Attention Is All You Need

Ashish Vaswani[*]
Google Brain
avaswani@google.com

Noam Shazeer[*]
Google Brain
noam@google.com

Niki Parmar[*]
Google Research
nikip@google.com

Jakob Uszkoreit[*]
Google Research
usz@google.com

Llion Jones[*]
Google Research
llion@google.com

Aidan N. Gomez[*] [†]
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser[*]
Google Brain
lukaszkaiser@google.com

Illia Polosukhin[*] [‡]
illia.polosukhin@gmail.com

Ирина Хомченкова, 12.12.19

Что было

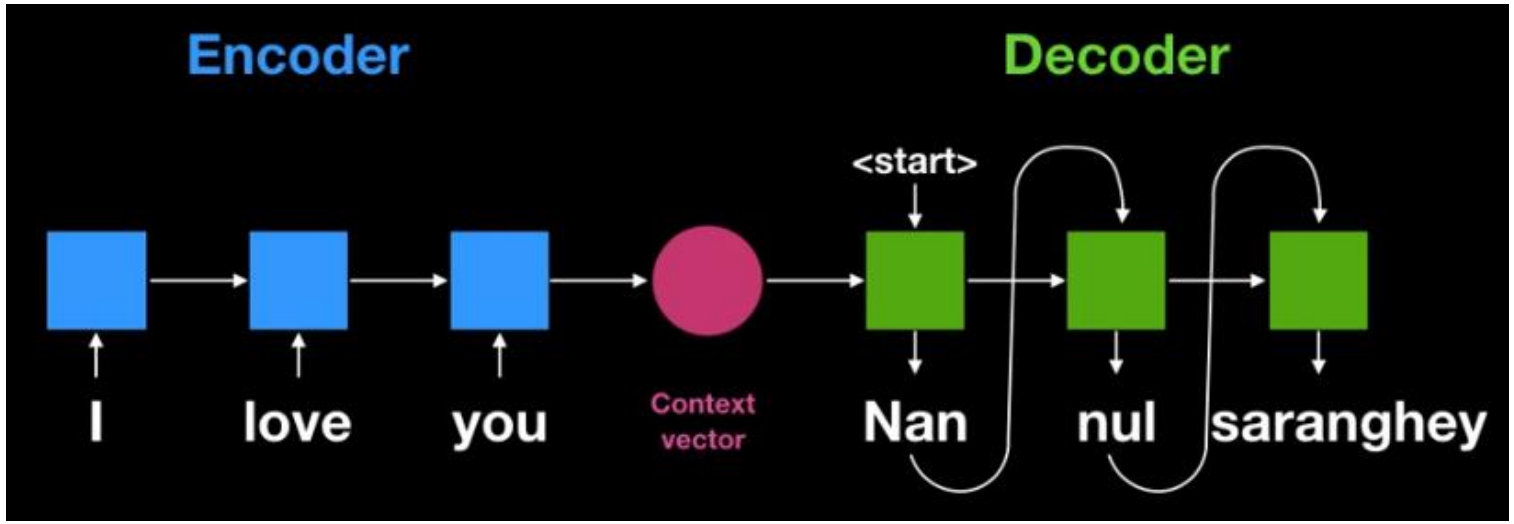
- Рекуррентные нейросети прекрасны в задачах моделирования языка и машинного перевода.
- Но связи между элементами образуют направленную **последовательность** → последовательная, а не параллельная обработка

Seq2seq

- Кодер обрабатывает входное предложение и преобразует информацию в контекстный вектор **фиксированной длины**

  (→ сложности с длинными предложениями).

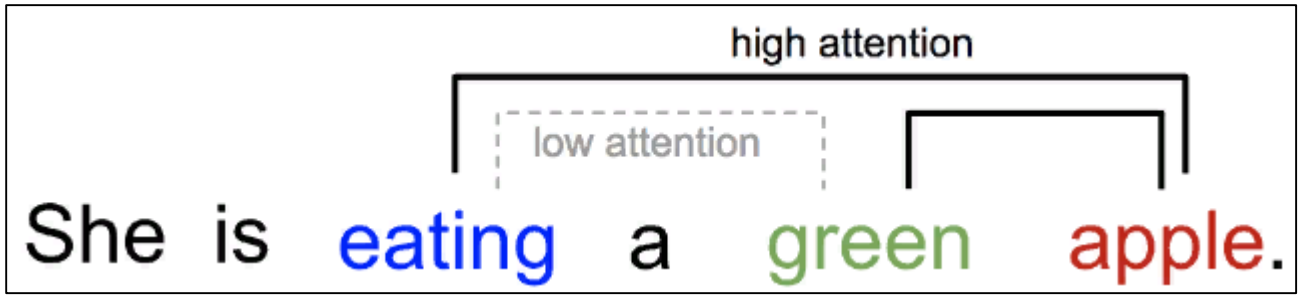
- Декодер берет контекстный вектор, чтобы выдать выходное предложение.



