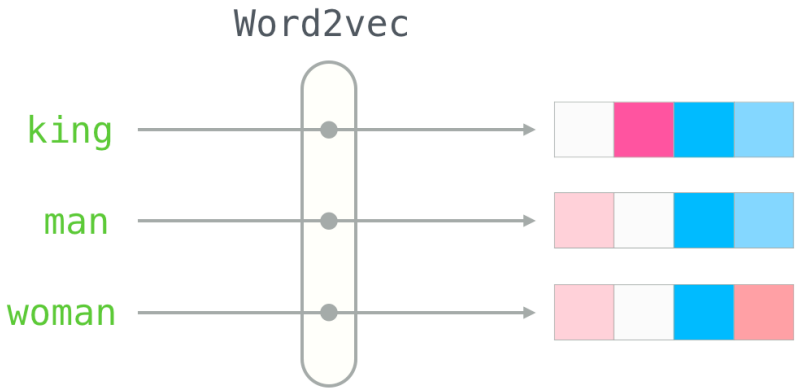
Трансформеры

Курс «Основы анализа текстовых даных»

Кафедра управления интеллектуальных технологий

НИУ «МЭИ»

Векторизация

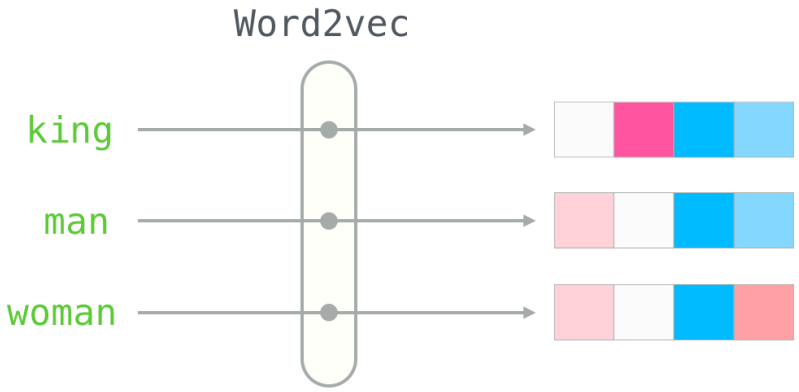


Нужен препроцессинг:

- Стемминг \ Лемматизация
- Удаление цифр, знаков пунктуации
- Проверка на опечатки\ошибки
- Удаление стоп-слов - ???

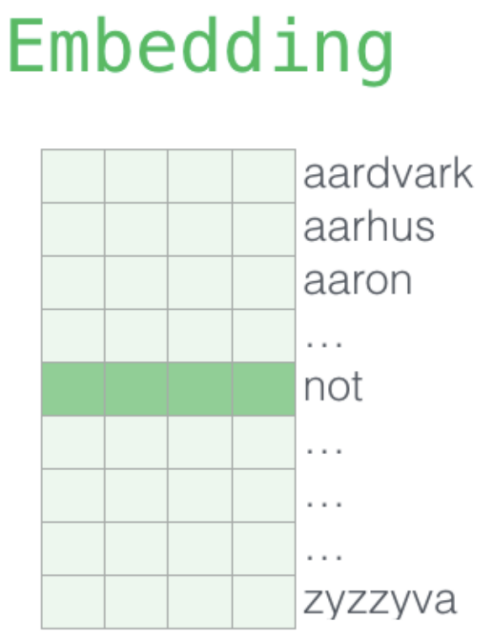
По мотивам https://www.youtube.com/watch?v=iOrNbK2T92M&t=105s

Векторизация



Нужен препроцессинг:

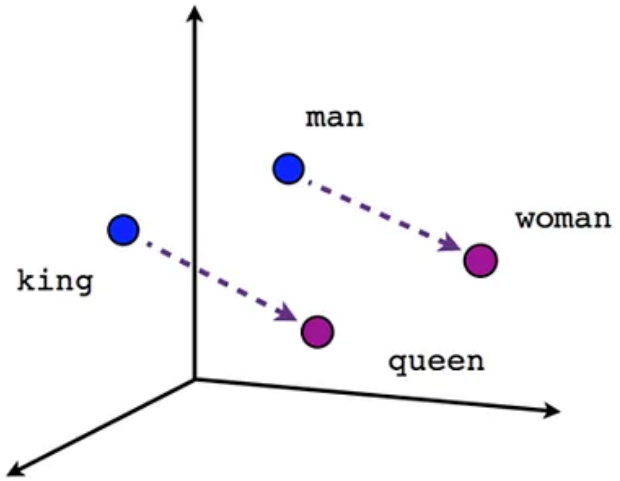
- Стемминг \ Лемматизация
- Удаление цифр, знаков пунктуации
- Проверка на опечатки\ошибки
- Удаление стоп-слов - ???



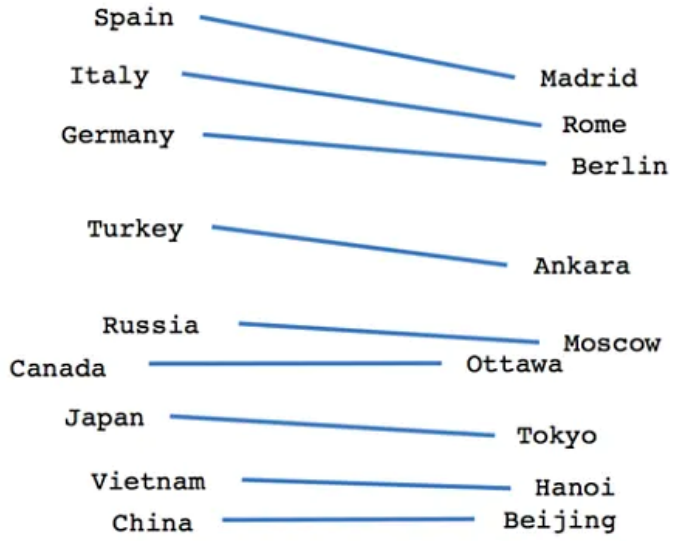
Высота – размер словаря
Ширина – размерность эмбедингов

Если два языка – высота увеличивается вдвое

Можем выделить направление для вектора слова



Male-Female



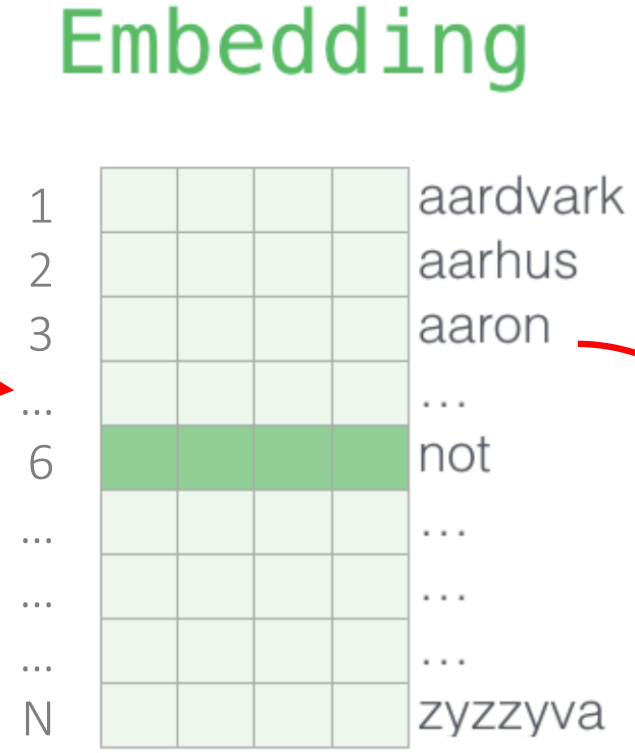
Country-Capital

Векторизация

original text "hello world!"

tokens ['hello', 'world', '!']

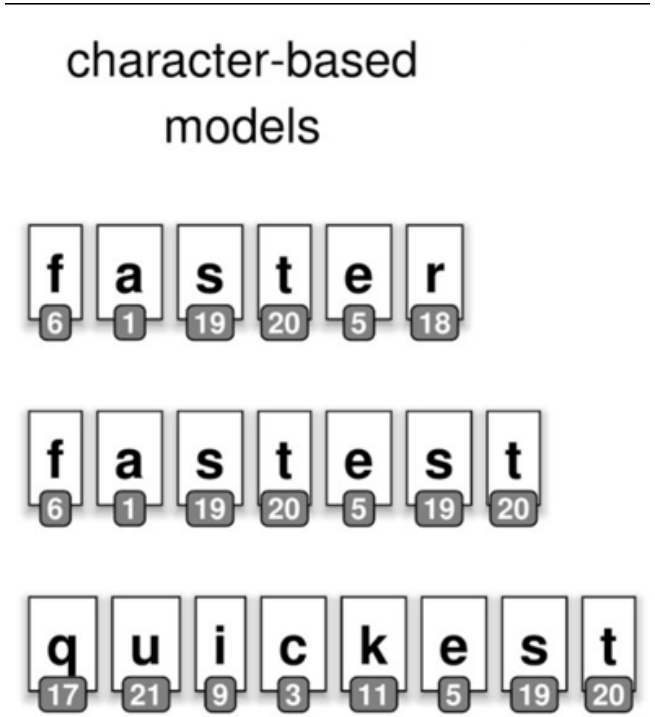
token IDs [7592, 2088, 999]



Предложение – усредненный вектор из слов

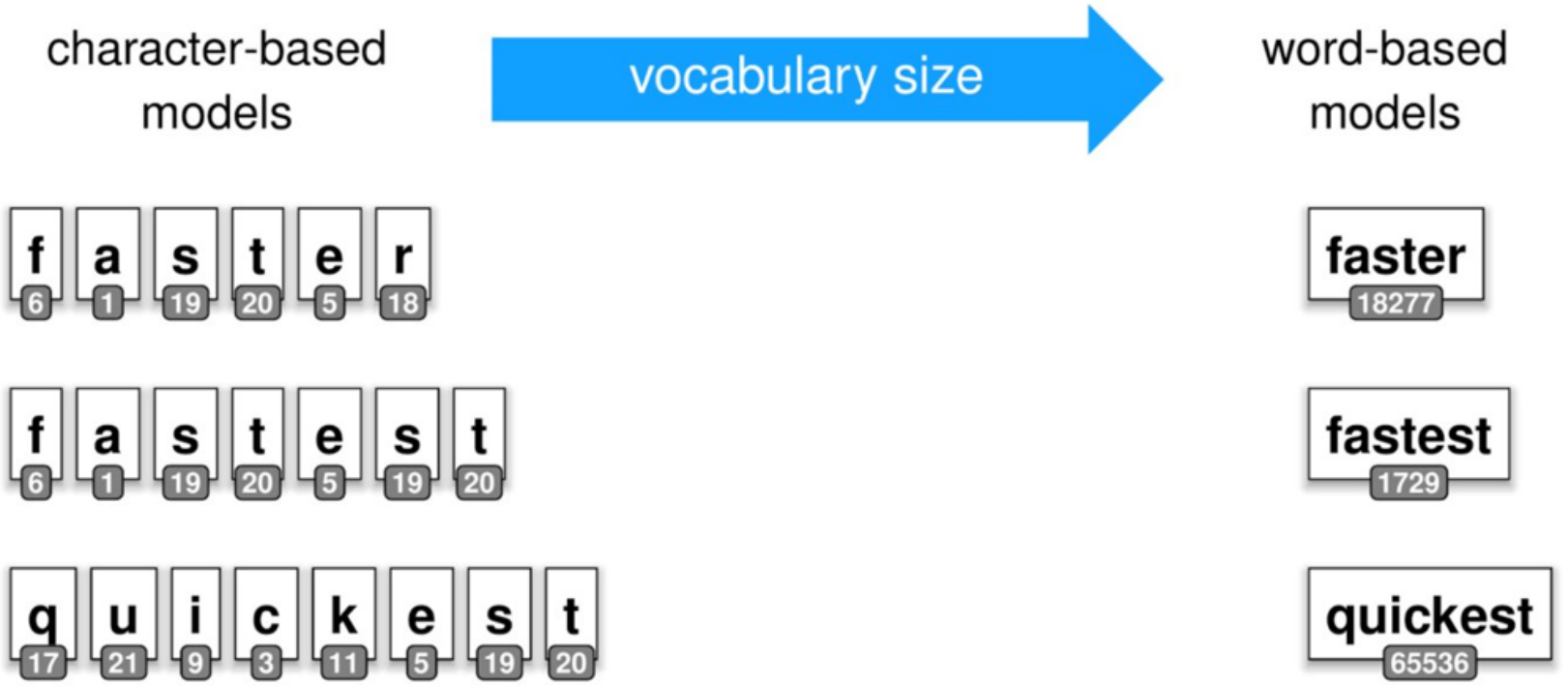
Модель Word2Vec <=> матрица эмбеддингов

Как можем представить текст?



Слова состоят из букв. Представляем слова как набор букв.
Размер словаря небольшой (~100 для одного языка).
Но размер текста очень большой

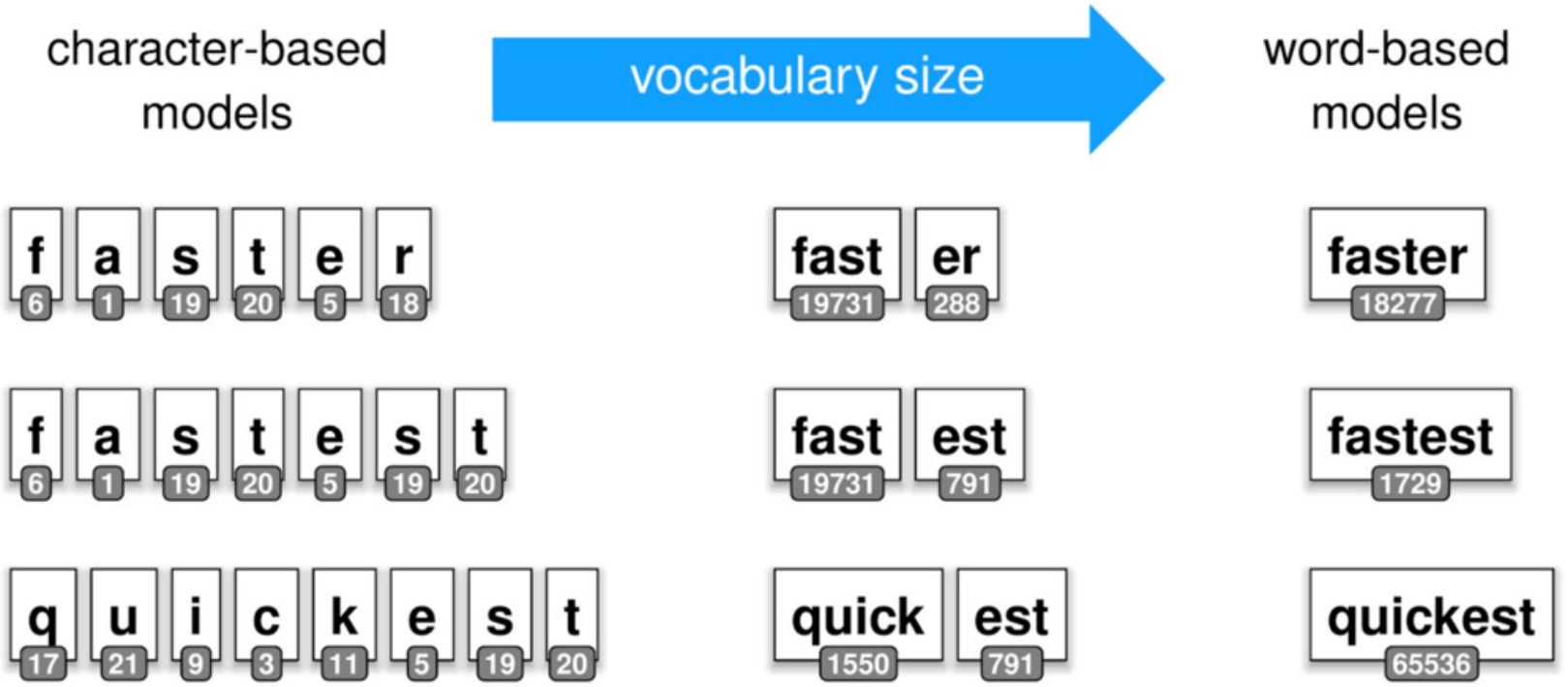
Как можем представить текст?



Слова состоят из букв. Представляем слова как набор букв.
Размер словаря небольшой (~100 для одного языка).
Но размер текста очень большой

Длина последовательности меньше, но на порядки возрастает длина словаря

Как можем представить текст?

