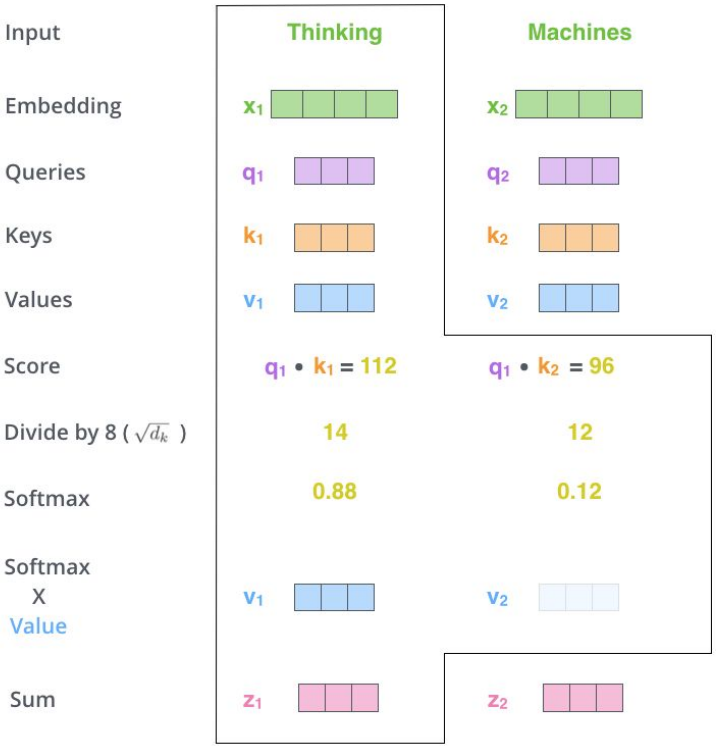

Query #9
it
0.1%
value #4
obey
0.1%
value #3
must
50%
value #2
robot
30%
value #1
a

Self-Attention



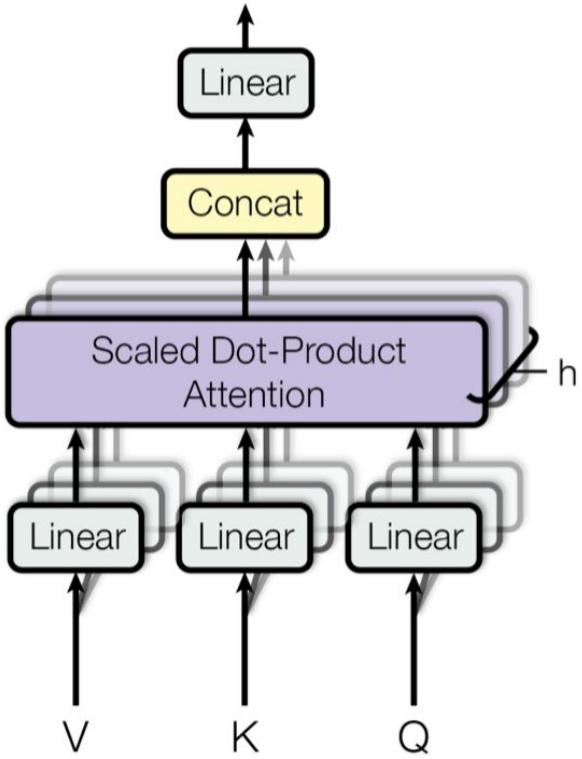
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Self-Attention



- For a vector **X**, compute **Q, K,** and **V** by multiplying learnable matrices $\mathbf{W_{Q/K/V}}$
- 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