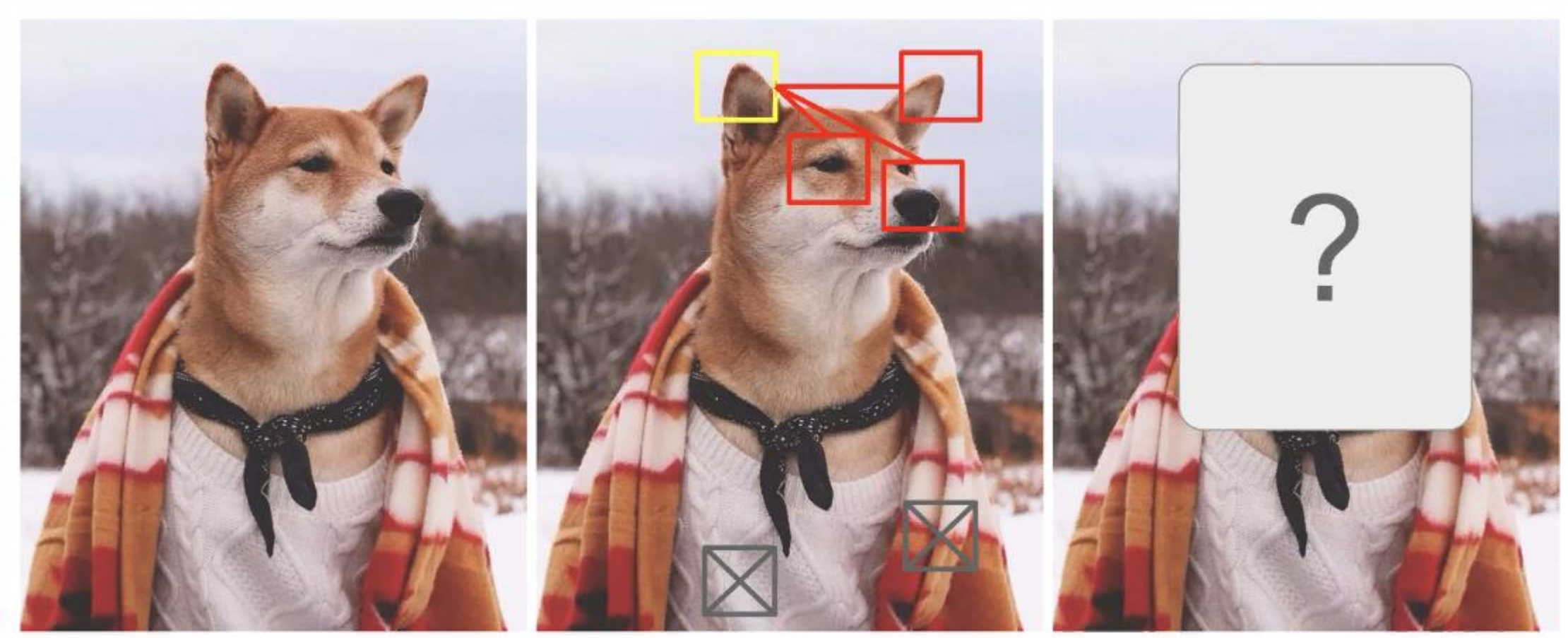
Что с этим делали

- Чтобы улучшить результаты, добавляли механизм внимания.
- Но практически везде механизм внимания использоваться вместе с рекуррентными нейросетями.

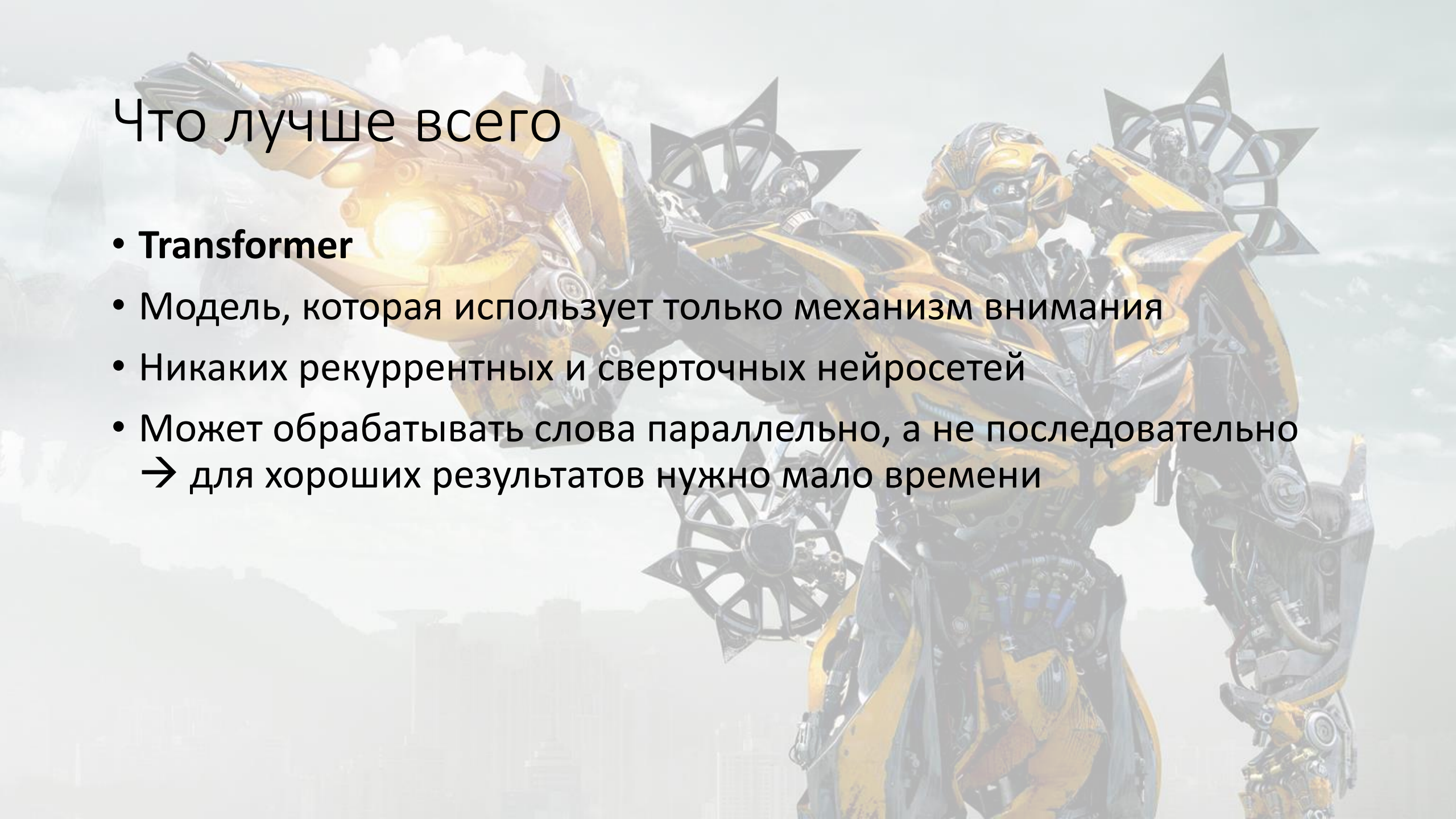
Attention



- На какие части объекта нужно обратить внимание?