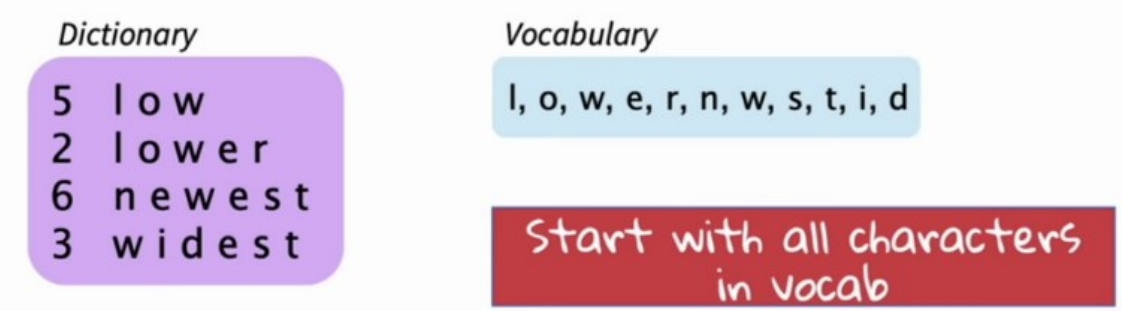


Слова состоят из букв. Представляем слова как набор букв.
Размер словаря небольшой (~100 для одного языка).
Но размер текста очень большой

Золотая середина – самые частовстречаемые буквосочетания.
Смысл и в корне, и в суффиксе

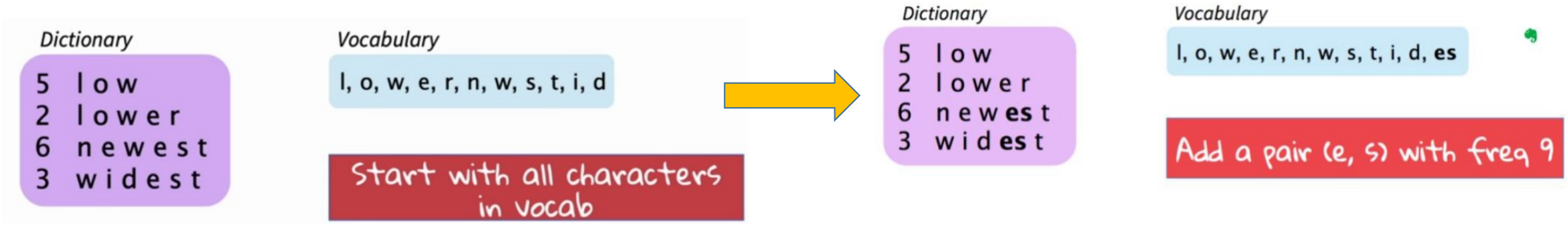
Длина последовательности меньше, но на порядки возрастает длина словаря

Как получить набор таких сочетаний (токенов)?



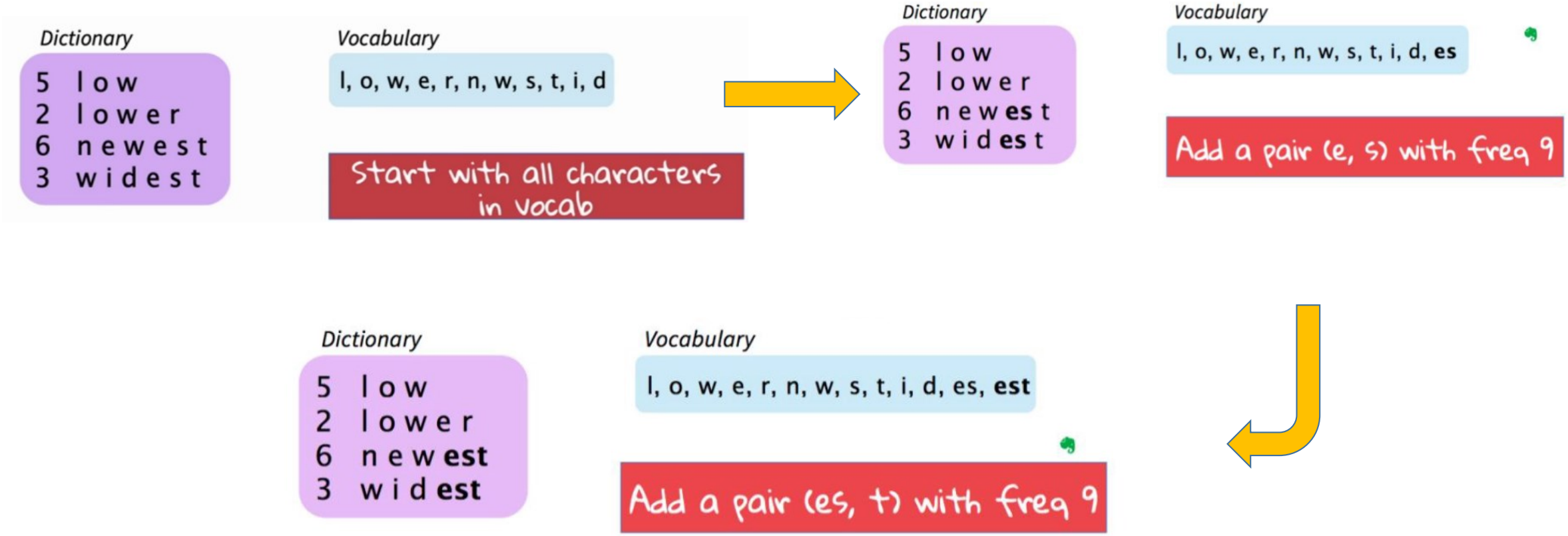
Строим словарь токенизатора:
Инициализируем словарь набором уникальных символов

Как получить набор таких сочетаний (токенов)?



Строим словарь токенизатора:
Инициализируем словарь набором уникальных символов
Добавляем наиболее частые пары символов

Как получить набор таких сочетаний (токенов)?



Строим словарь токенизатора:
Инициализируем словарь набором уникальных символов
Добавляем наиболее частые пары символов.
Добавляем наиболее частые тройки символов….
Повторяем, пока не достигнем заданного размера словаря.

Как получить набор таких сочетаний (токенов)?

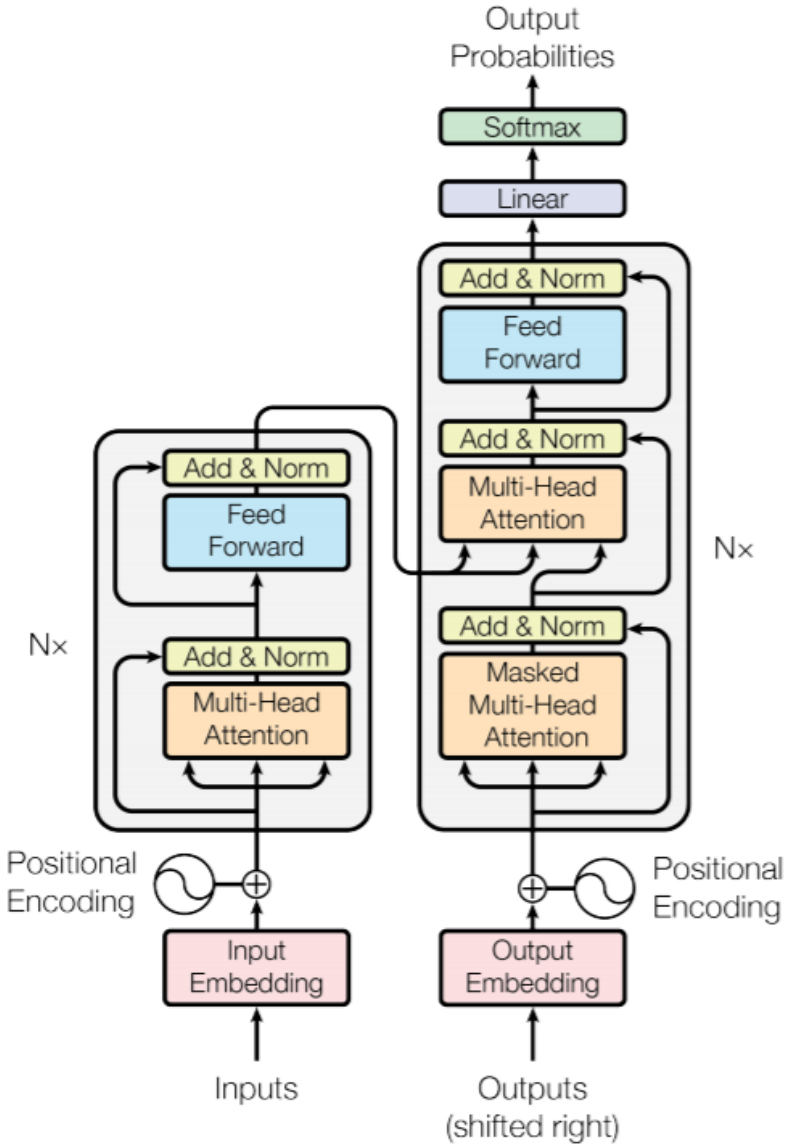
Слишком большой словарь - много вычислений, попадут токены (слова), которые встречаются всего несколько раз на огромной выборке (в т.ч. – слова с опечатками).
На таких словах не обучится эмбеддинг – он будет «случайным» - зашумляется модель

Часто используют токенизаторы:

- Byte Pair Encoding (BPE) – оперируем байтами => можем представить символ любого алфавита
  - GPT-like models (OpenAI, …)
- WordPiece + SentencePiece
  - BERT-like models (Google, …)

```
['ноутбук', 'acer', 'aspire', '3', '(', 'a317', '-', '32', '-', 'p3', '##dh', ')', '(', 'intel', 'pentium', 'n5000',
'1100mhz', '/', '17', '.', '3', '"', '/', '1600x900', '/', '4gb', '/', '256gb', 'ssd', '/', 'dvd', 'нет', '/', 'inte
l', 'uhd', 'graphics', '605', '/', 'wi', '-', 'fi', '/', 'bluetooth', '/', 'endless', 'os', ')', 'nx', '.', 'hf', '##
2er', '.', '005', 'черныи']
```

Трансформер



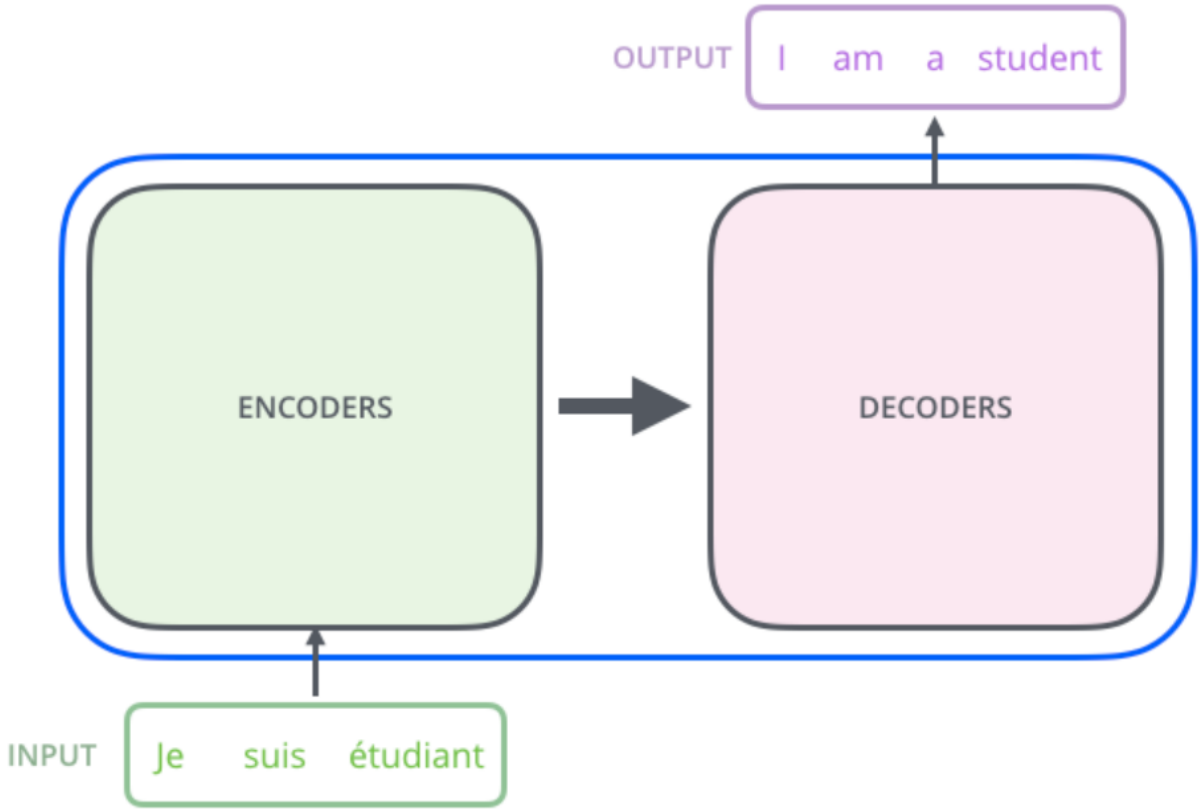
Энкодер Декодер

Attention is all you need

Paper: arxiv.org/abs/1706.03762

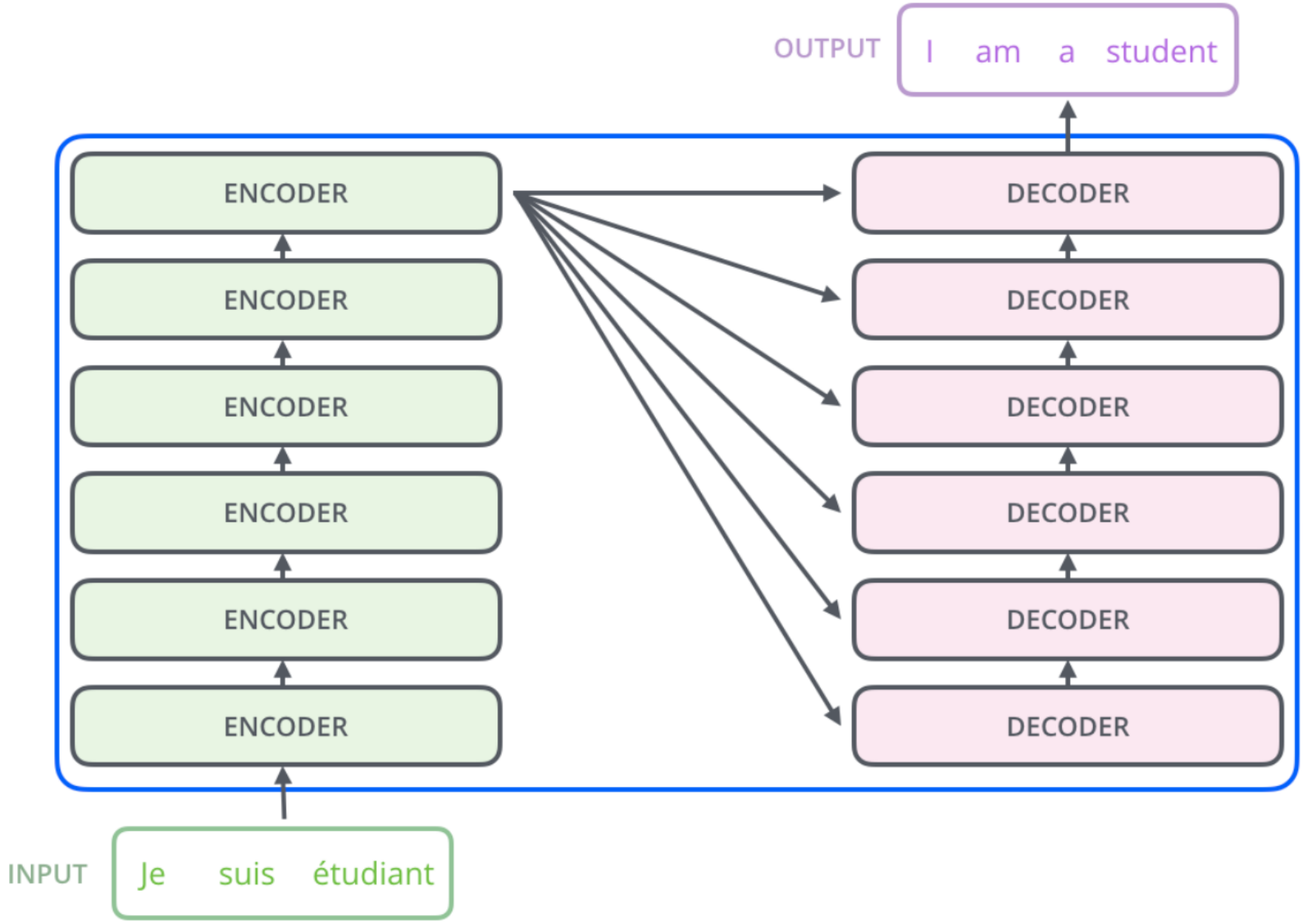
Процитировано 116 тыс. раз
(на апрель 2024)