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

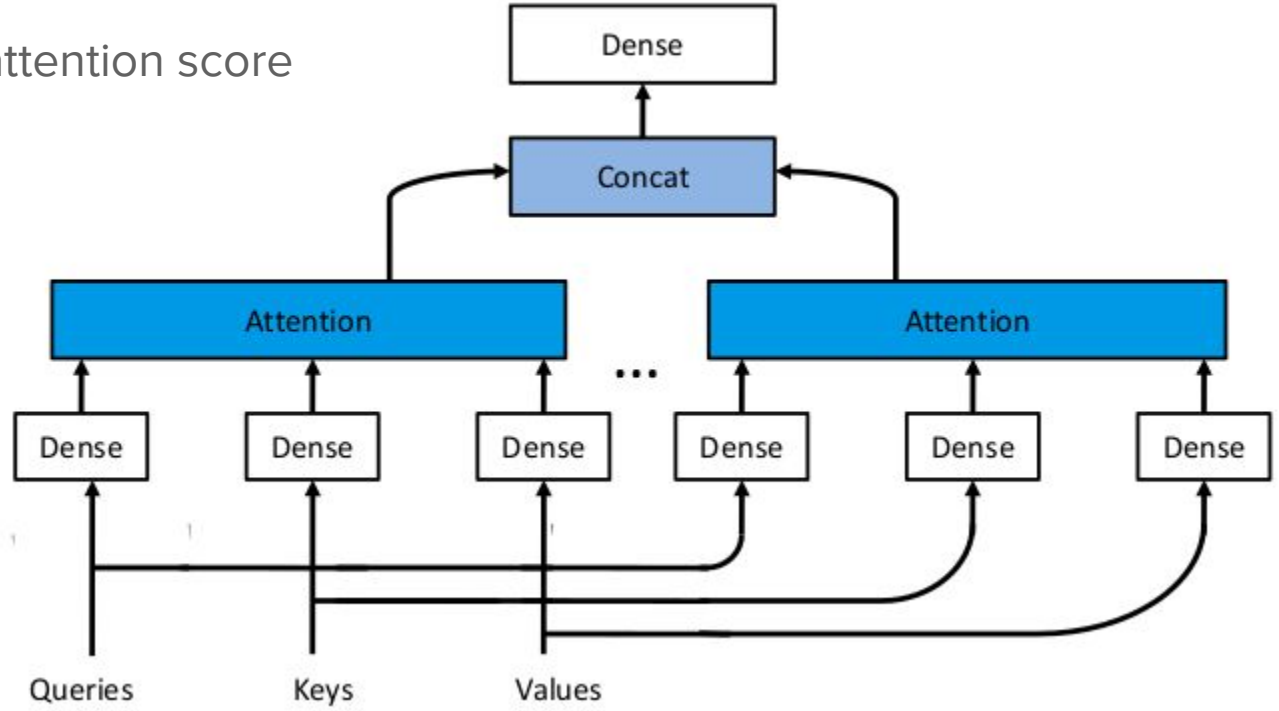
Where the projections are parameter matrices:

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

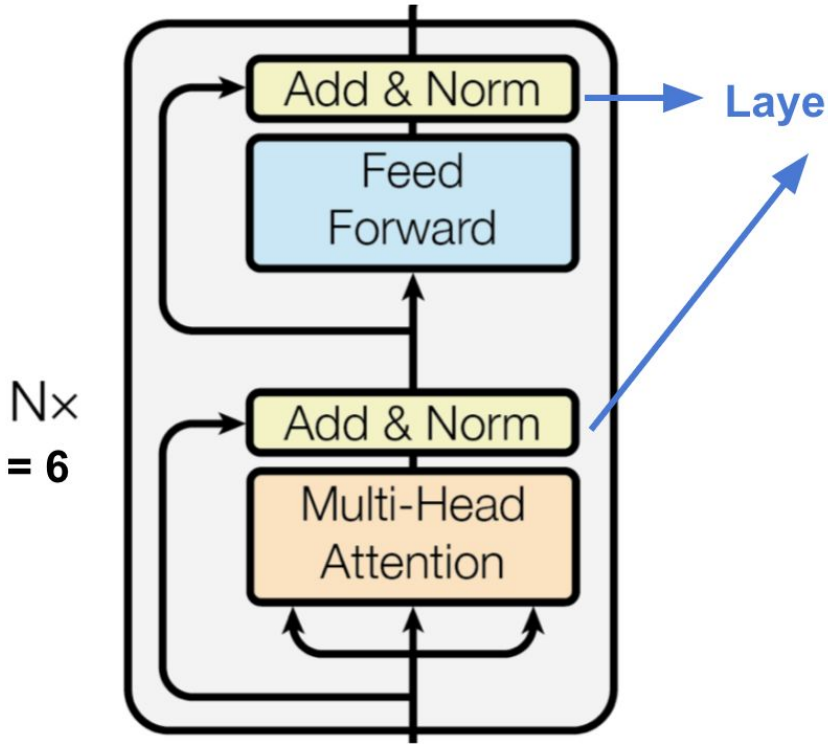
$$\boldsymbol{h} = \text{MHA}(\boldsymbol{q}, \{\boldsymbol{k}_j, \boldsymbol{v}_j\}) = \mathbf{W}_o \begin{pmatrix} \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_h \end{pmatrix} \in \mathbb{R}^{p_o}$$