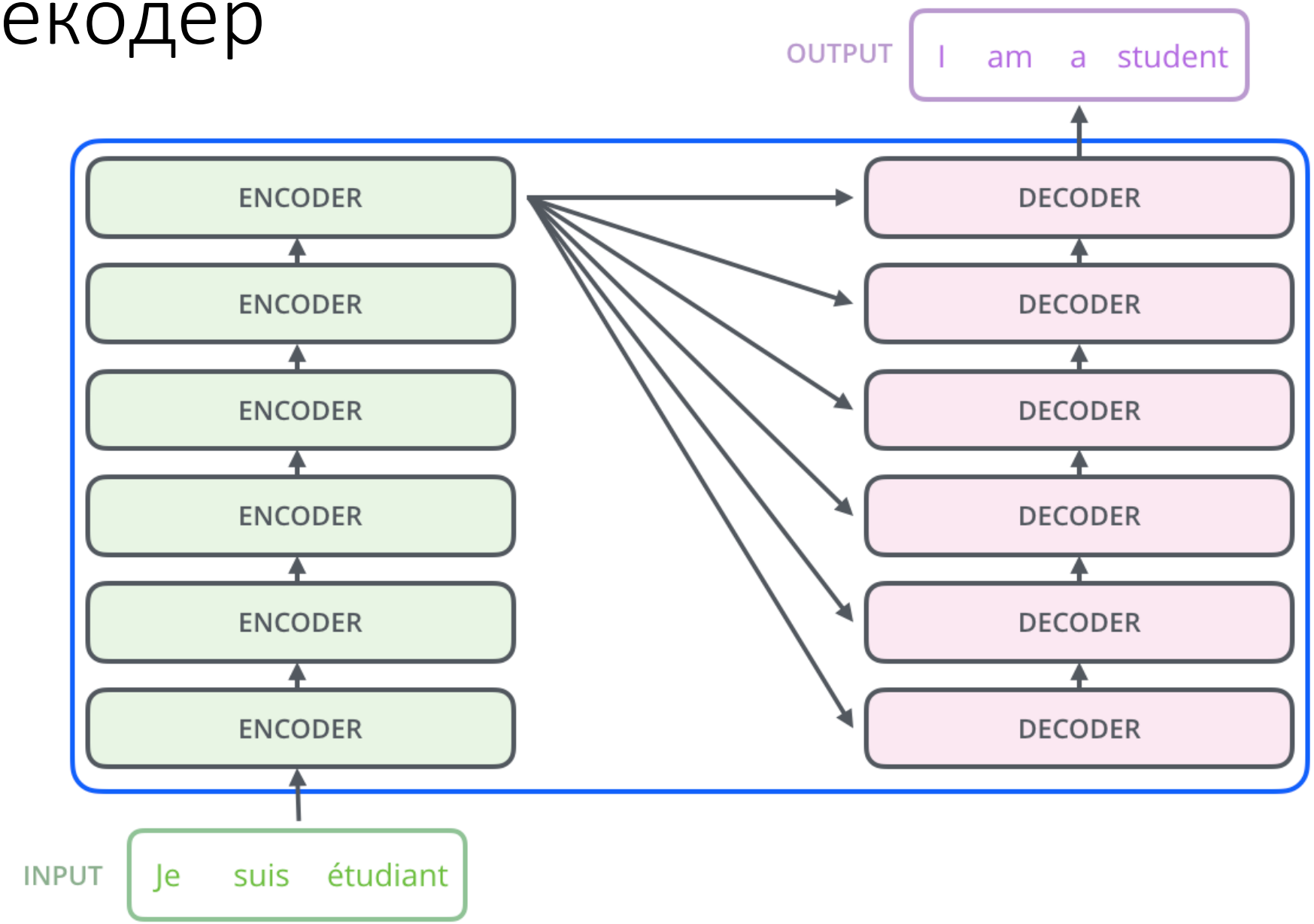
Что лучше всего



- **Transformer**
- Модель, которая использует только механизм внимания
- Никаких рекуррентных и сверточных нейросетей
- Может обрабатывать слова параллельно, а не последовательно → для хороших результатов нужно мало времени

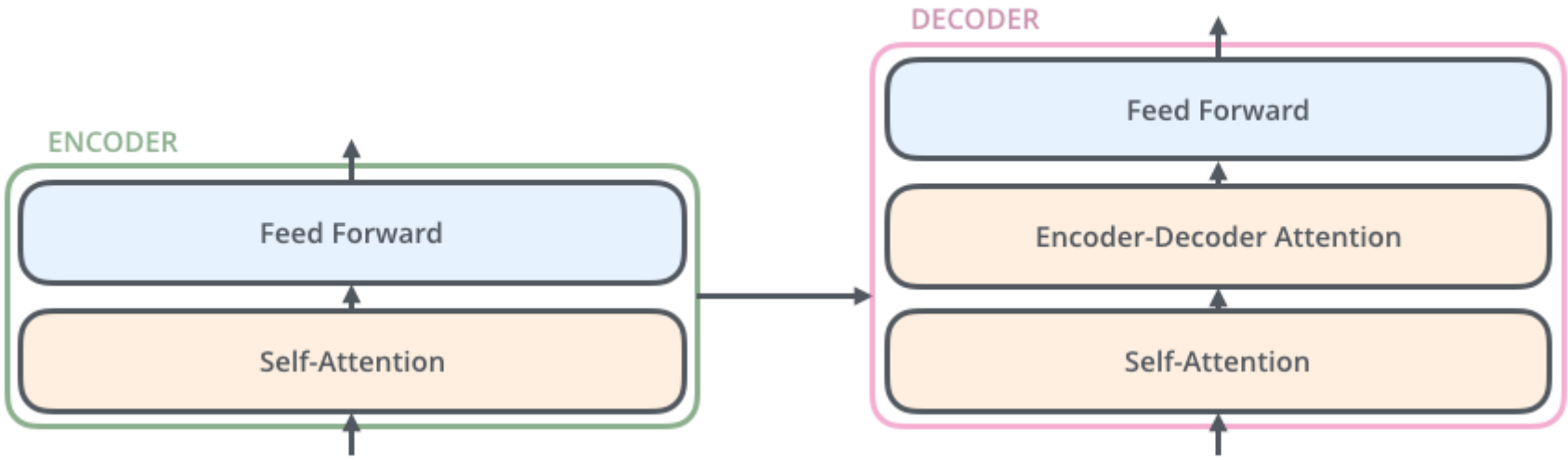
Кодер и декодер

- По 6 слоев



Кодер и декодер

- В каждом слое есть два подслоя
  - **self-attention** layer.
  - **feed-forward** neural network.
- В декодере есть еще третий слой с вниманием.