Трансформер

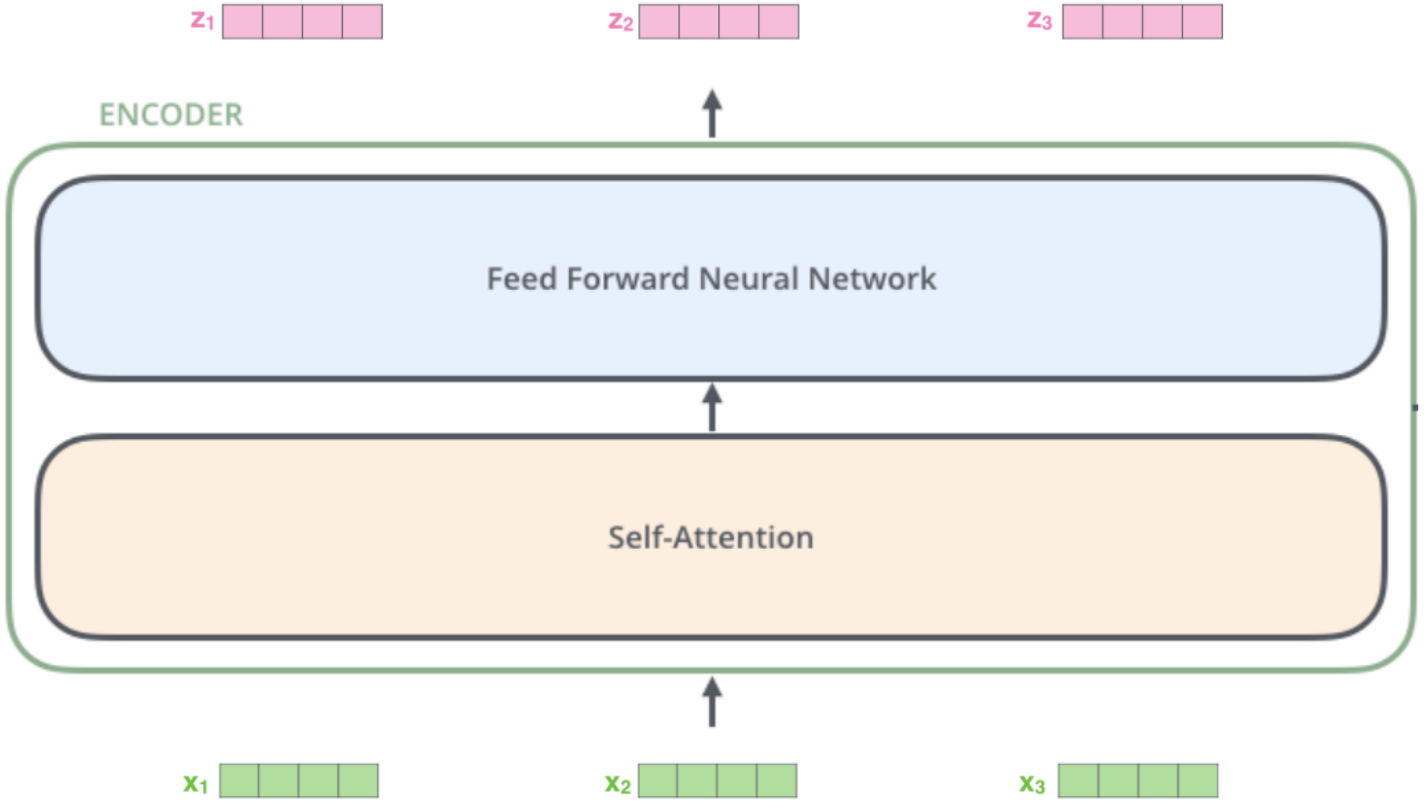


Текст на одном языке подается на энкодер, который вычленяет «смысл». Затем этот «смысл» передается в декодер, который переводит его на второй язык.
Не нужно обучать модель на каждой паре языков. Собираем из обученных энкодеров-декодеров нужную пару.

Энкодер-декодер



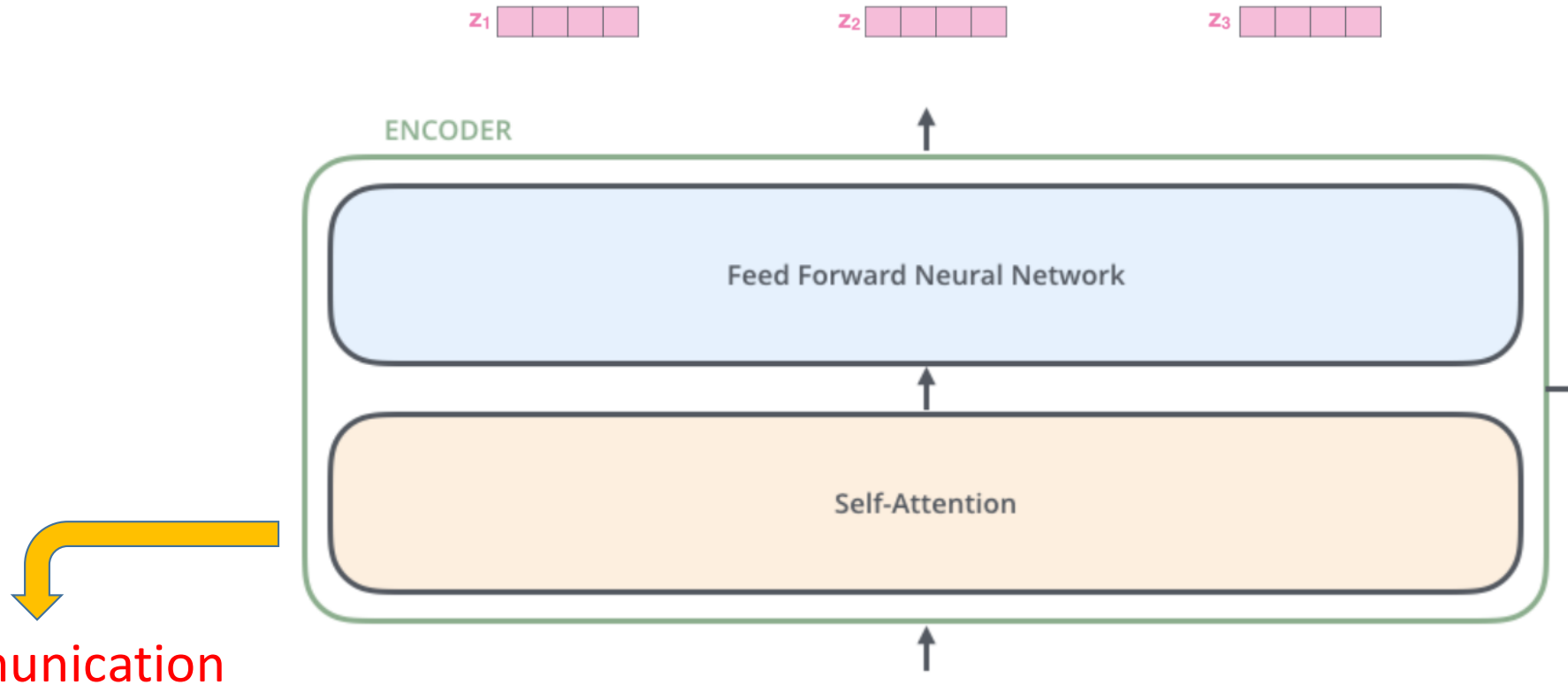
Энкодер



Каждый блок энкодера имеет одинаковые интерфейсы:

- N эмбеддингов размера D – на Входе
- N эмбеддингов размера D – на Выходе

Энкодер



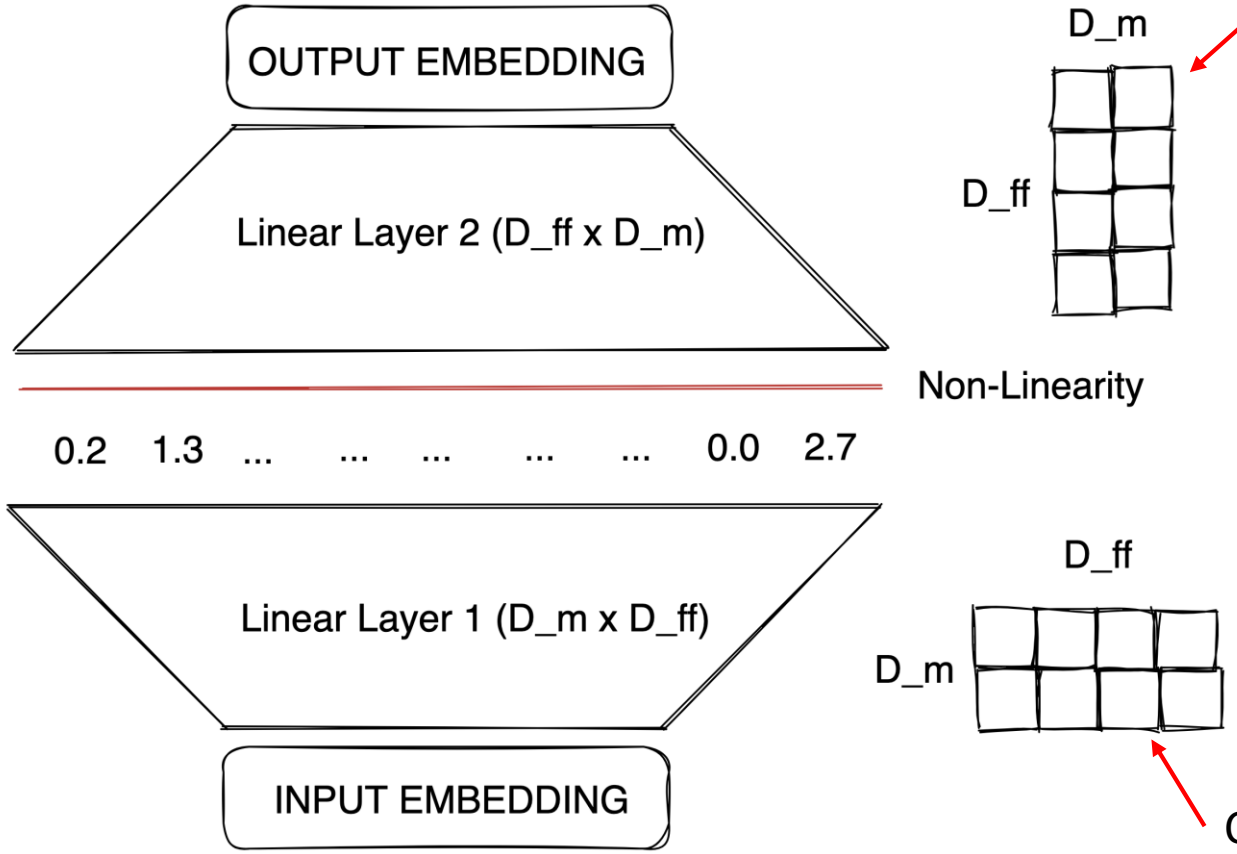
Computation

Обрабатываем токены независимо параллельно, дополнительно выделяя более сложные зависимости

Communication

Обрабатываем все входные токены за один раз. При обработке токенов смотрим на контекст и выделяем зависимости

Feed Forward слой



Non-Linearity

Хранит знания

Feed Forward состоит из двух линейных слоев: один увеличивает размерность эмбеддинга, второй сжимает его обратно:

D_m → D_ff → D_m, where D_ff >> D_m

Обычно (для BERT-Base):

D_m=768

D_ff=3072 (x4)

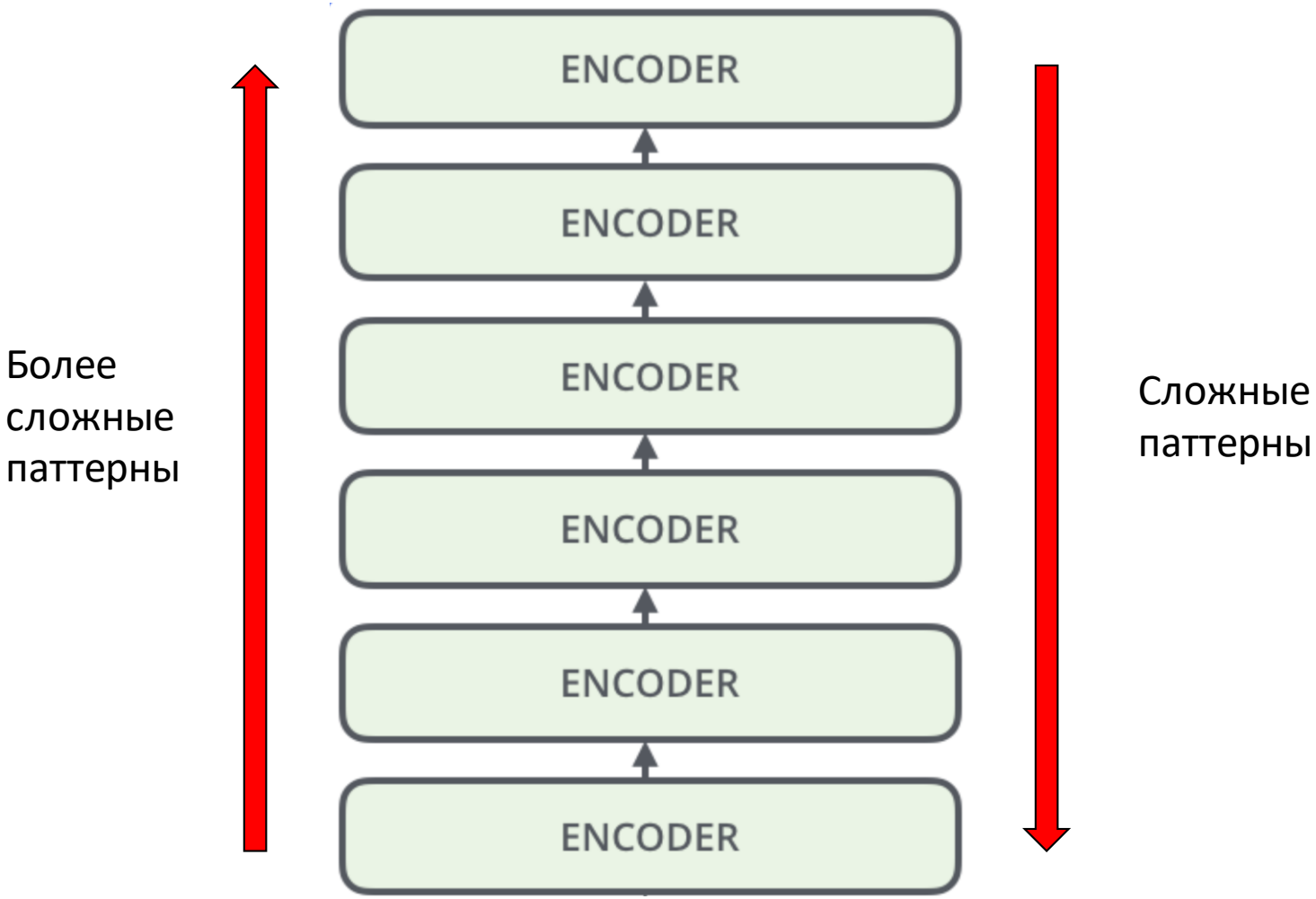
Содержит от 2/3 до ¾ от общего числа параметров модели –

Содержит основные «знания»