MHA

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



Encoder

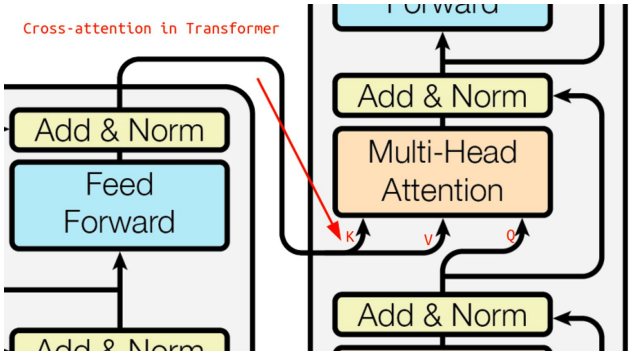


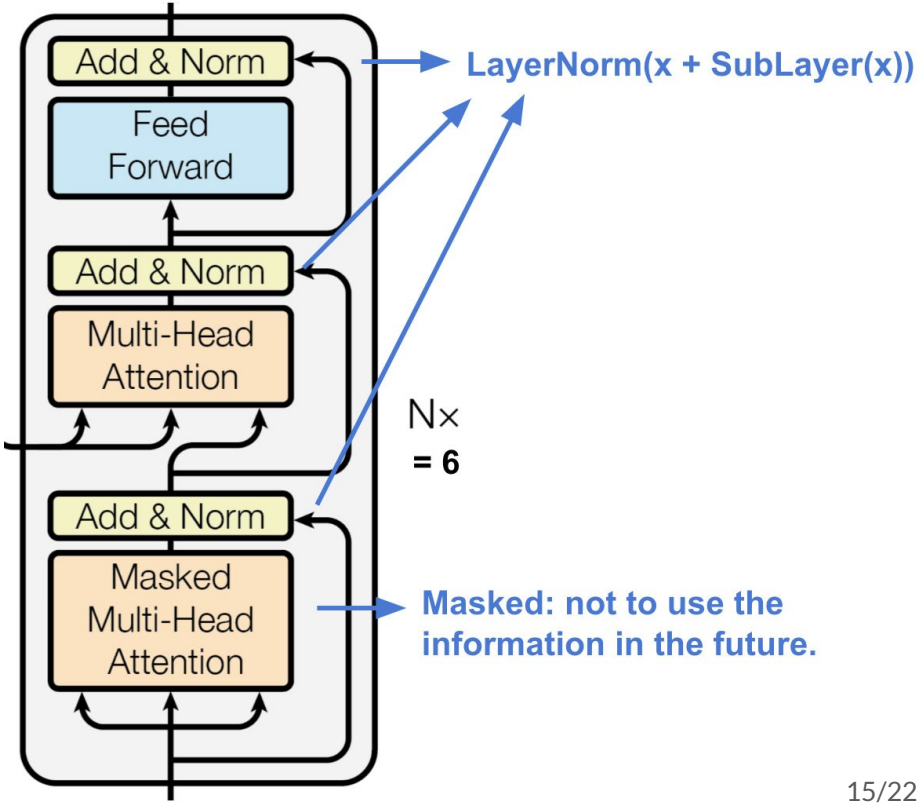
LayerNorm(x + SubLayer(x))