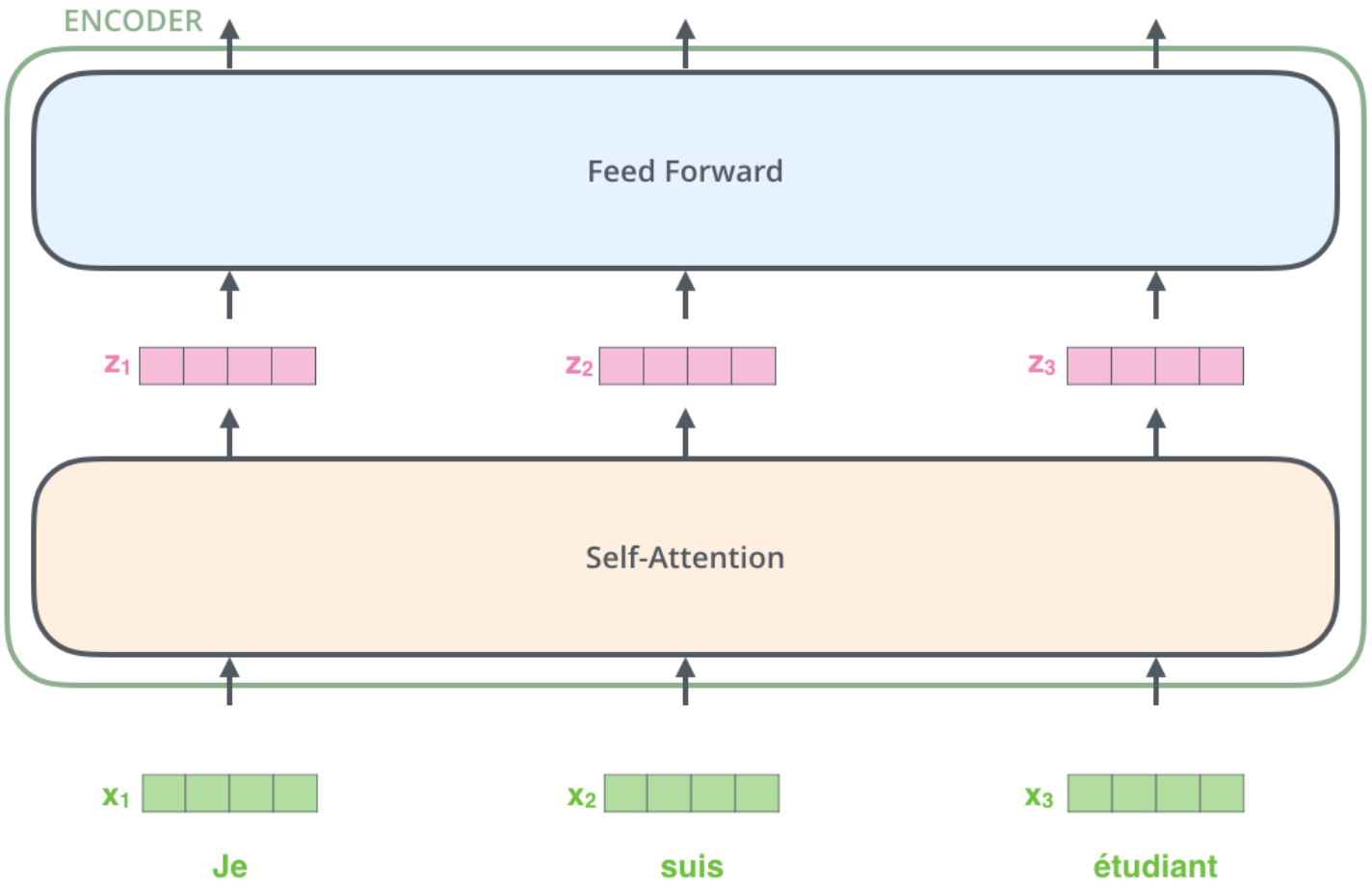
Кодер и декодер

- Используется residual/skip connection, после этого делается нормализация слоя, т.е. output каждого подслоя:

$$LayerNorm(x + Sublayer(x))$$

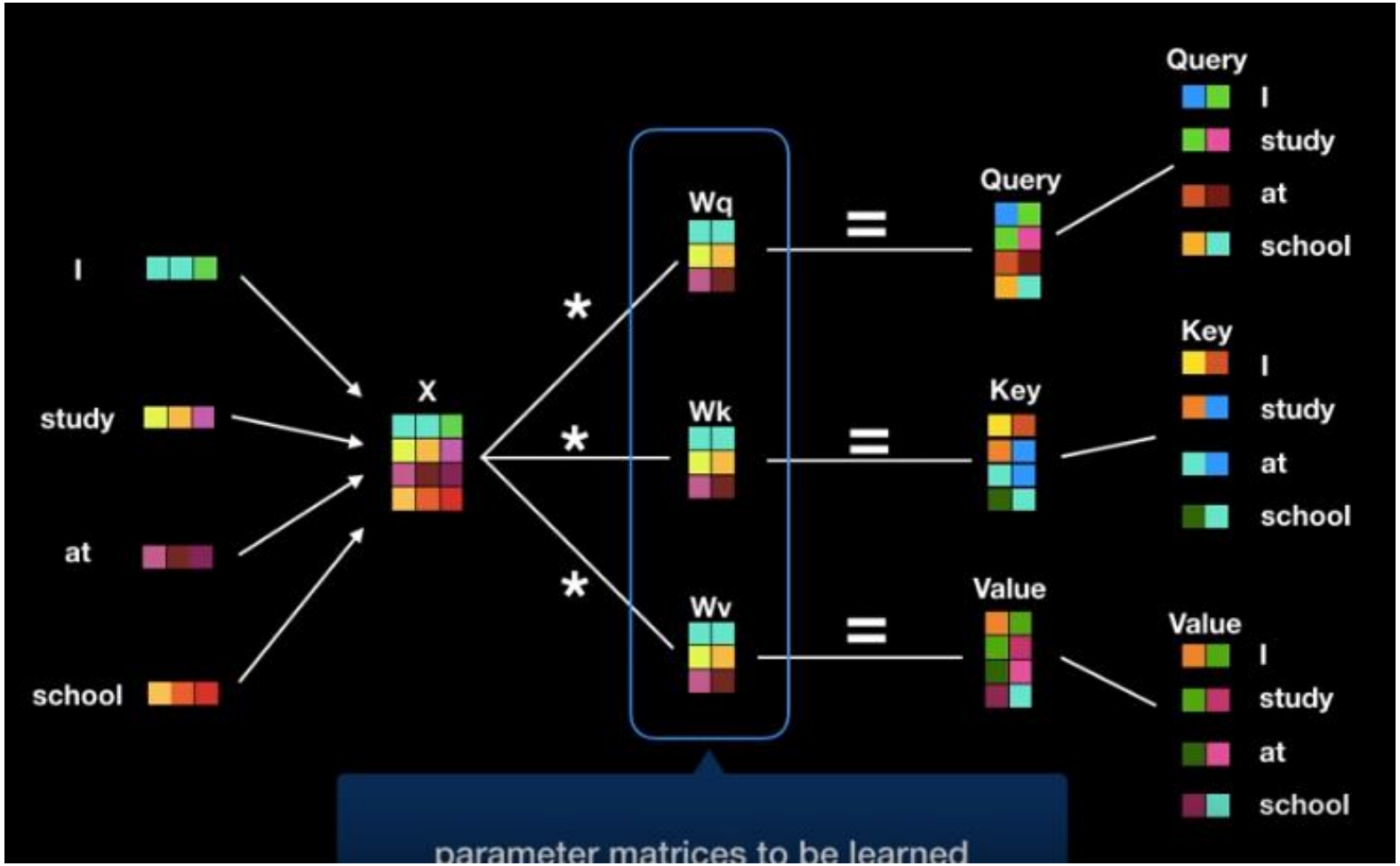
Кодер

- Каждое слово представляется в виде вектора
- Эмбеддинг происходит в самом нижнем слое кодера
- Каждое слово проходит через слои отдельно от других



Как считать self-attention

- Кладем наши эмбеддинги в матрицу X
- В процессе обучения получаем W^Q, W^K, W^V
- Умножаем X на матрицы W^Q, W^K, W^V → получаем Q, K, V