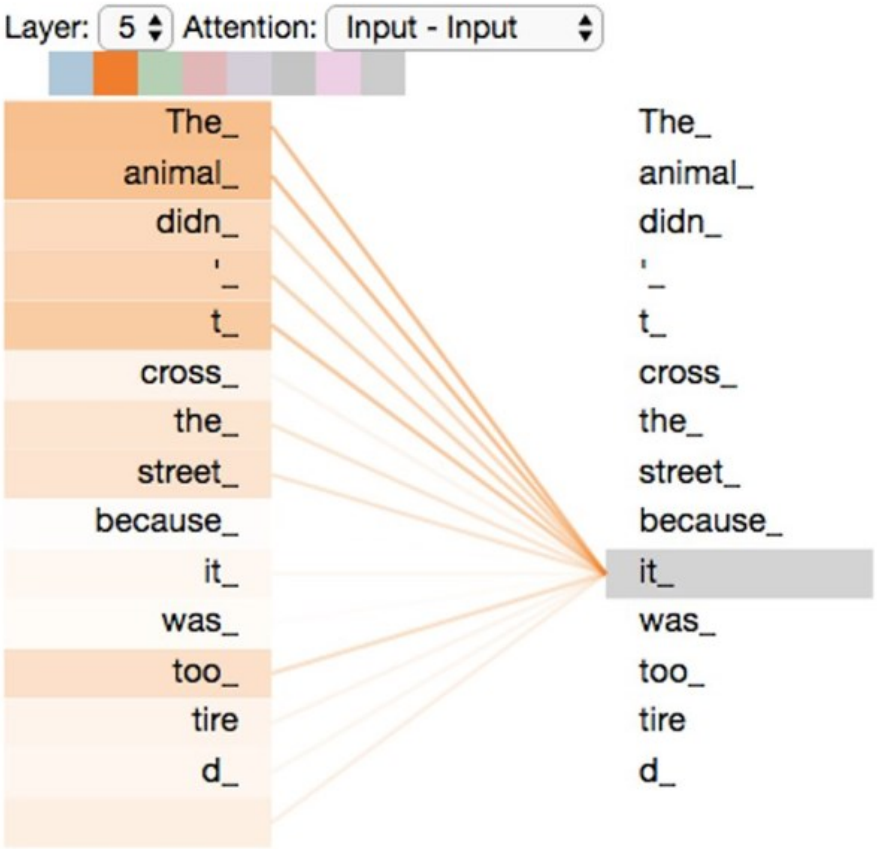
С какими весами нужно взять информацию

Transformer Feed-Forward Layers Are Key-Value Memories: arxiv.org/abs/2012.14913

Feed Forward слой



Self-Attention

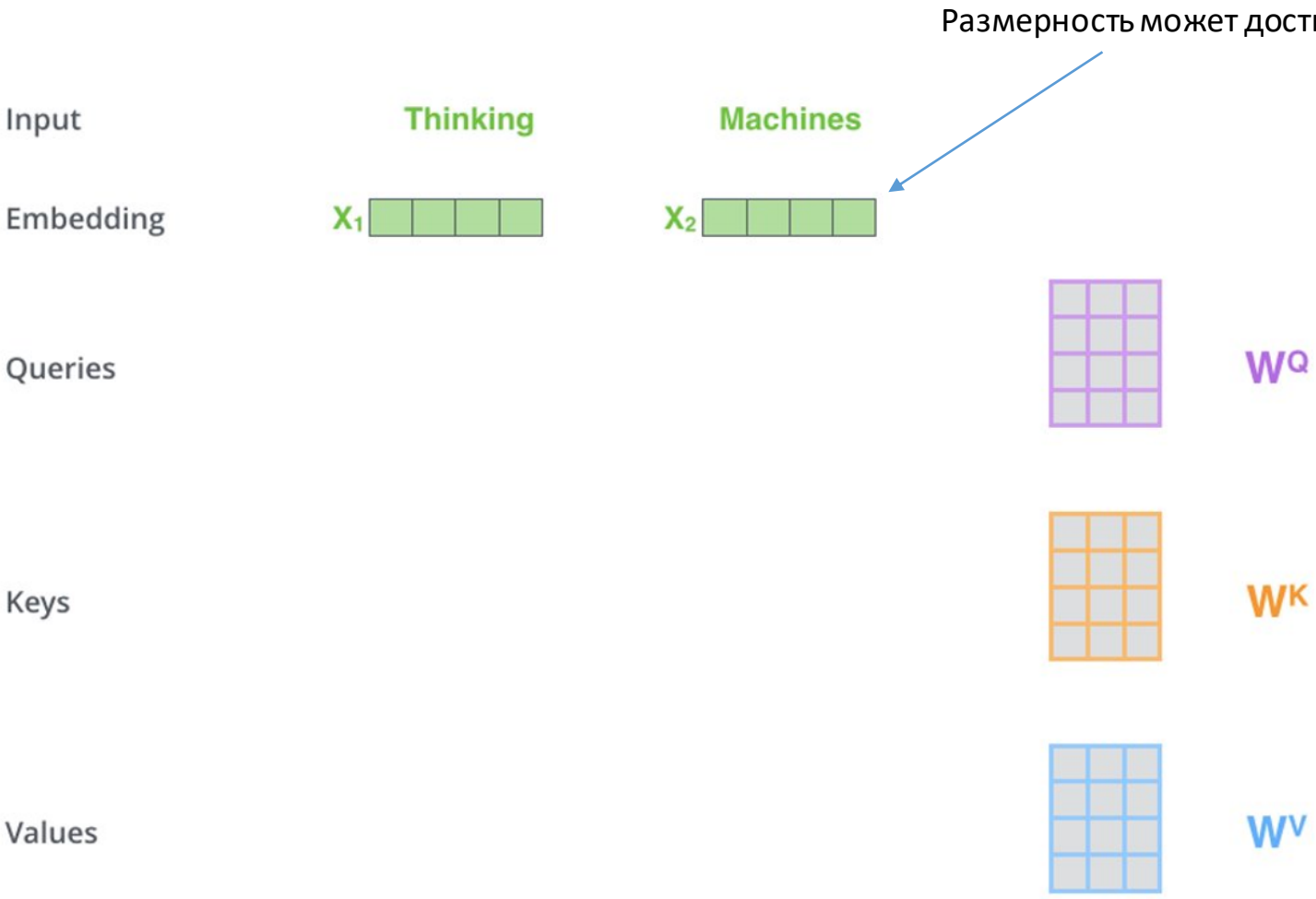


Мы по ходу чтения текста можем предсказывать какое будем следующее слово, т.к. слова зависимы

Формально, Self-Attention предсказывает степень «зависимости» одного слова от другого, пытаясь выявить связи слов в предложении.

Будем предсказывать силу связи между каждым токеном.

Self-Attention



Три (обучаемые) матрицы. Одинаковые для каждого слова, но разные в разных блоках self-attention:

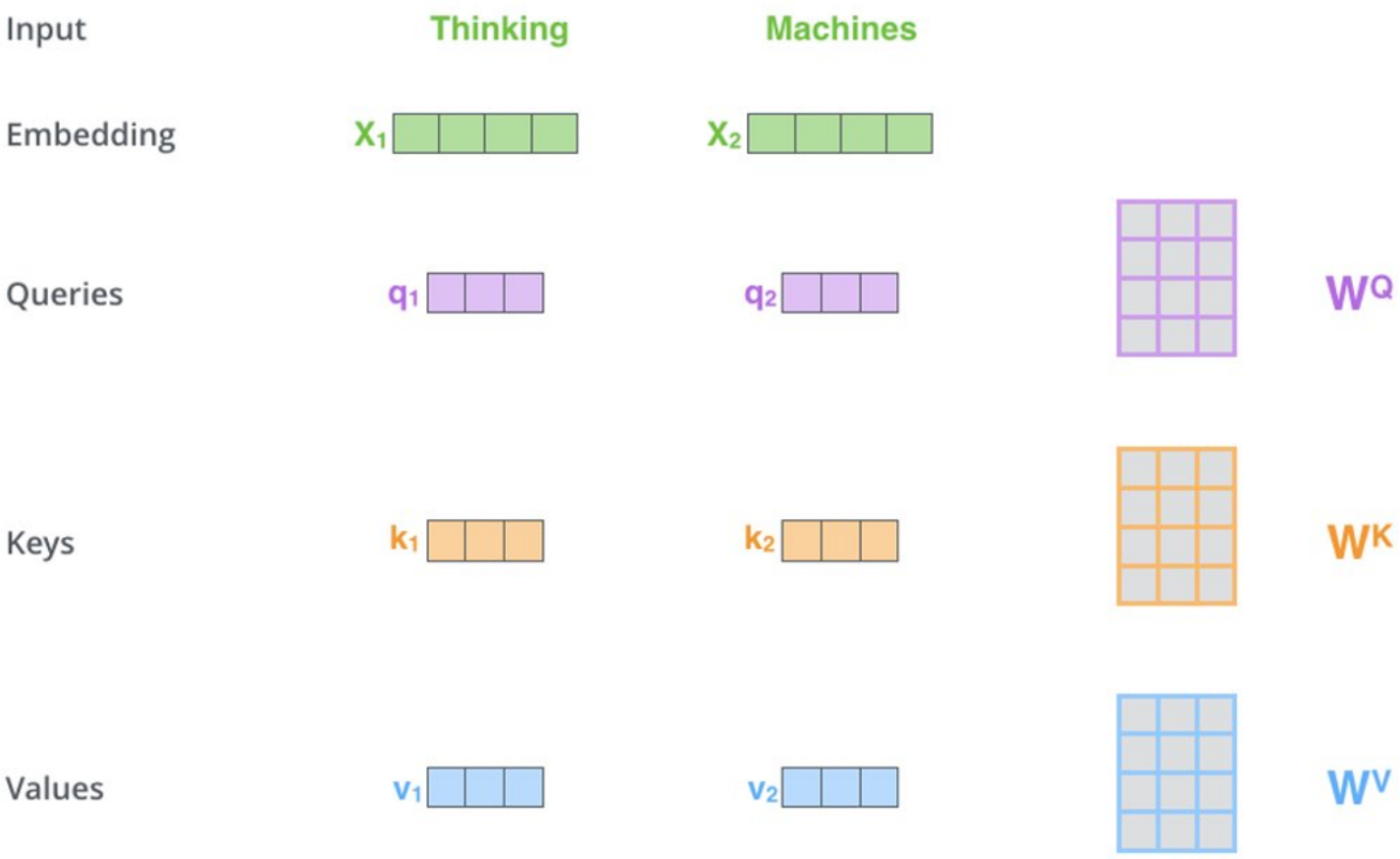
Q – query - что мы хотим получить

K – key – базовое (краткое) значение слова

V – value

Перемножаем эмбеддинг каждого слова X_i с каждой матрицей W

Self-Attention



Три (обучаемые) матрицы. Одинаковые для каждого слова, но разные в разных блоках self-attention:

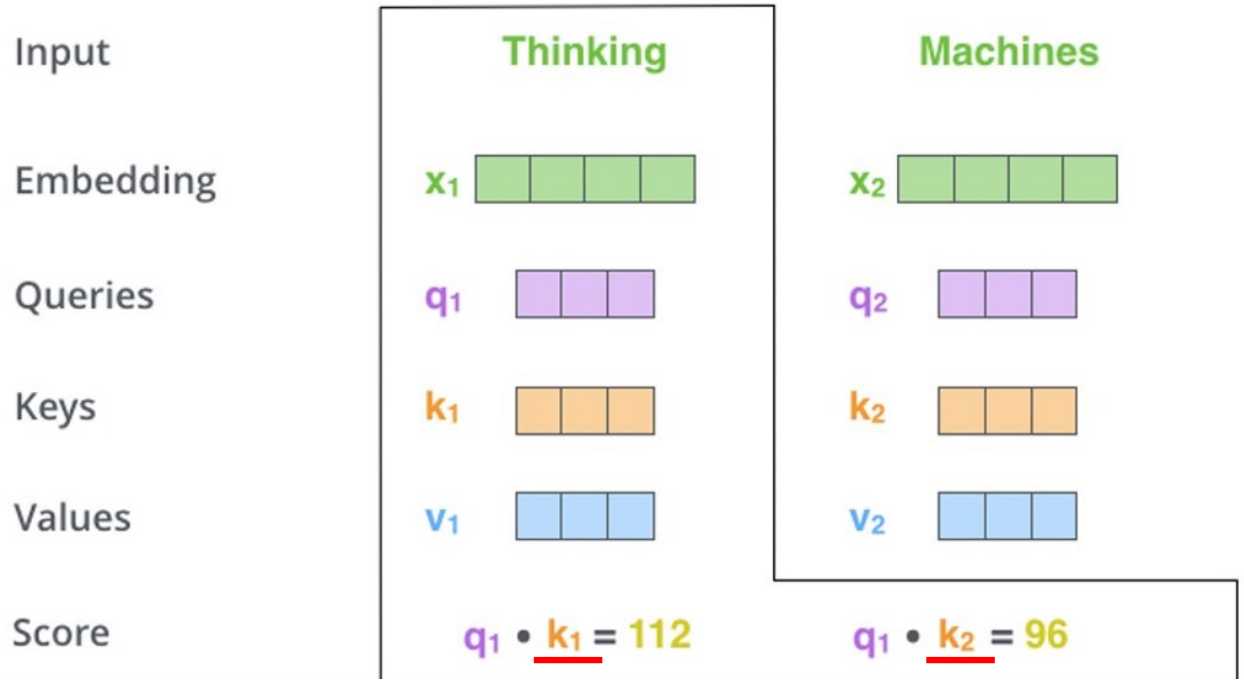
Q – query - что мы хотим получить

K – key – базовое (краткое) значение слова

V – value

Поскольку X_i разные, получаем разные вектора q_i, k_i, v_i для каждого слова.

Self-Attention



Для каждого слова считаем скалярное произведение между QUERY самого слова и KEYS каждого слова в предложении.
Чем оно больше, тем больше key_i подходит для запроса q_1

Скалярное произведение (dot-product):

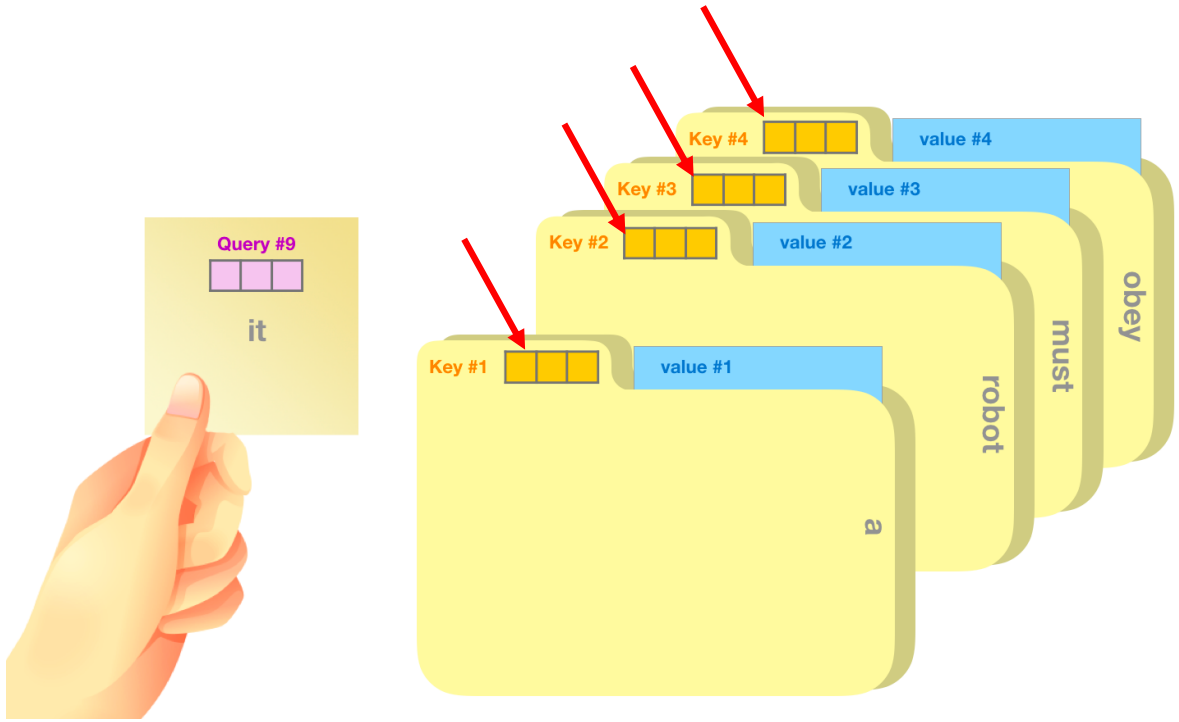
$$(1,3,-5)\cdot(4,-2,-1) = 1\cdot 4 + 3\cdot(-2) + (-5)\cdot(-1)$$
$$= 4 - 6 + 5$$
$$= 3.$$

Query – запрос токена к контексту: что мне нужно найти?
Key – краткое значение слова: что слово подразумевает?

Не нормируем! Чем более «выразительны», тем больше произведение.