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

Decoder

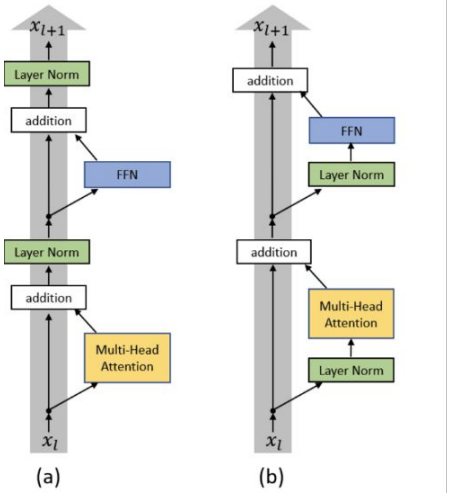
- Cross-Attention: **K** and **V** are coming from encoder, **Q** is from previous layer
- Masked MHA: assign large negative number to any token from the future

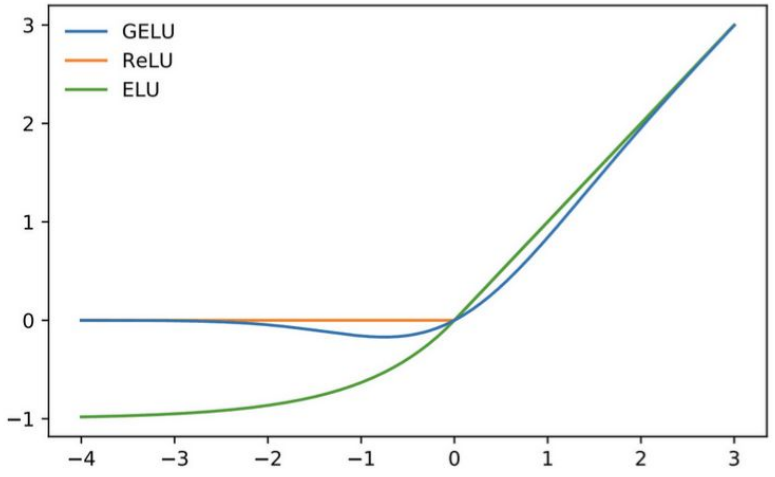




Feed Forward

- Feed Forward — простая двухслойная dense сетка
- Используются GELU активации
- Layer Normalization in PreLN regime: сначала нормализуем, потом делаем скип коннекшн