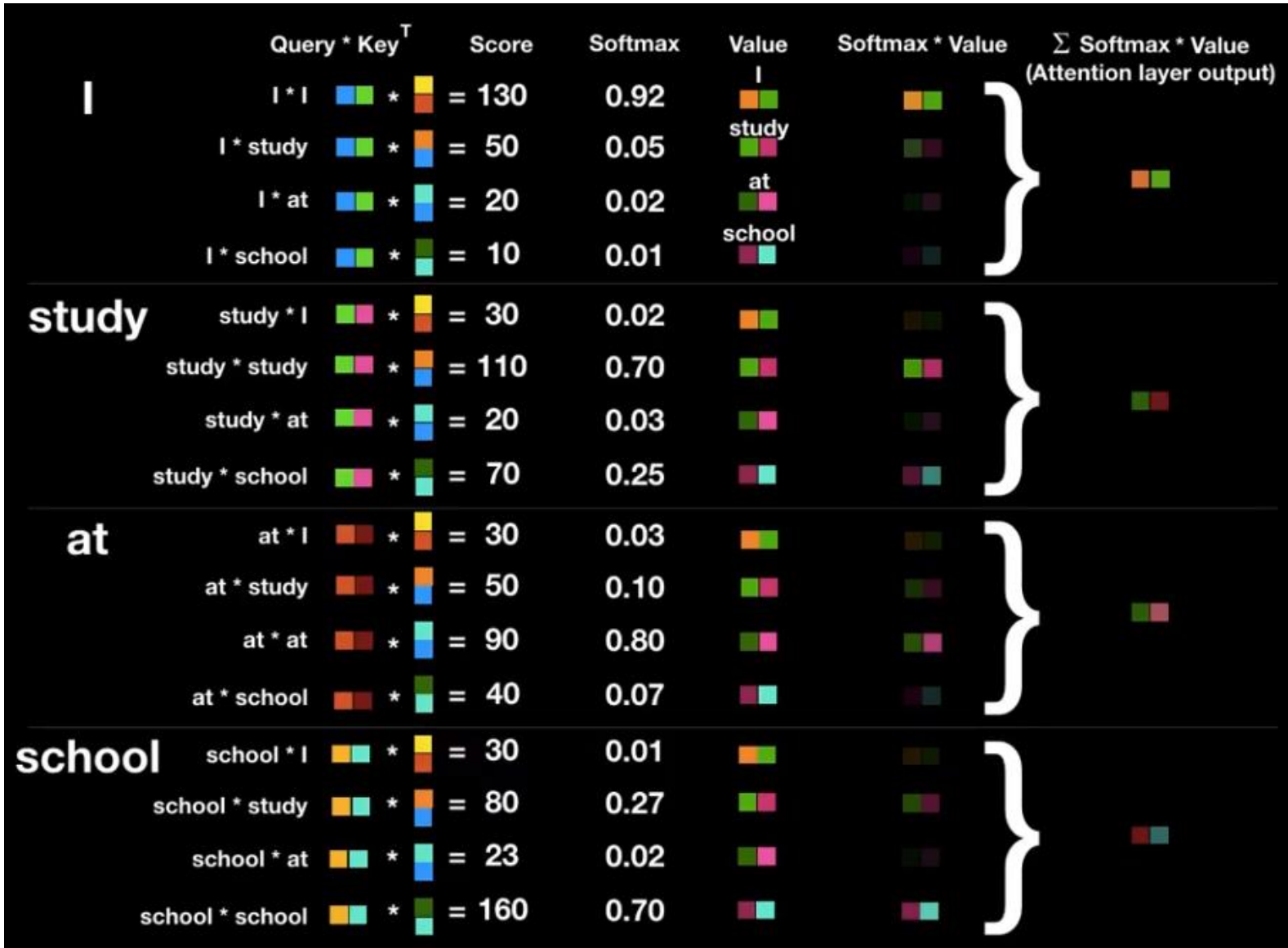


Scaled Dot-Product Attention

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

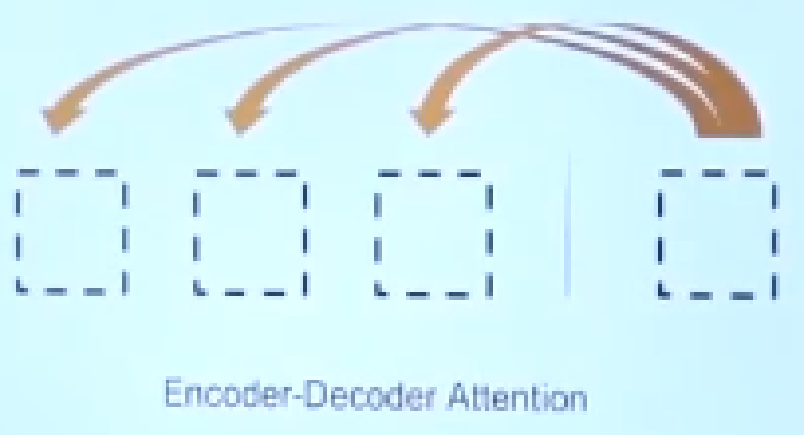
- Take the query
- Find the most similar key
- Get the values that correspond to these similar keys

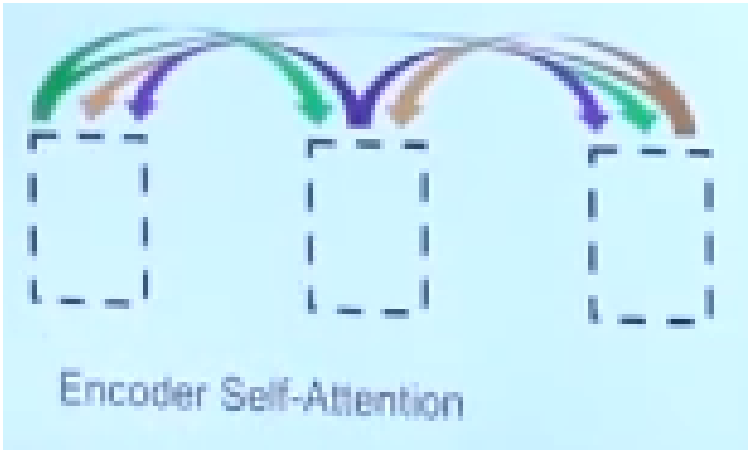
- Softmax gives the probability distribution over keys, which are peaked at the ones that are similar to a query.



Three ways of attention

- **Encoder-decoder attention**: Q из предыдущего слоя декодера, K, V из output'а кодера.
  - Every position attends over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in seq2seq models.
- **Encoder self-attention**: K,V,Q из output'а предыдущего слоя кодера. Each position in the encoder can attend to all positions in the previous layer of the encoder.
- **Decoder self-attention**: each position in the decoder attends to all positions in the decoder up to and including that position.
  - We implement this inside of scaled dot-product attention by masking out (setting to $-\infty$) all values in the input of the softmax which correspond to illegal connections.





Механизмы внимания

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Scaled-dot product

- **Dot-product attention** быстрее и more space-efficient, чем **additive attention,** потому что мы просто перемножаем матрицы.
- Для маленьких значений d_k эти два механизма работают похожим образом, но если d_k большое, то additive attention лучше.
- Подозрение: с большим d_k, the dot products могут каузировать очень маленький градиент у функции softmax, что было бы плохо для обучения → деление на корень из d_k

Feed-Forward

- Для каждого слова делаем следующее:

$$FFN(x) = W_2 \mathrm{ReLU}(W_1 X + b_1) + b_2$$

- ReLU – функция активации f(s) = max(0,s).
- Преобразования одинаковы по разным словам, но у них разные параметры в зависимости от слоя.