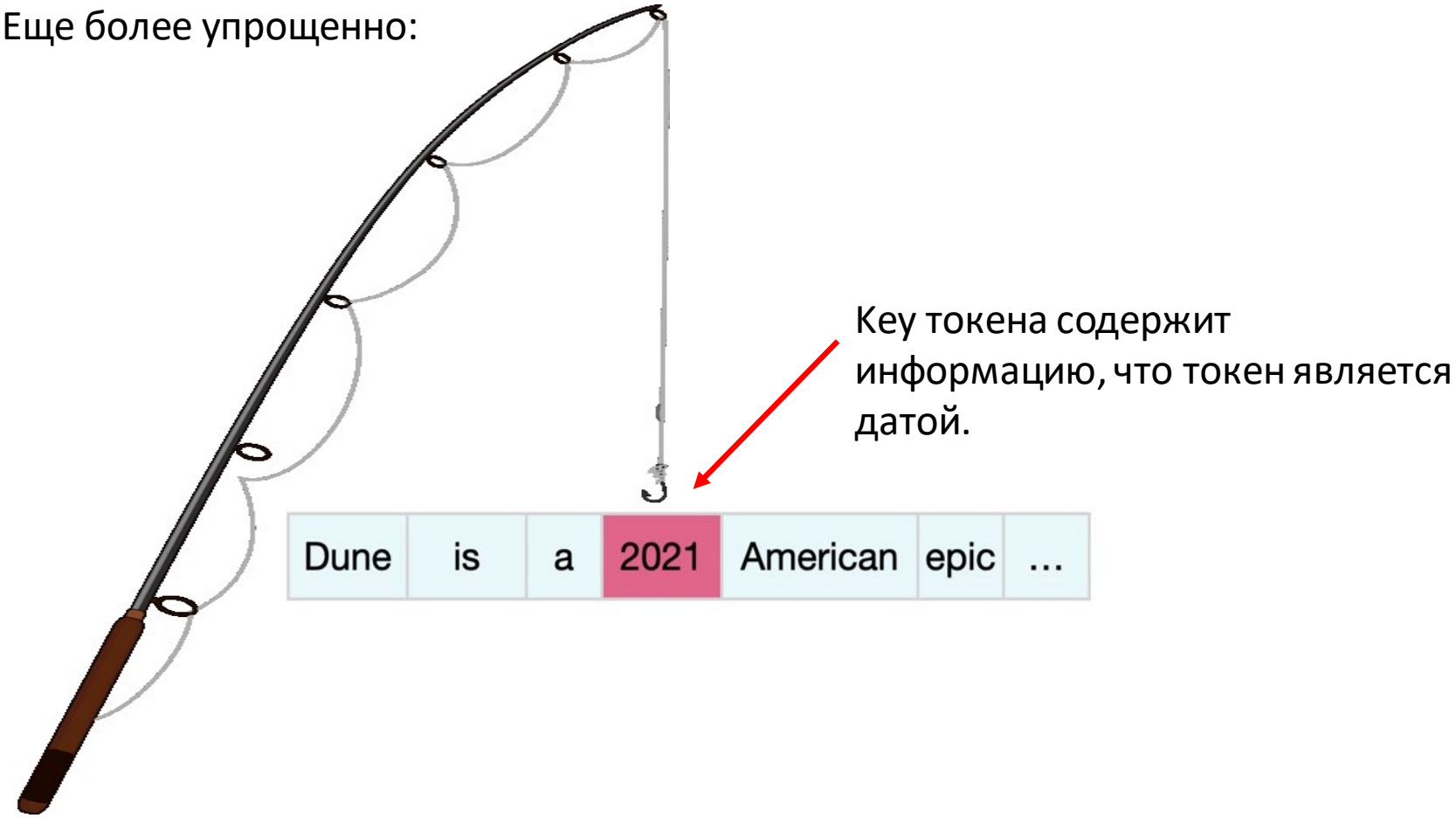
Self-Attention

Упрощенно:



Self-Attention

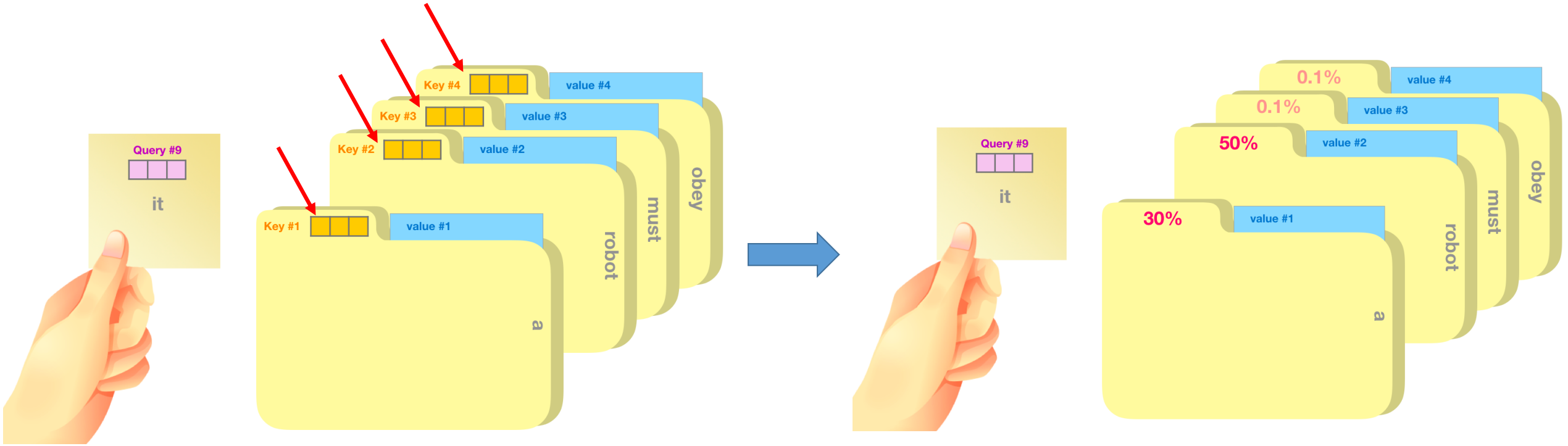
Еще более упрощенно:



Мне нужна дата!

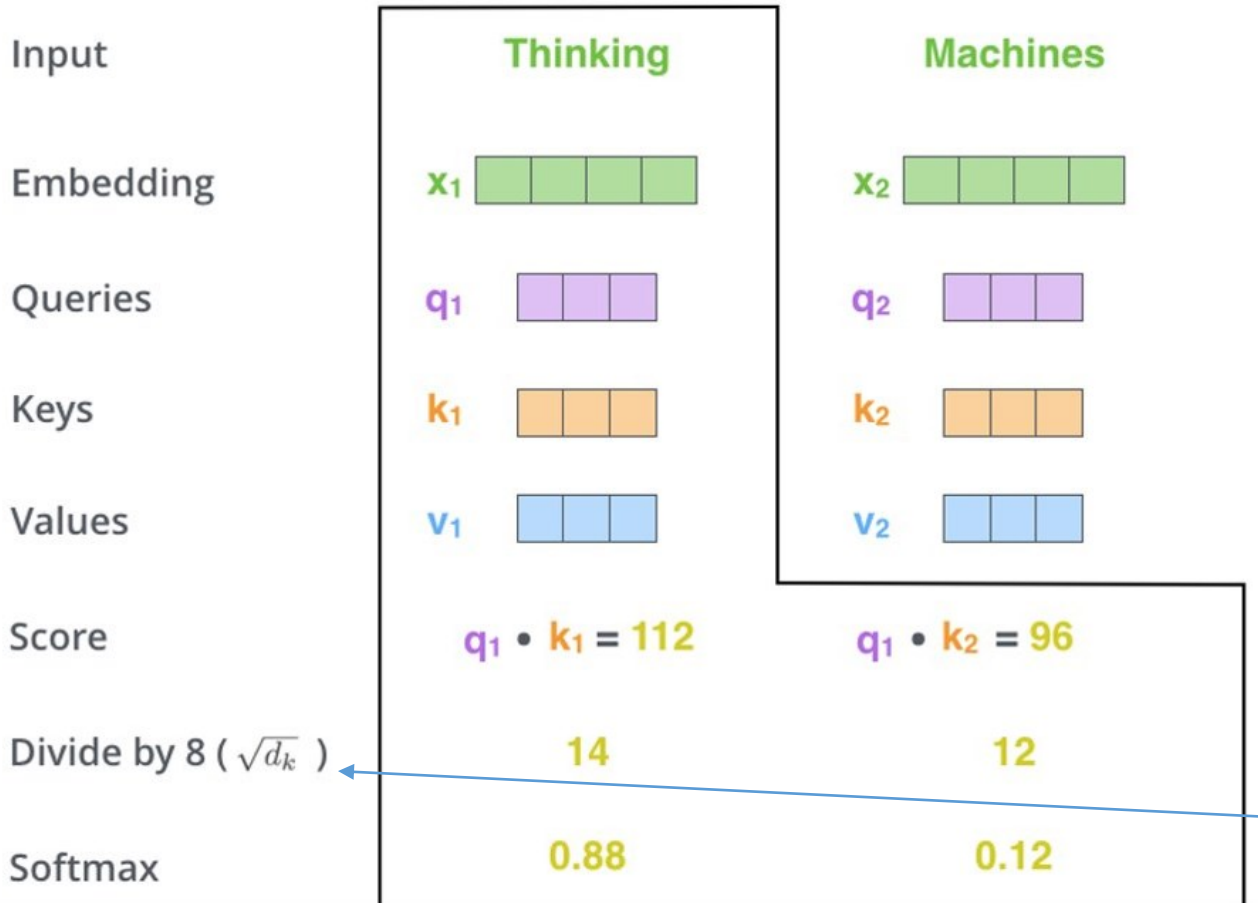
Self-Attention

Упрощенно:



После того, как нашли нужную папку, которая больше всего подходит к запросу, достаем из этой папки информацию value

Self-Attention




```
A = np.random.randn(100, 300) Two matrices, from
B = np.random.randn(100, 300) normal distribution

A_std = A.std()
B_std = B.std()  Their STDs

A_x_B = A @ B.T  Matrix Multiplication
A_x_B_std = A_x_B.std()  Its STD

A_x_B_std_adj = A_x_B_std / (A.shape[1] ** 0.5)

print(f'{A_std=:.02f}, {B_std=:.02f}')
print(f'{A_x_B_std=:.02f}')
print(f'{A_x_B_std_adj=:.02f}')
```

```
A_std=1.00, B_std=1.00  All OK
A_x_B_std=17.29  ???
A_x_B_std_adj=1.00
```

Корень размерности эмбеддинга модели. Нормировка для более стабильных вычислений (убираем затухание\взрыв градиента)

Query – запрос токена к контексту: что мне нужно найти?

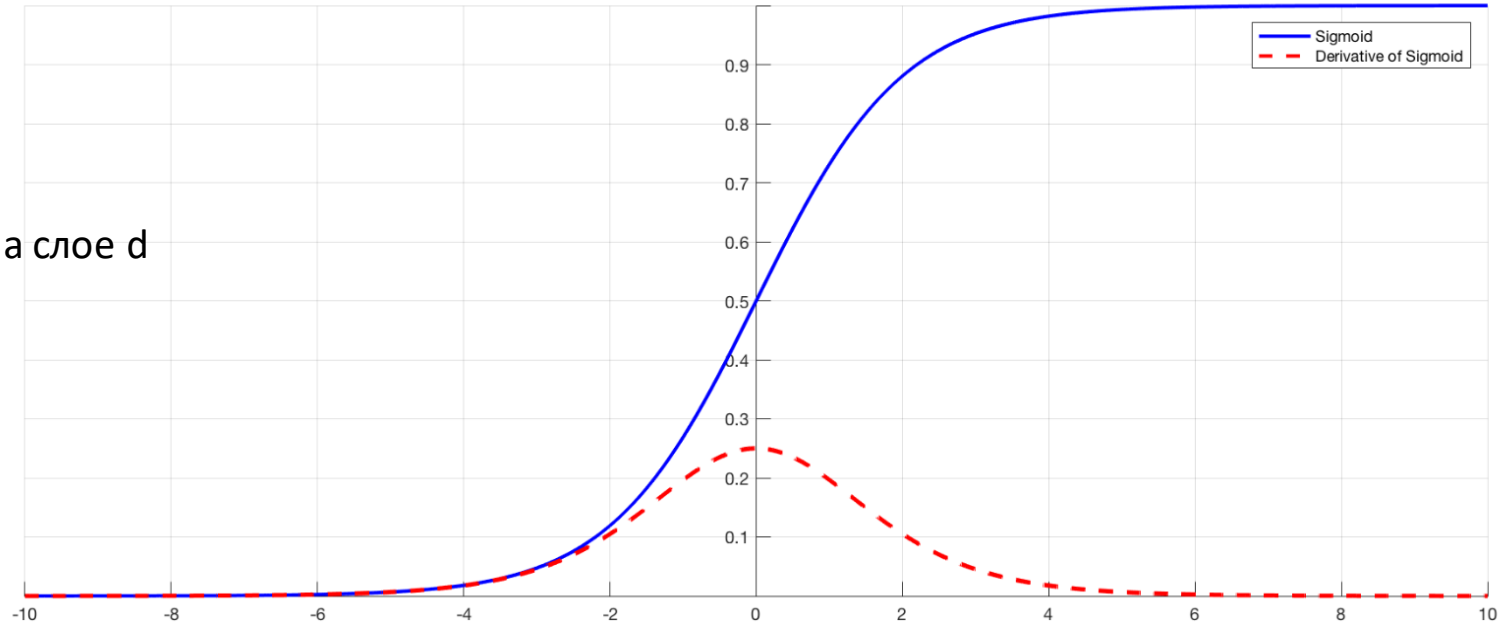
Key – краткое значение слова: что слово подразумевает?

Взрывающийся и затухающий градиент

Пусть сеть – подряд идущие нейроны,
а функция потерь:

$$L(y) = MSE(y, \hat{y}) = (y - \hat{y})^2$$

u_d - значение, поступающее на вход нейрону на слое d
w_d - вес нейрона на слое d
y - выход из последнего слоя



Оценим частные производные по весам
такой нейронной сети на каждом слое

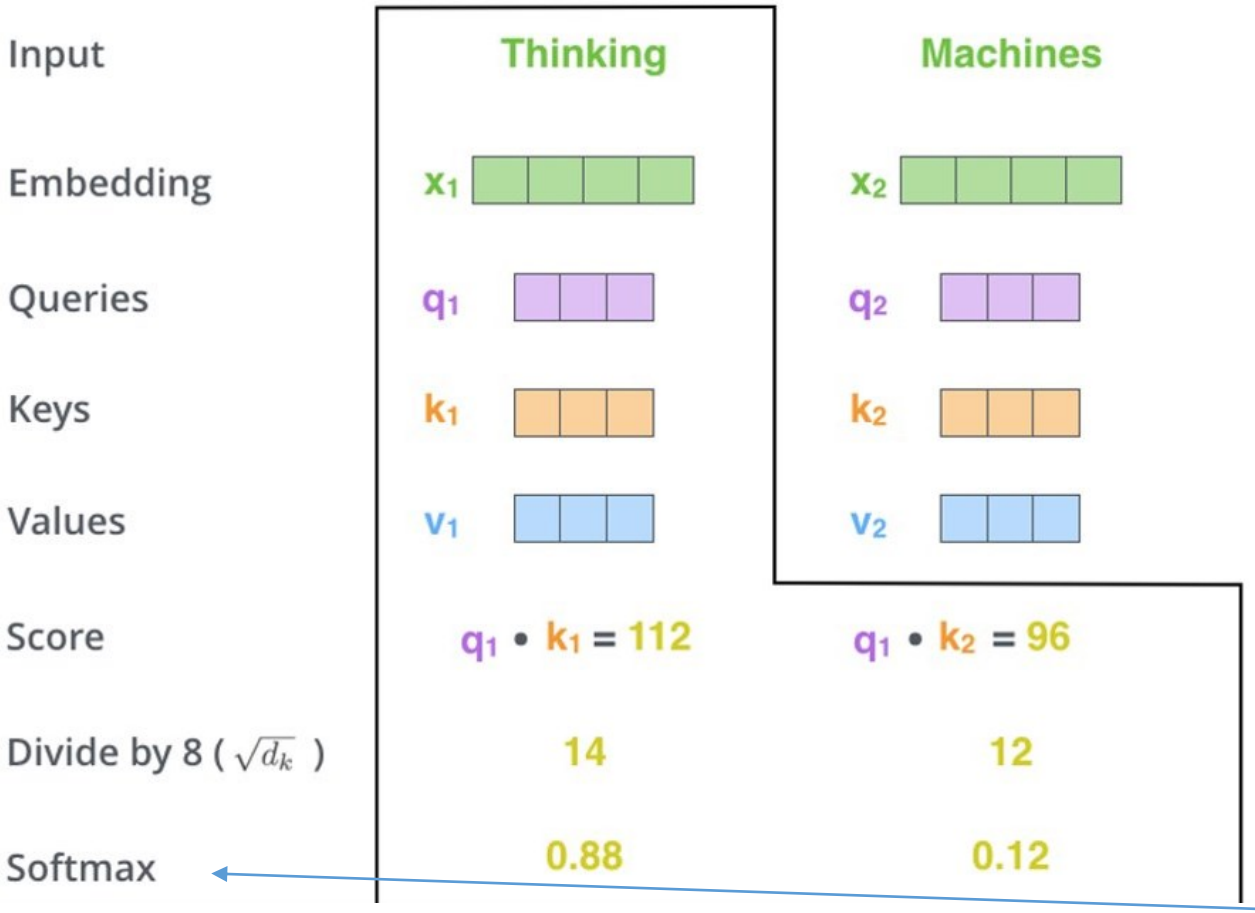
$$\frac{\partial(L(y))}{\partial(w_d)} = \frac{\partial(L(y))}{\partial(y)} \cdot \frac{\partial(y)}{\partial(w_d)} = 2(y - \hat{y}) \cdot \sigma'(w_d u_d) u_d \leq 2(y - \hat{y}) \cdot \frac{1}{4} u_d$$

$$\frac{\partial(L(y))}{\partial(w_{d-1})} = \frac{\partial(L(y))}{\partial(w_d)} \cdot \frac{\partial(w_d)}{\partial(w_{d-1})} \leq 2(y - \hat{y}) \cdot (\frac{1}{4})^2 u_d u_{d-1}$$

...