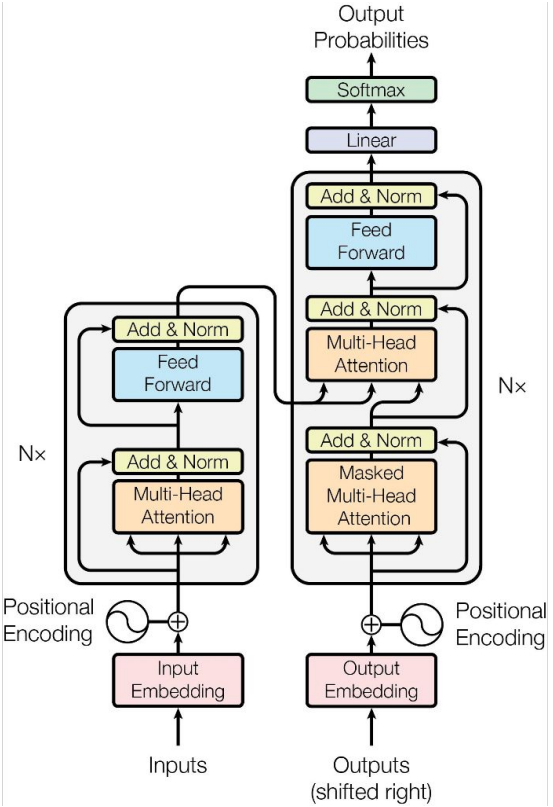




Transformer

- Объединяя все вместе, получаем известную архитектуру трансформера с режимом энкодер-декодер
- Можно использовать только декодер (ГПТ)

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



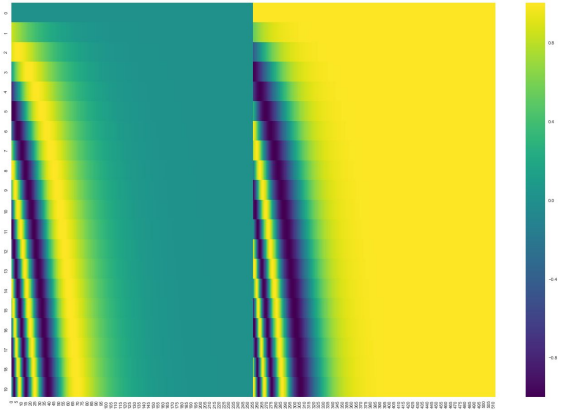
One Small Problem

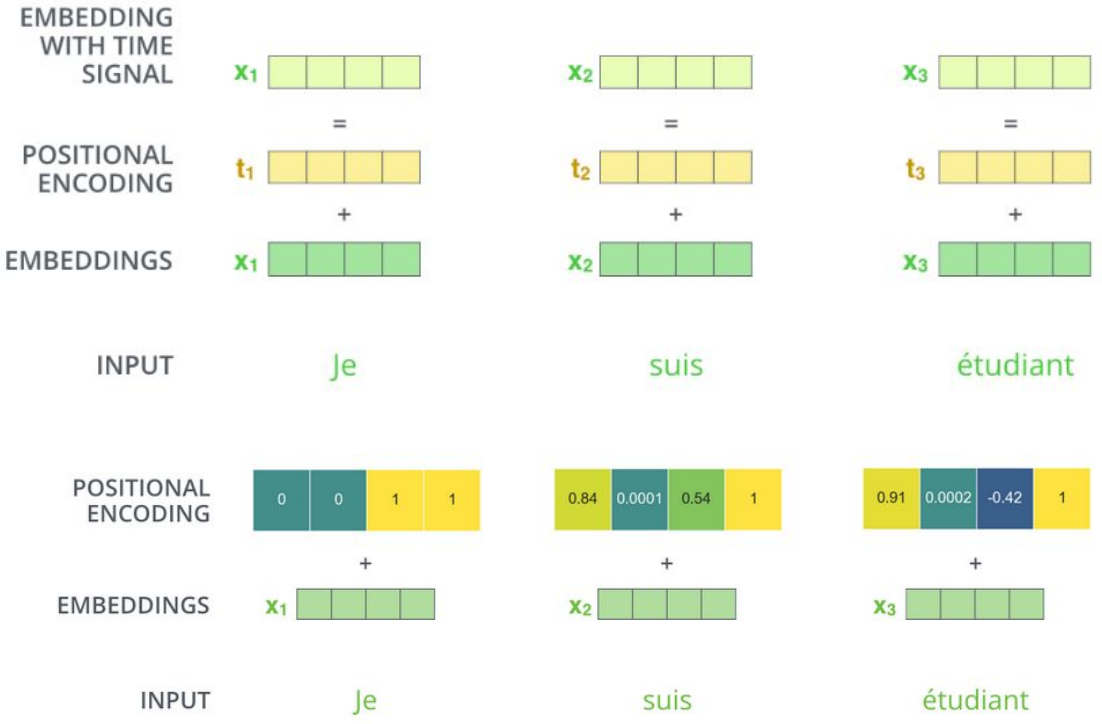
- Мы говорили о механизме внимания и о том, как оно влияет на процесс принятия решений
- Однако мы не поговорили об одной важной проблеме: внимание инвариантно к перестановкам, и потому не игнорирует порядок слов входной последовательности
- Для решения этой проблемы введем такое понятие как **positional embedding**, который кодирует позиции токенов, и объединим его с эмбеддингами токенов