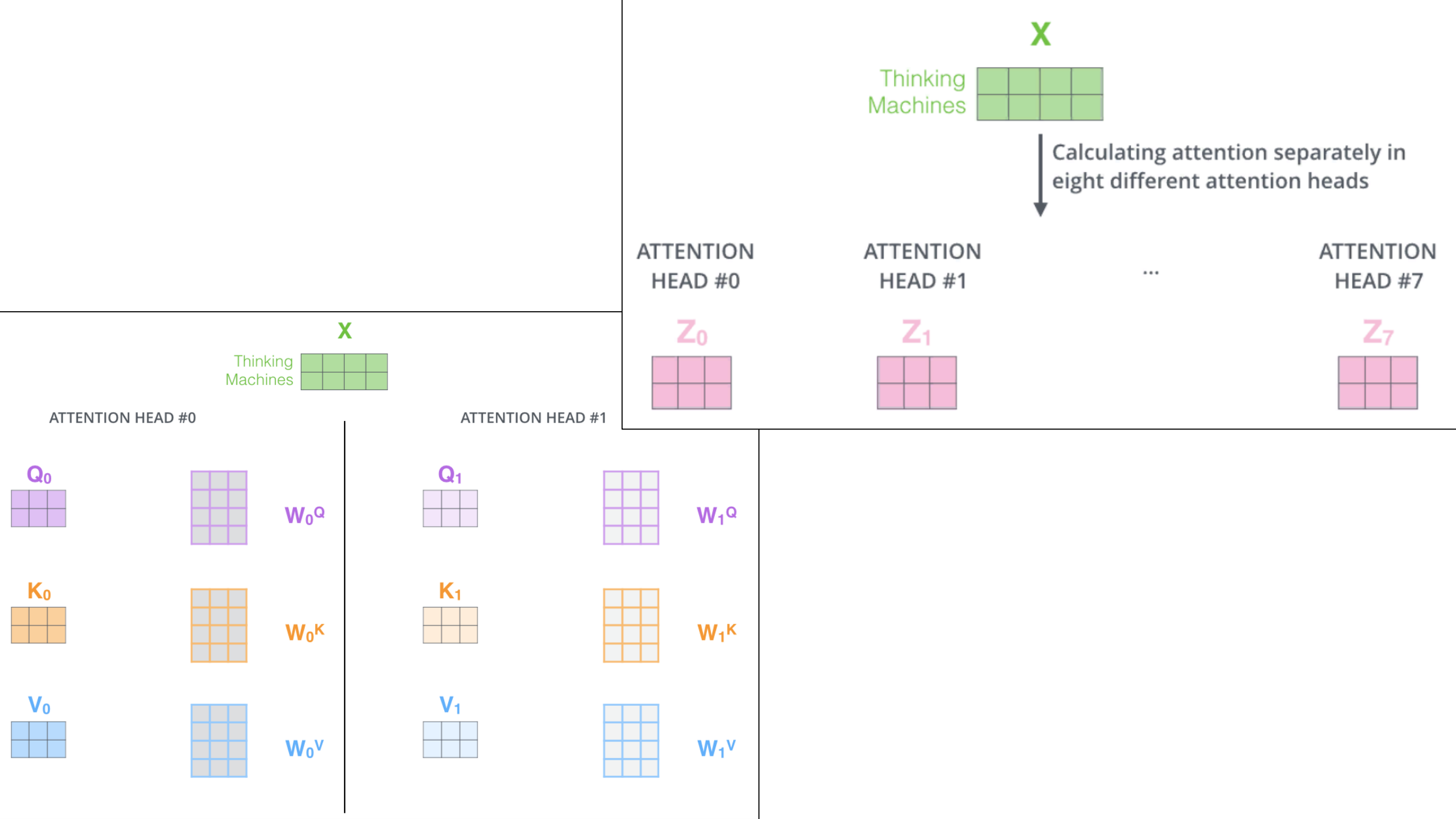
Ну и как?

- n^2*d лучше чем $n*d^2$
- D обычно около 1000, n – количество слов в предложении (70)

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

Multi-head attention

- h = 8 parallel attention layers (heads).
- Для каждого $d_k = d_v = d_{model}/h = 64$.
- Сократили размерность каждого слоя → total computational cost примерно такая же, как если бы у нас был single-head attention with full dimensionality


X
Thinking
Machines
Calculating attention separately in
eight different attention heads
ATTENTION
HEAD #0
ATTENTION
HEAD #1
...
ATTENTION
HEAD #7
Z0
Z1
Z7
X
Thinking
Machines
ATTENTION HEAD #0
ATTENTION HEAD #1
Q0
K0
V0
W0Q
W0K
W0V
Q1
K1
V1
W1Q
W1K
W1V

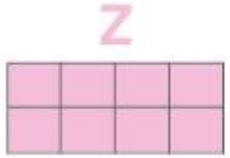
1) Concatenate all the attention heads

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

=



$$MultiHead(Q,K,V) = Concat(head_1, \dots head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

1) This is our input sentence*

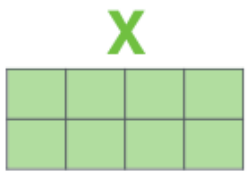
2) We embed each word*

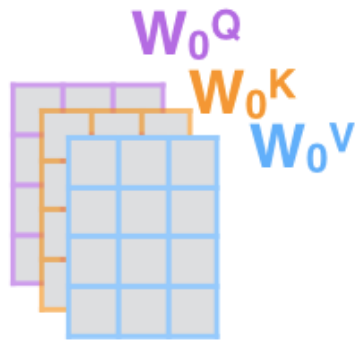
3) Split into 8 heads. We multiply X or R with weight matrices

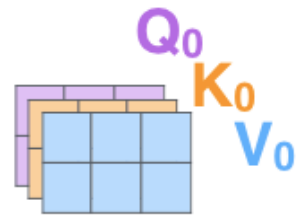
4) Calculate attention using the resulting Q/K/V matrices

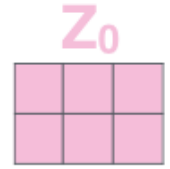
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