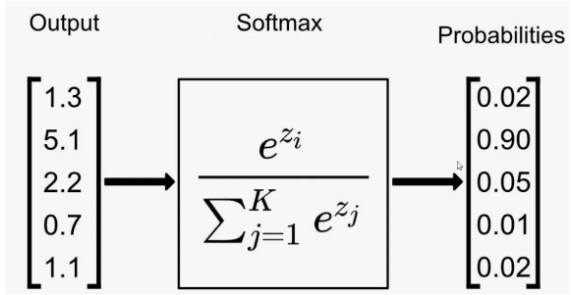
Видно, что оценка элементов градиента растет экспоненциально при движении ко входу НС. Это в свою очередь может приводить либо к экспоненциальному росту градиента от слоя к слою, когда входные значения нейронов — числа, по модулю большие 1, либо к затуханию, когда эти значения — числа, по модулю меньшие 1

Self-Attention




```
A = np.random.randn(100, 300) Two matrices, from
B = np.random.randn(100, 300) normal distribution

A_std = A.std()
B_std = B.std()  Their STDs

A_x_B = A @ B.T  Matrix Multiplication
A_x_B_std = A_x_B.std()  Its STD

A_x_B_std_adj = A_x_B_std / (A.shape[1] ** 0.5)

print(f'{A_std=:.02f}, {B_std=:.02f}')
print(f'{A_x_B_std=:.02f}')
print(f'{A_x_B_std_adj=:.02f}')
```

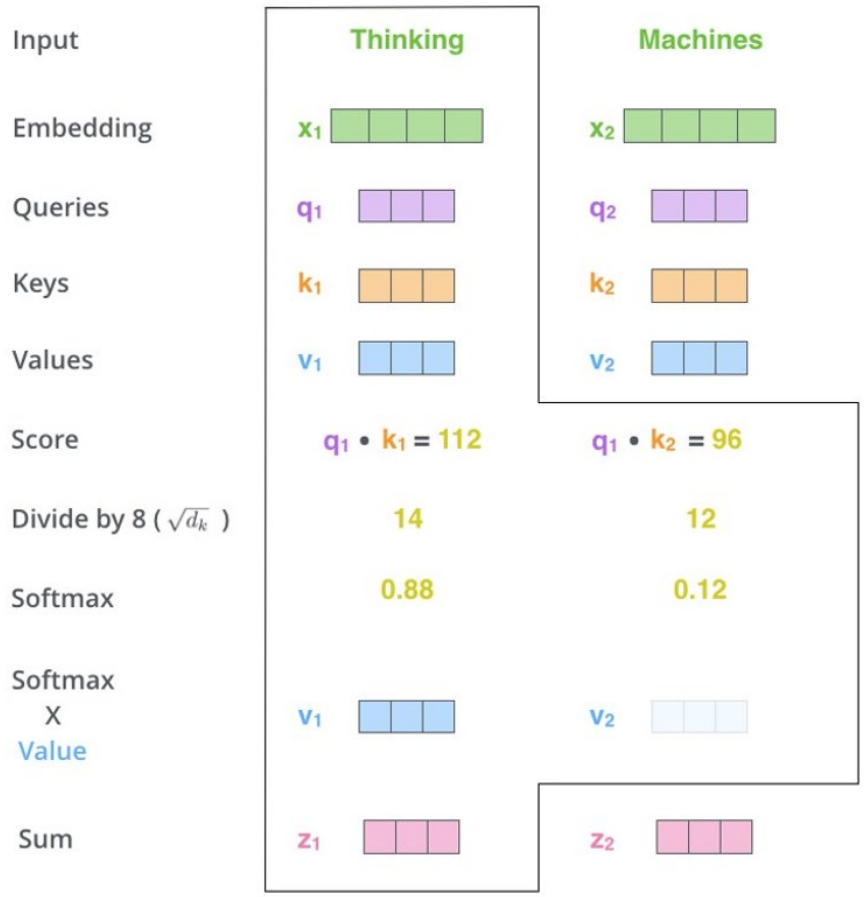
```
A_std=1.00, B_std=1.00  All OK
A_x_B_std=17.29  ???
A_x_B_std_adj=1.00
```



Query – запрос токена к контексту: что мне нужно найти?

Key – краткое значение слова: что слово подразумевает?

Self-Attention

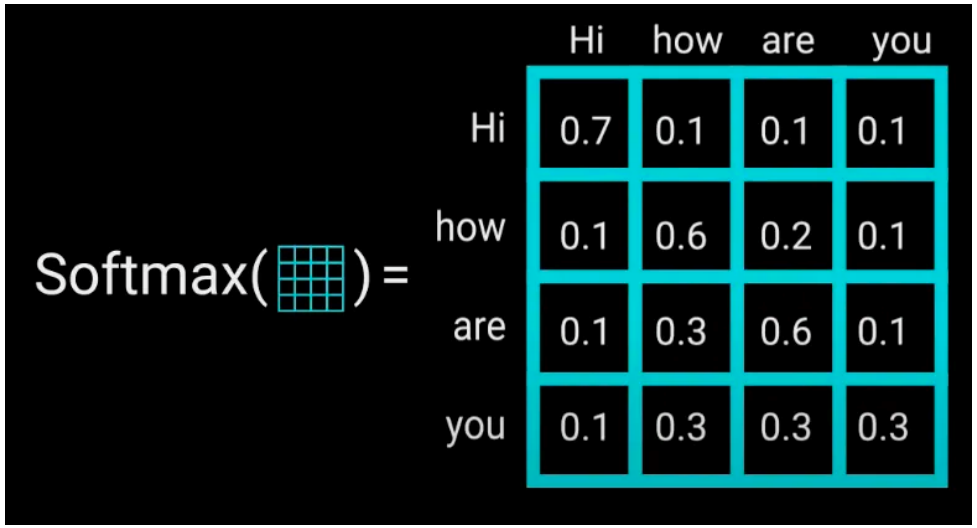


Итоговый эмбеддинг представляется как взвешенная сумма эмбеддингов value.
Взвешенная по значению softmax

Query – запрос токена к контексту: что мне нужно найти?
Key – краткое значение слова: что слово подразумевает?
Value – полное значение слова – что я скажу тебе, если ты обратишься с запросом query. Размер может отличаться от query и key

Self-Attention

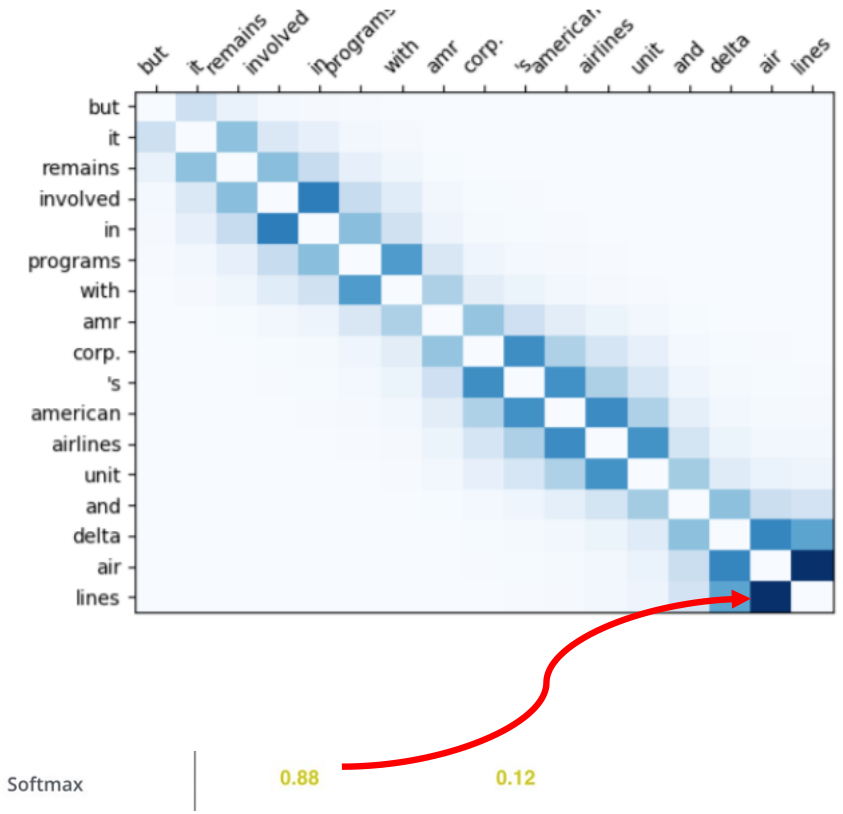
Вычислительная сложность O(n^2), где n количество входных токенов, потому что матрица весов рассчитывается для каждого токена с каждым.

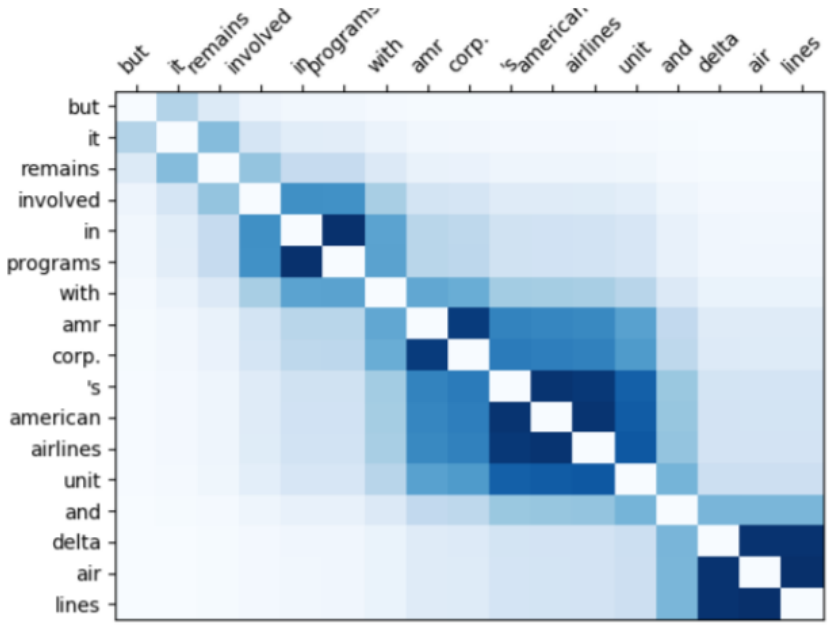


В матрице каждое значение – результат перемножения query для слова из строки и key для слова из столбца.
Чем длиннее последовательность, которую хотим обработать, тем больше (квадратично) количество требуемых вычислений.
Поэтому важно количество токенов. Чем лучше токенайзер, тем короче последовательность, которую нужно обработать

Attention matrix

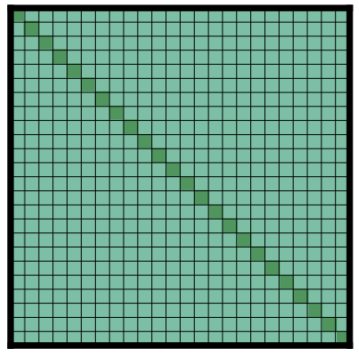
Для разных слоев внимание может быть разным



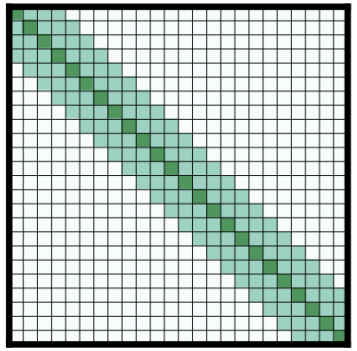


Attention matrix

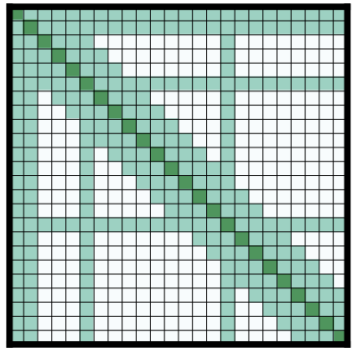
- **(Used in GPT-3) Generating Long Sequences with Sparse Transformers:** arxiv.org/abs/1904.10509
- **Reformer: The Efficient Transformer:** arxiv.org/abs/2001.04451
- **Longformer: The Long-Document Transformer:** arxiv.org/abs/2004.05150
- **Linformer: Self-Attention with Linear Complexity:** arxiv.org/abs/2006.04768
- **Rethinking Attention with Performers:** arxiv.org/abs/2009.14794



(a) Full n^2 attention



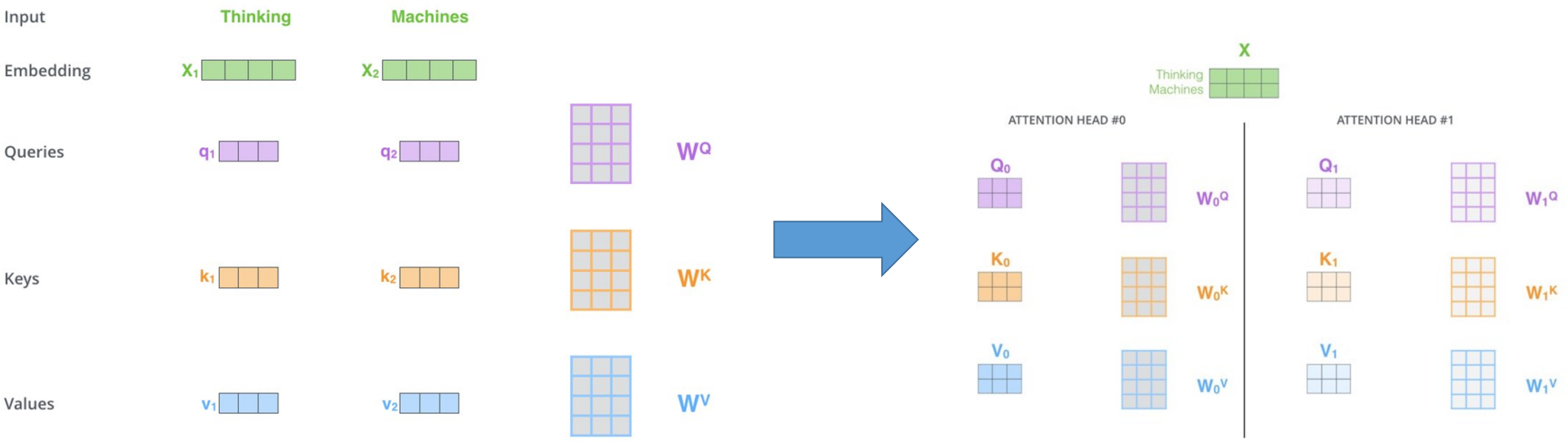
(b) Sliding window attention



(d) Global+sliding window

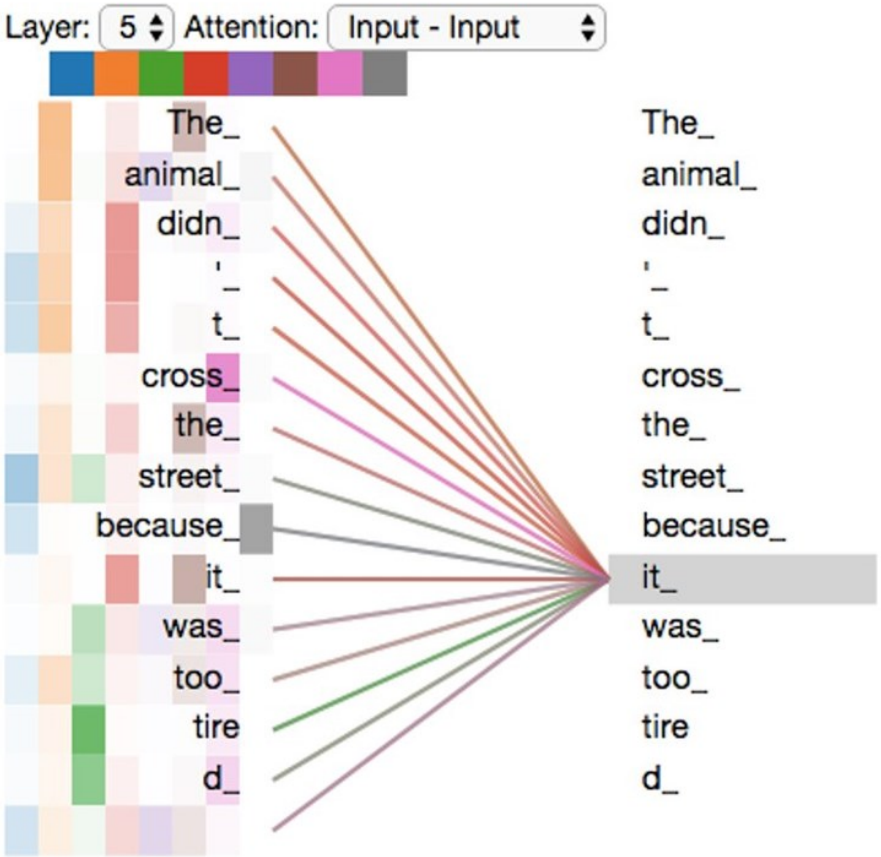
MultiHead Self Attention

Если что-то работает хорошо – сделайте это много раз: Random Forest, XGBoost,... Multihead attention



Есть один триплет матриц. Что если добавить много таких триплетов? Эмбеддинг слова один, но каждая тройка матриц («голова») будет обращать внимание на разные связи в предложении.