Positional Encoding

Variant from the initial paper:

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





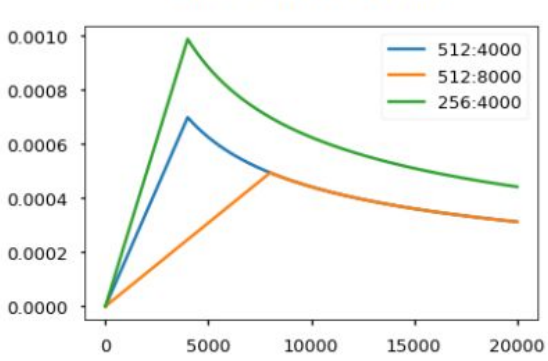
A real example of positional encoding with a toy embedding size of 4

Training details

- Cross-Entropy Loss $NLL(y_{1:M}) = -\sum_{t=1}^{M} \log p(y_t|t_{t-1})$
- Teacher Forcing для декодера
- Adam Optimizer
- lr с warm-up шедулером

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

- 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

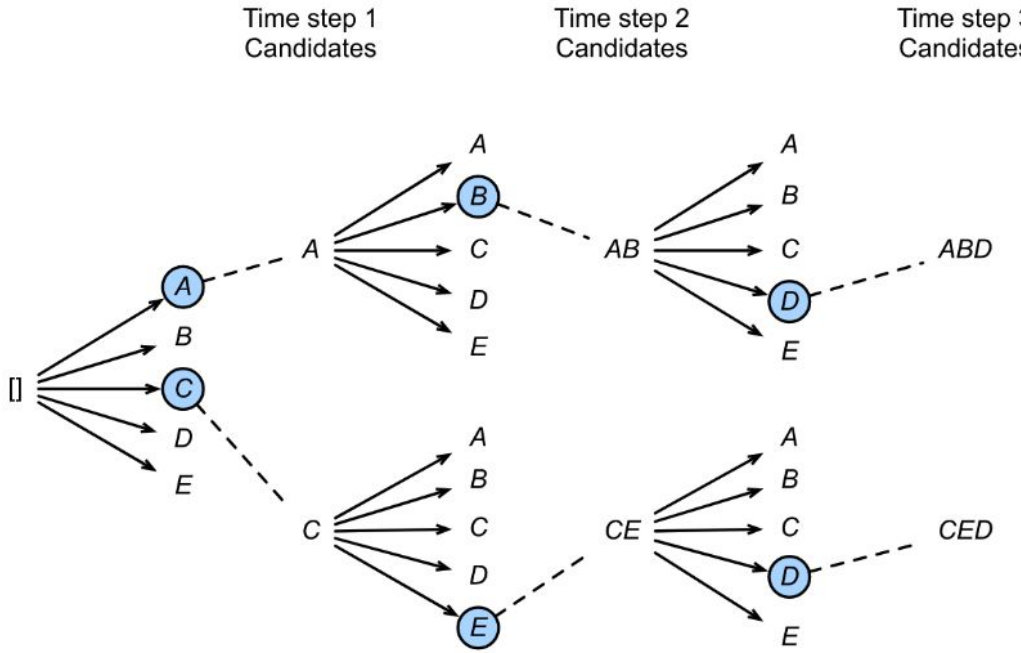
$P = 0.5 \times 0.4 \times 0.4 \times 0.6 = 0.048$

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:

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

length penalty, alpha = 0.75