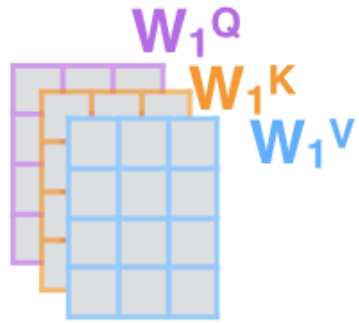
Thinking Machines

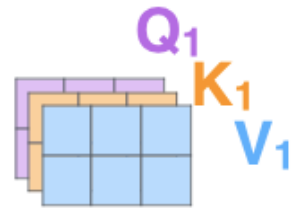


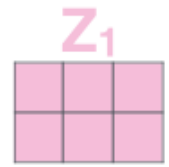


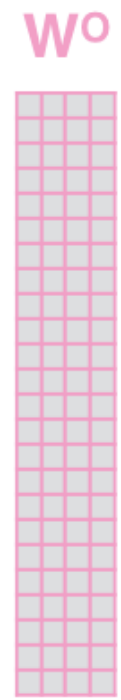


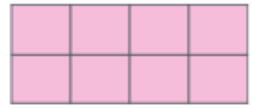




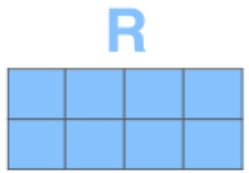


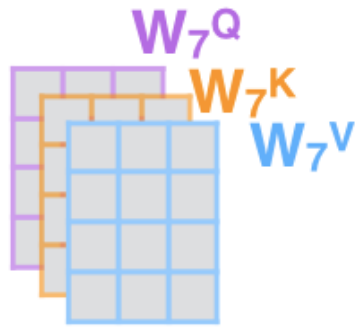


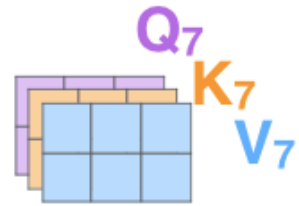


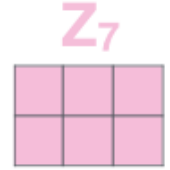


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

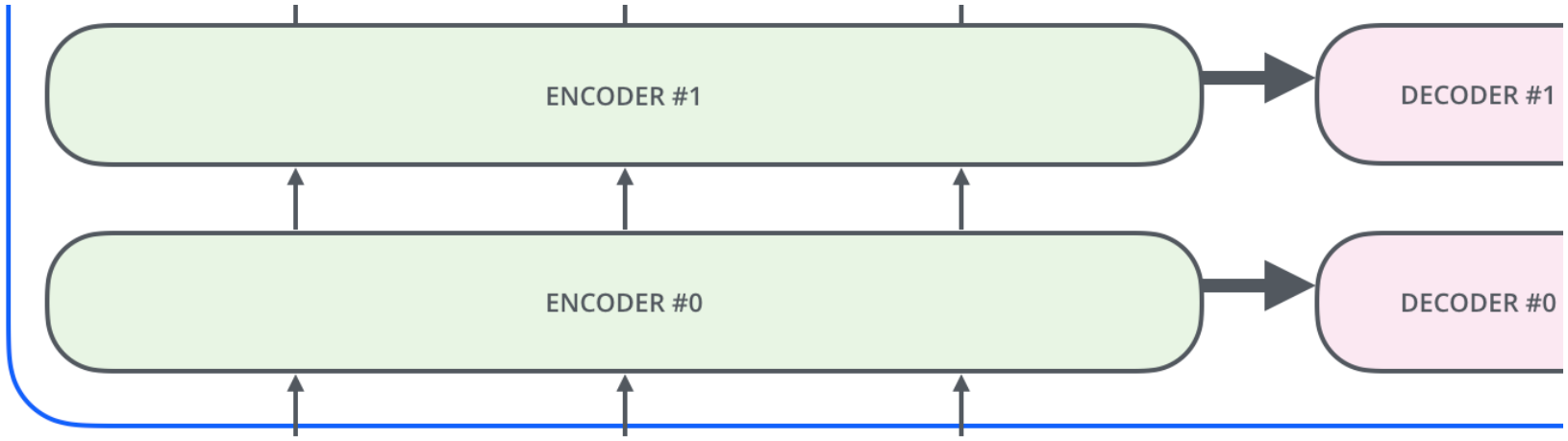


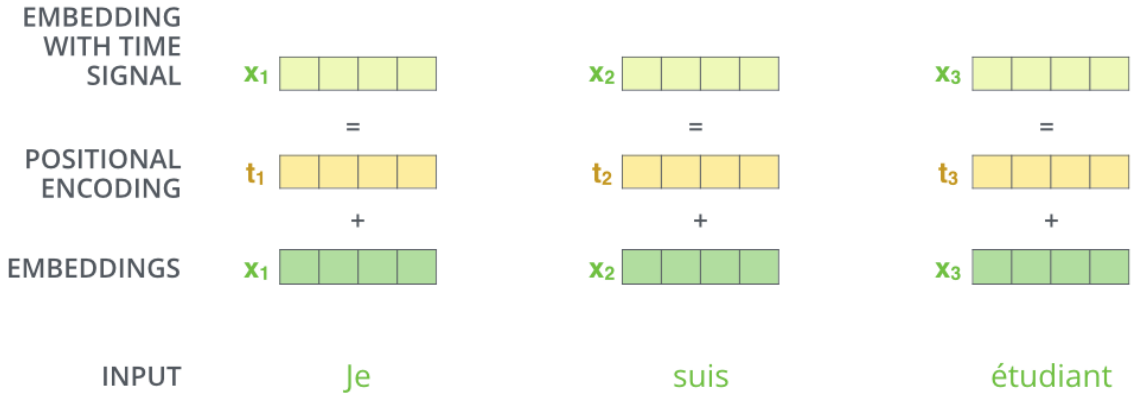






Positional encoding



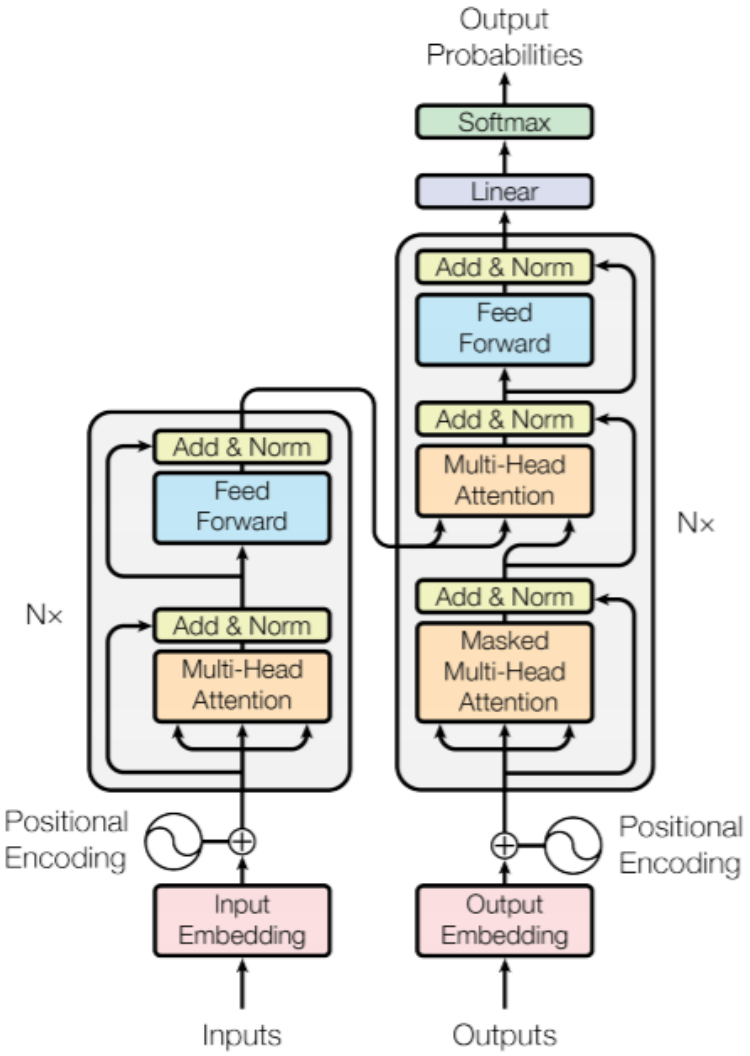


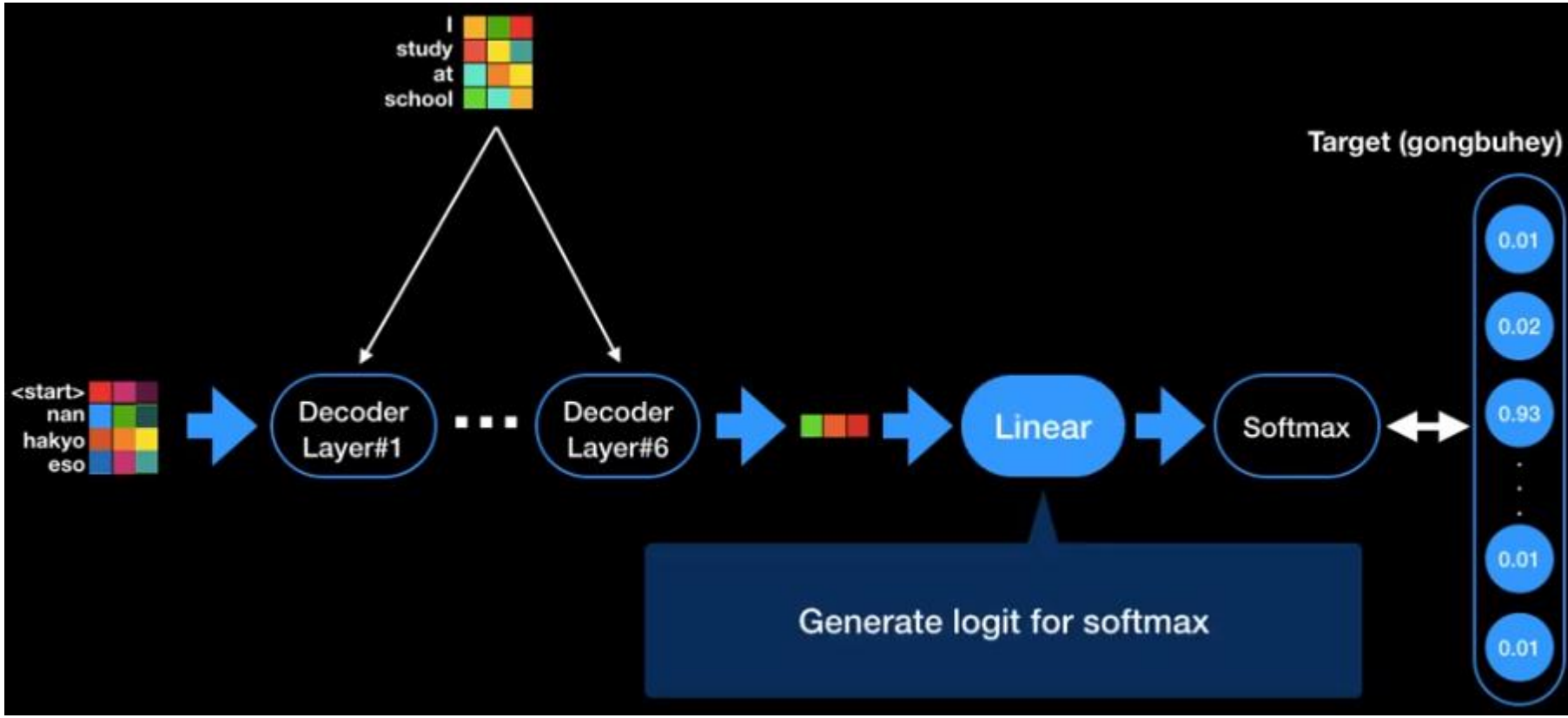
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Pos – position,

i – dimension

Декодер

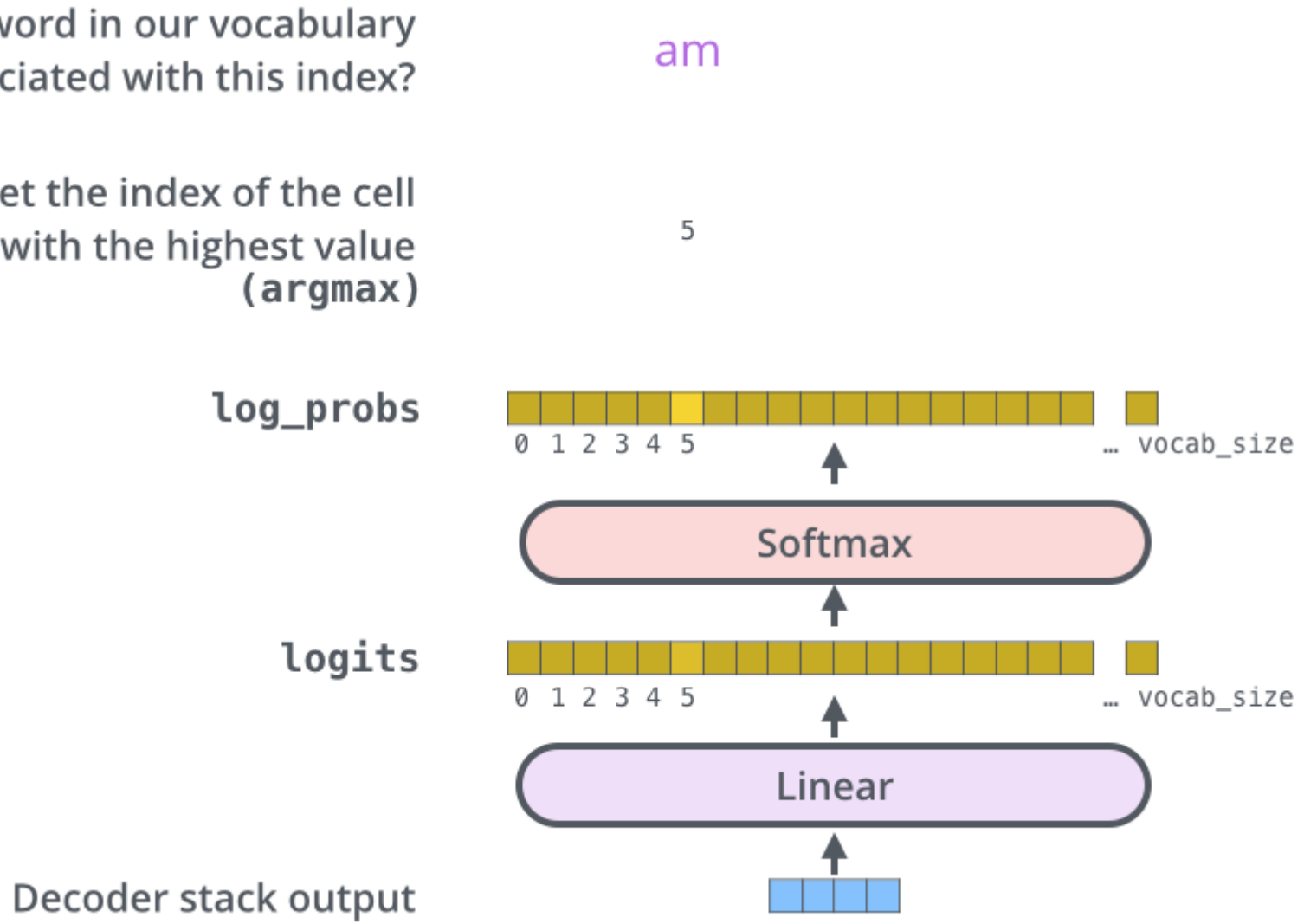




- At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

Декодер

- У нас есть какие-то матрицы
- Как получить слово?
- Linear: нейросеть, которая из нашего вектора делает вектор еще больше (logits vector)
- Допустим наша модель знает 10000 слов (output vocabulary), которые она выучила по обучающей выборке → в векторе 10000 клеточек.
- softmax потом превращает это в вероятности.



Machine translation results

Base: 12 часов
Big: 3,5 дней

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Результат

- Transformer, model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention.
- For translation tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.
- On both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks, we achieve a new state of the art.

Coreference resolution (Winograd Schema)