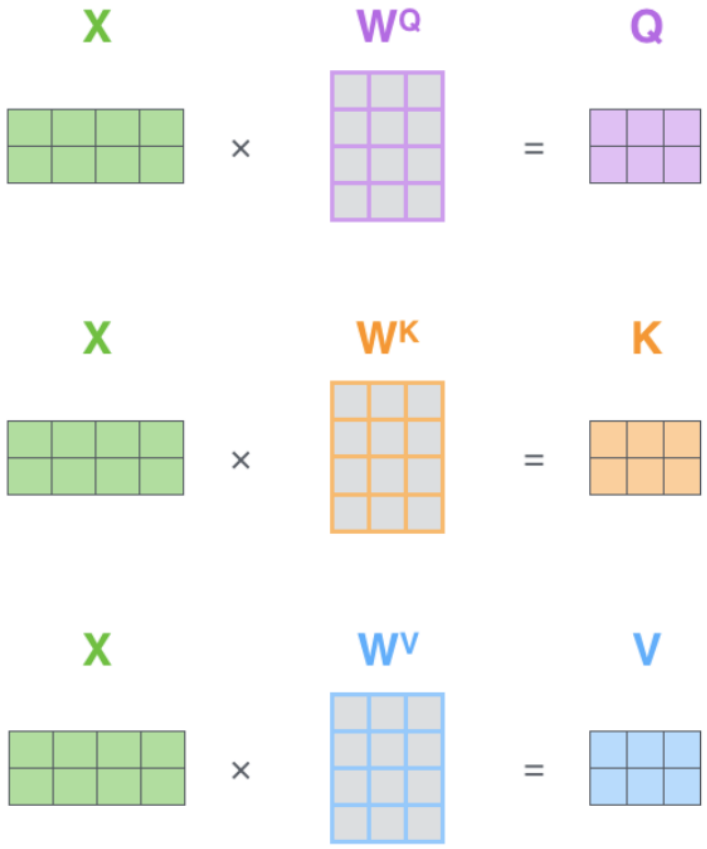
MultiHead Self Attention

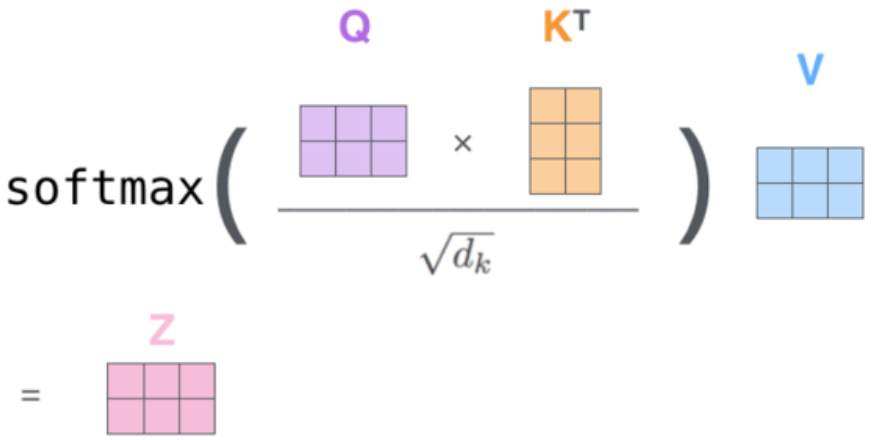


Добавляем много разных триплетов (8 штук). Каждая голова обучается обращать внимание на разные части предложения – ищет свой собственный паттерн среди эмбеддингов

Визуализация

Матричные вычисления

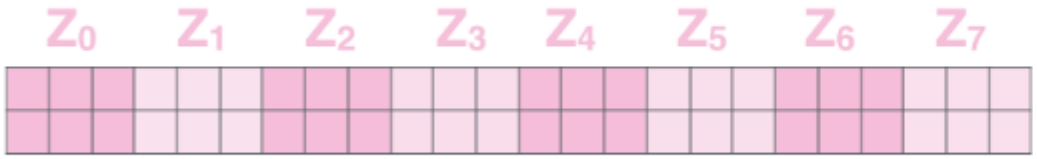




Поскольку все вычисления – матричные, это быстро

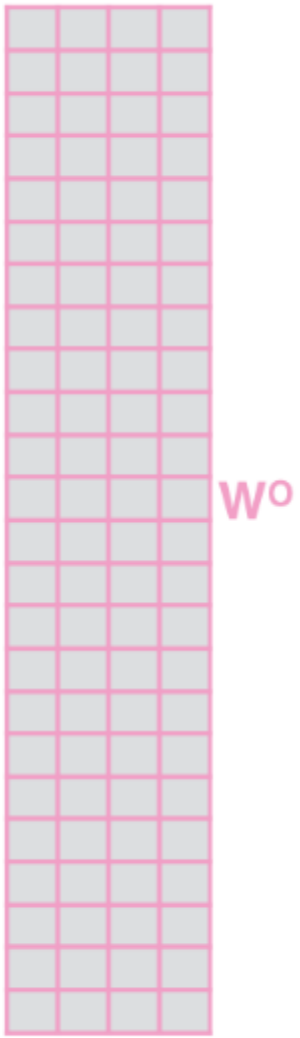
Матричные вычисления

1) Concatenate all the attention heads



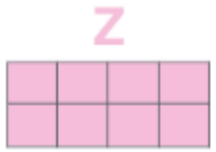
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



На выходе Attention Layer получили матрицу той же размерности что и на входе. Передаем ее в Feed Forward слой.

Шаги работы Self Attention

1) This is our input sentence*

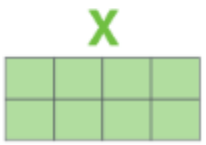
2) We embed each word*

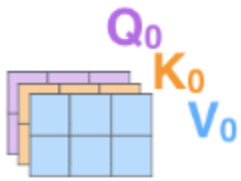
3) Split into 8 heads. We multiply X or R with weight matrices

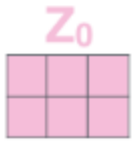
4) Calculate attention using the resulting Q/K/V matrices

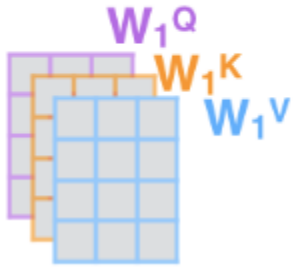
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

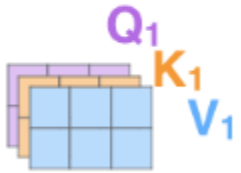
Thinking
Machines

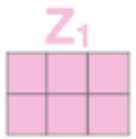








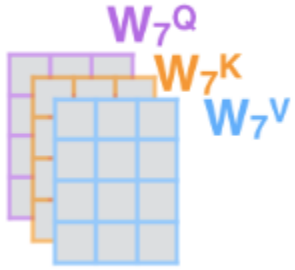


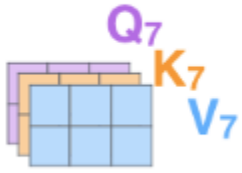


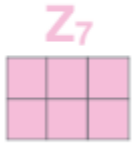
...

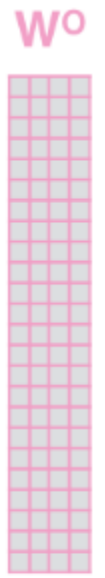
...

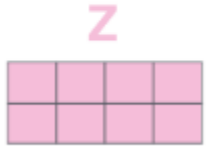
...





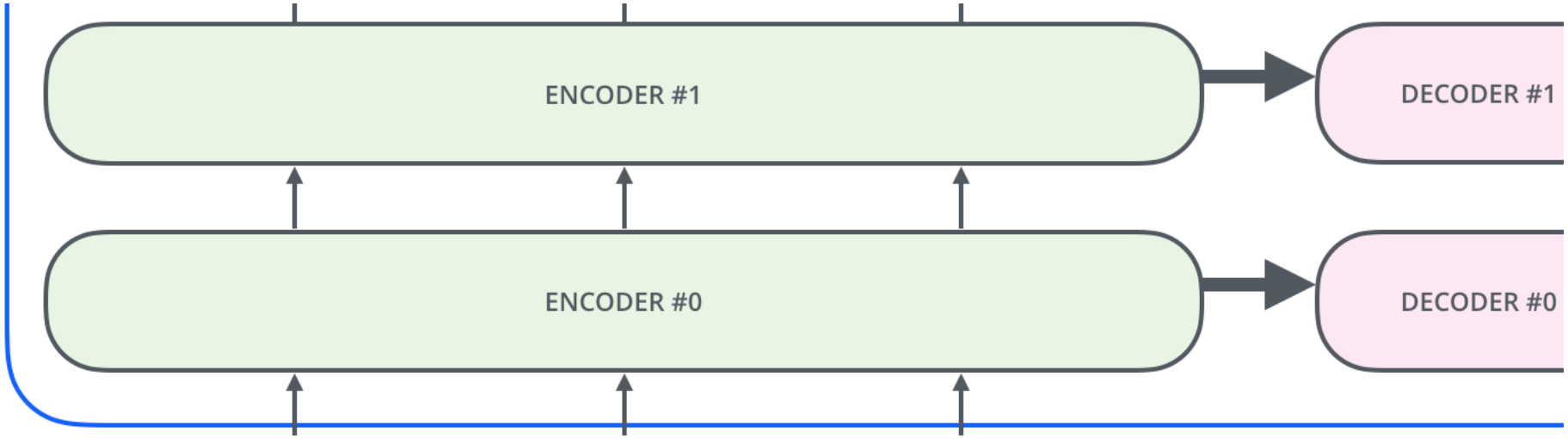


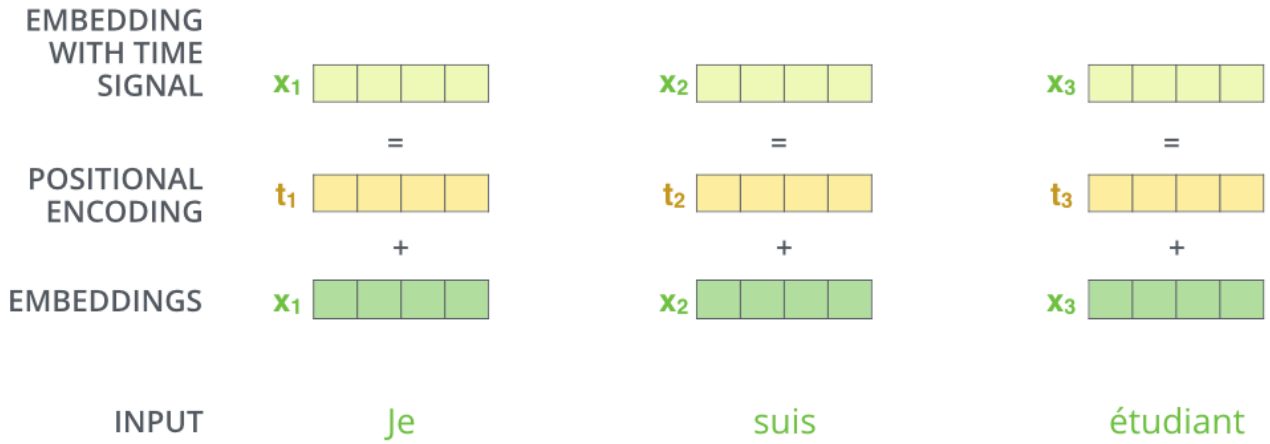




* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Инвариантность к последовательности на входе





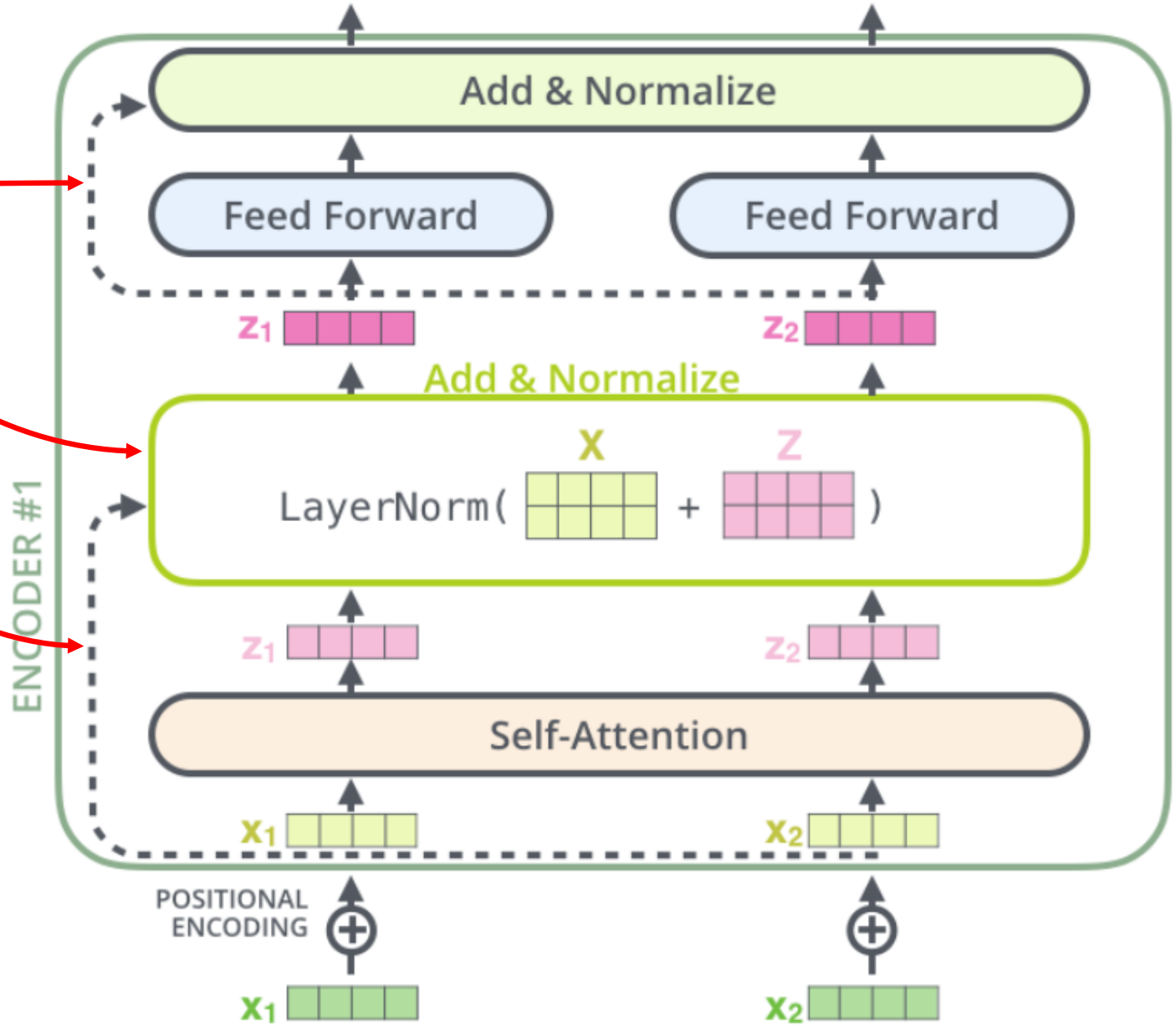
До сих пор у нас не было речи о том, в какой последовательности идут слова в предложении на входе.

Добавляем позиционное кодирование Positional encoding.

Их сложно тренировать => ограничиваем длину входной последовательности

Дополнительные хитрости

- Layer-Norm
- Residuals



Дополнительные хитрости

- Residuals

  X + F(X)

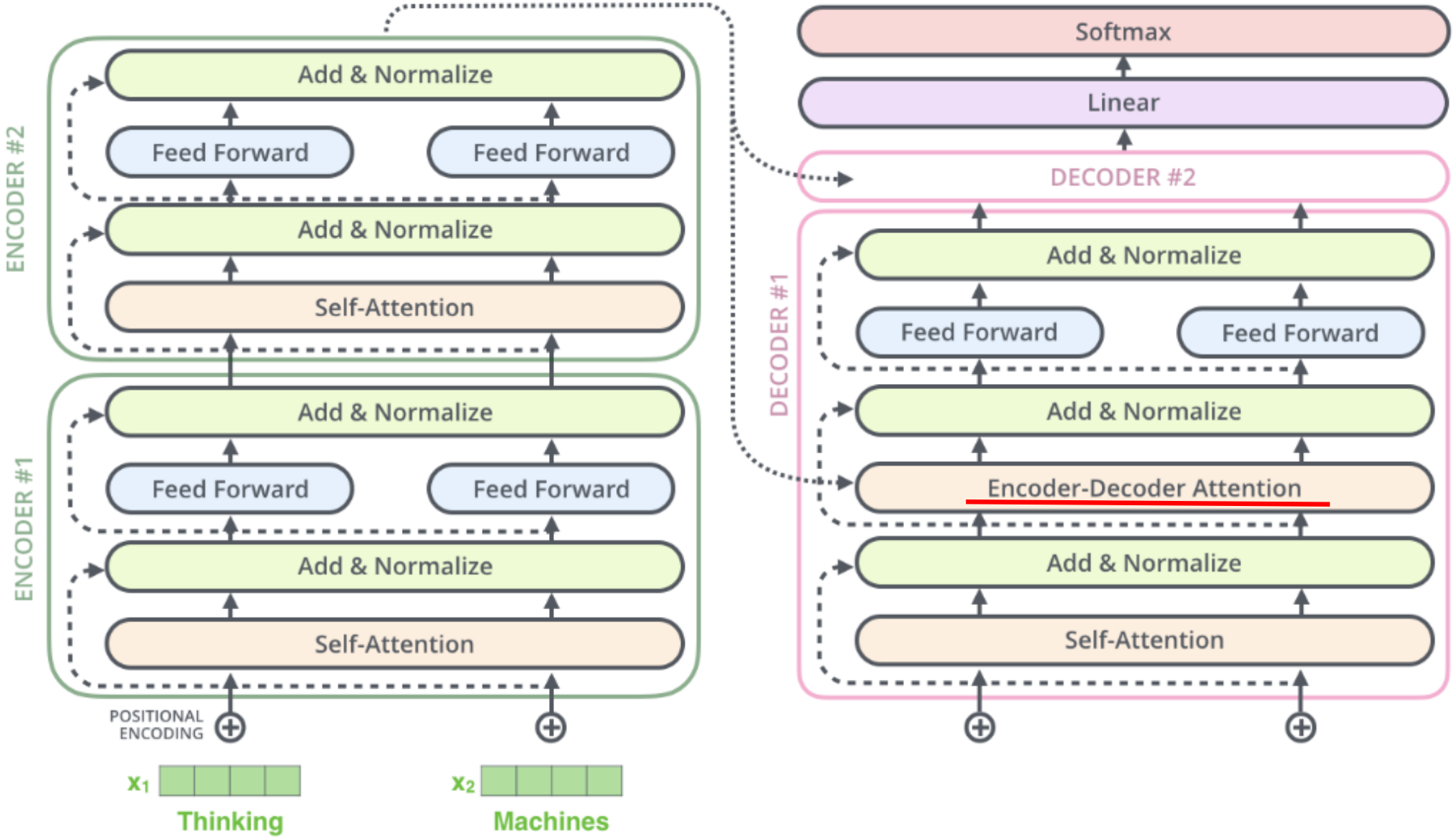
```
def forward(self, x, sublayer):
    "Apply residual connection to any sublayer with the same size."
    return x + self.dropout(sublayer(self.norm(x)))
```

- LayerNorm

  A * (X – mean_X) / (std_X) + b

```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."
    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```

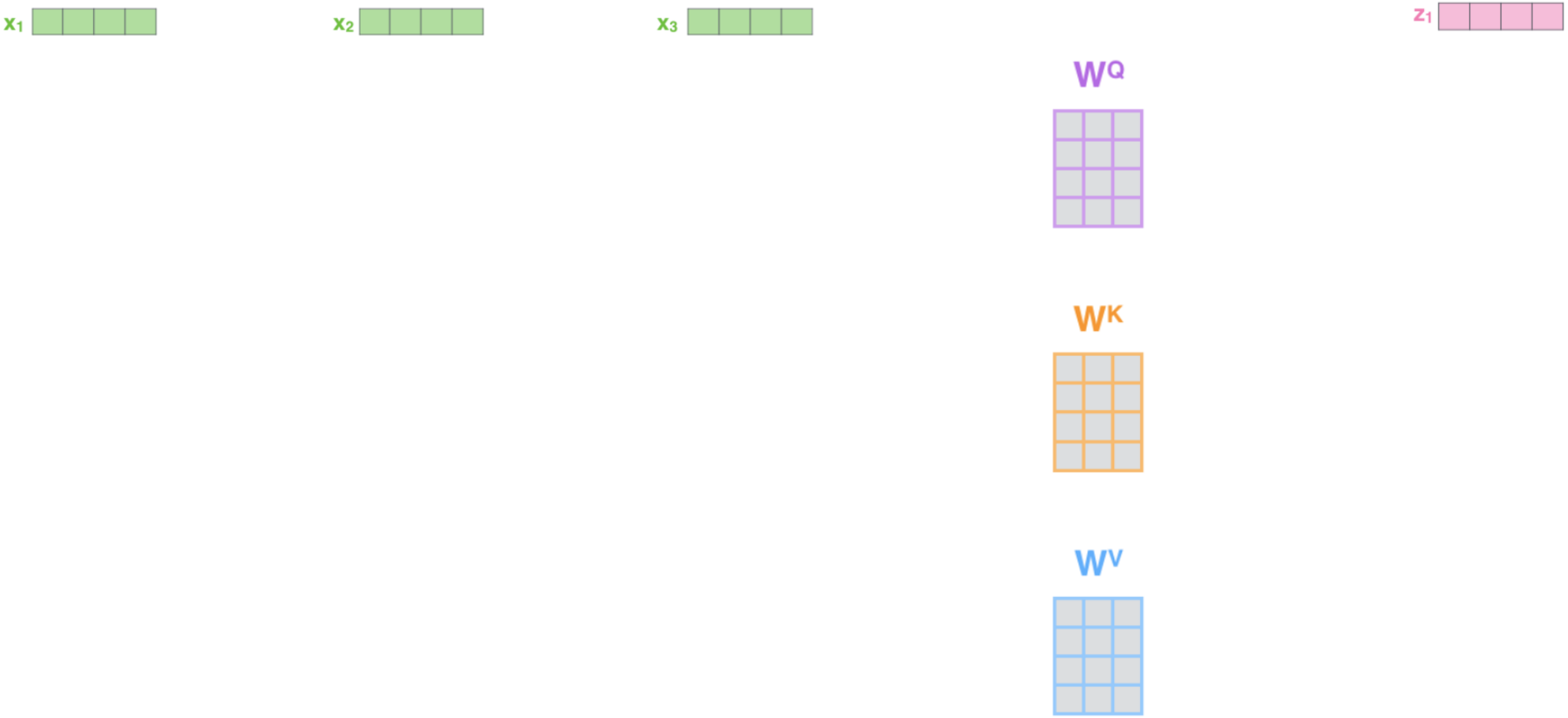
Декодер


Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention
ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
x2
Thinking
Machines

Encoder-Decoder (Cross) attention

Encoder

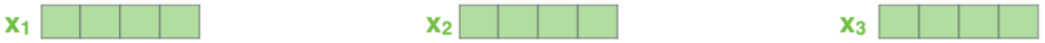
Decoder

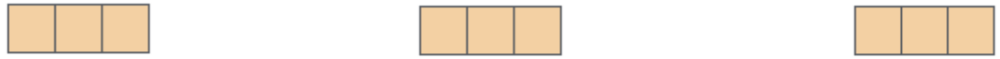


Encoder-Decoder (Cross) attention

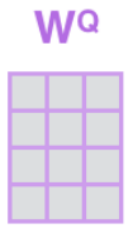
Encoder

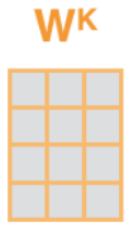
I *study* *ML*

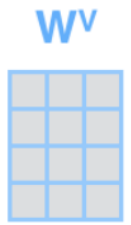




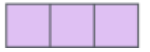
Контекст спрятан в энкодере (он знает весь текст)





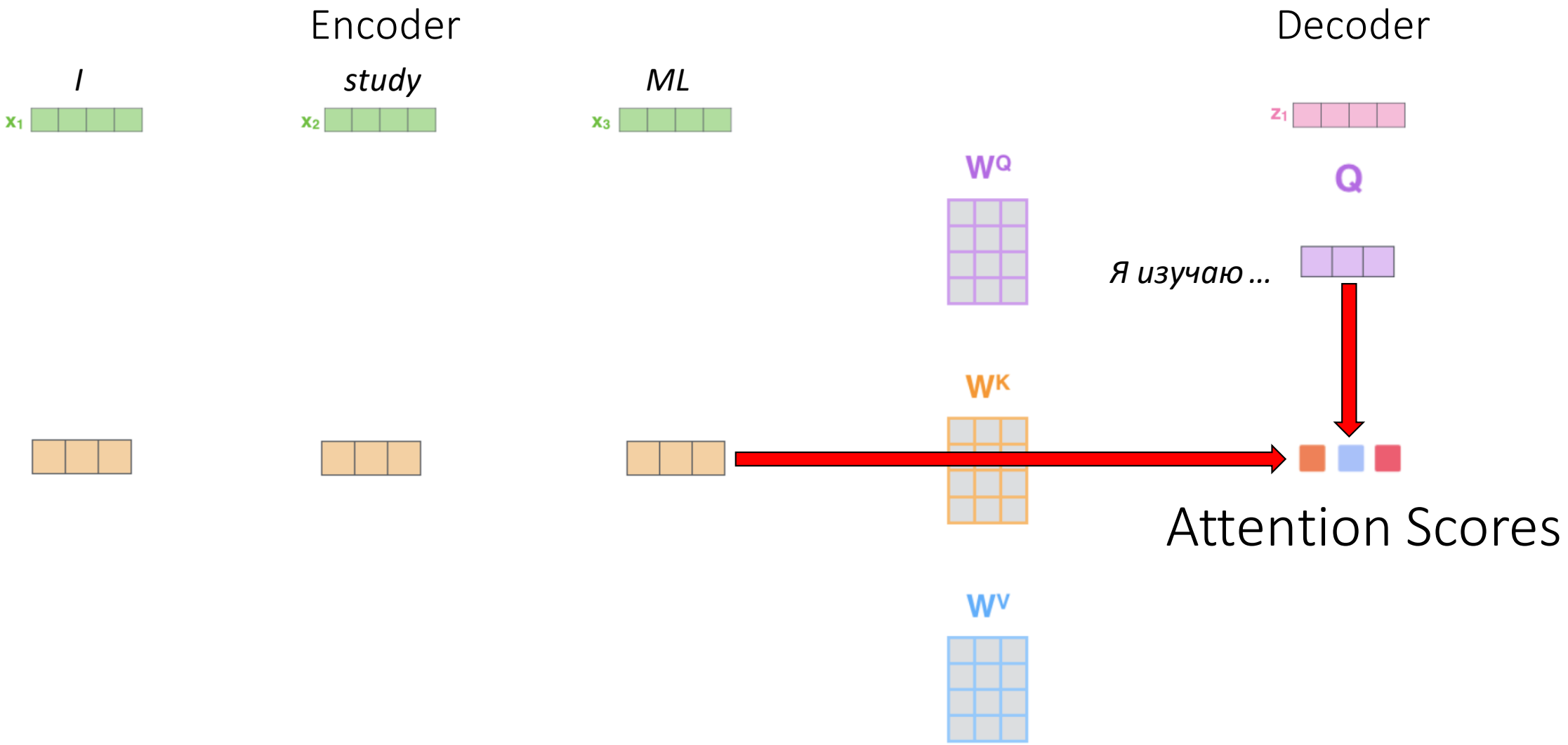


Decoder

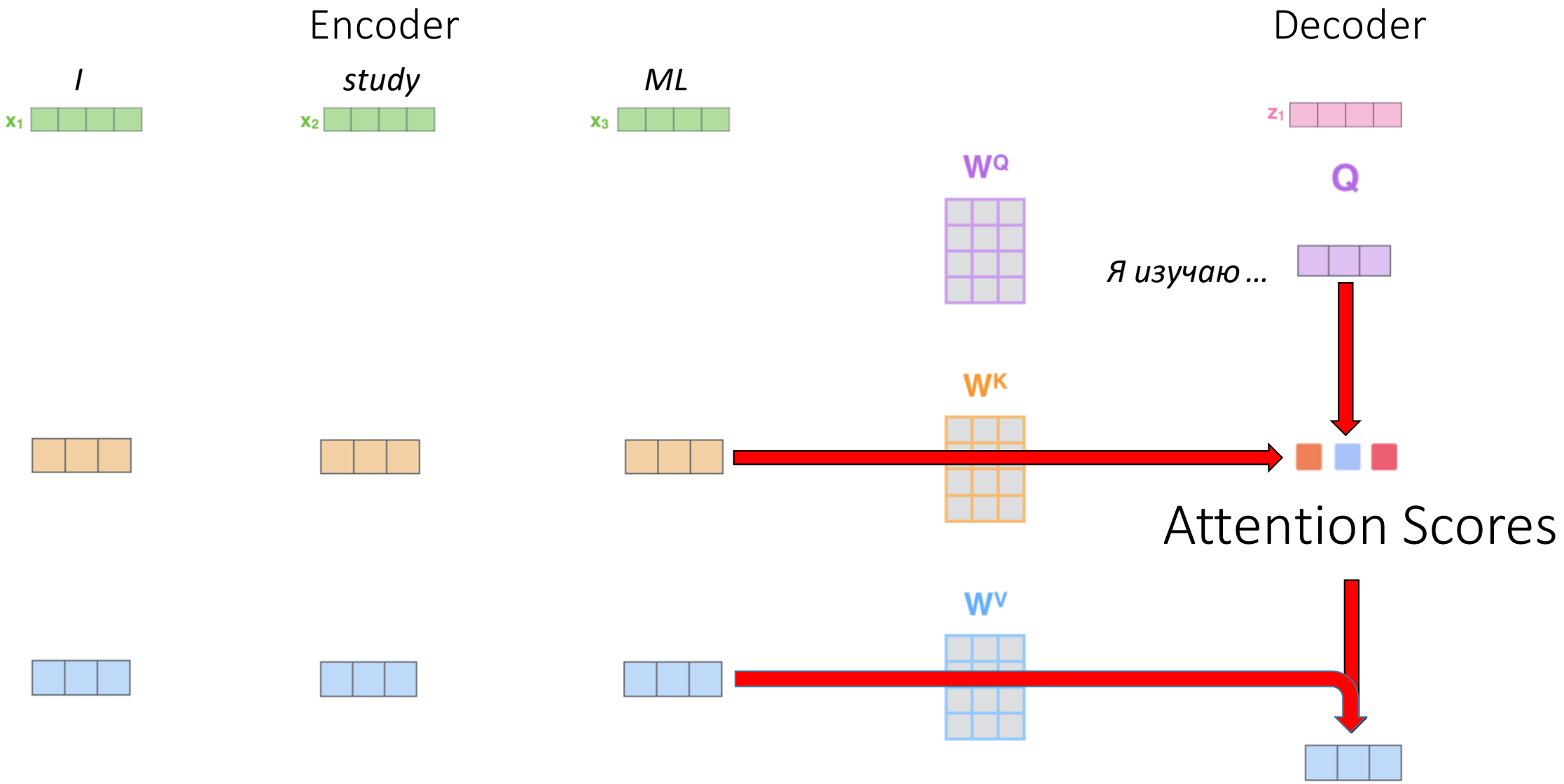
Я изучаю … 

Какое слово писать дальше?

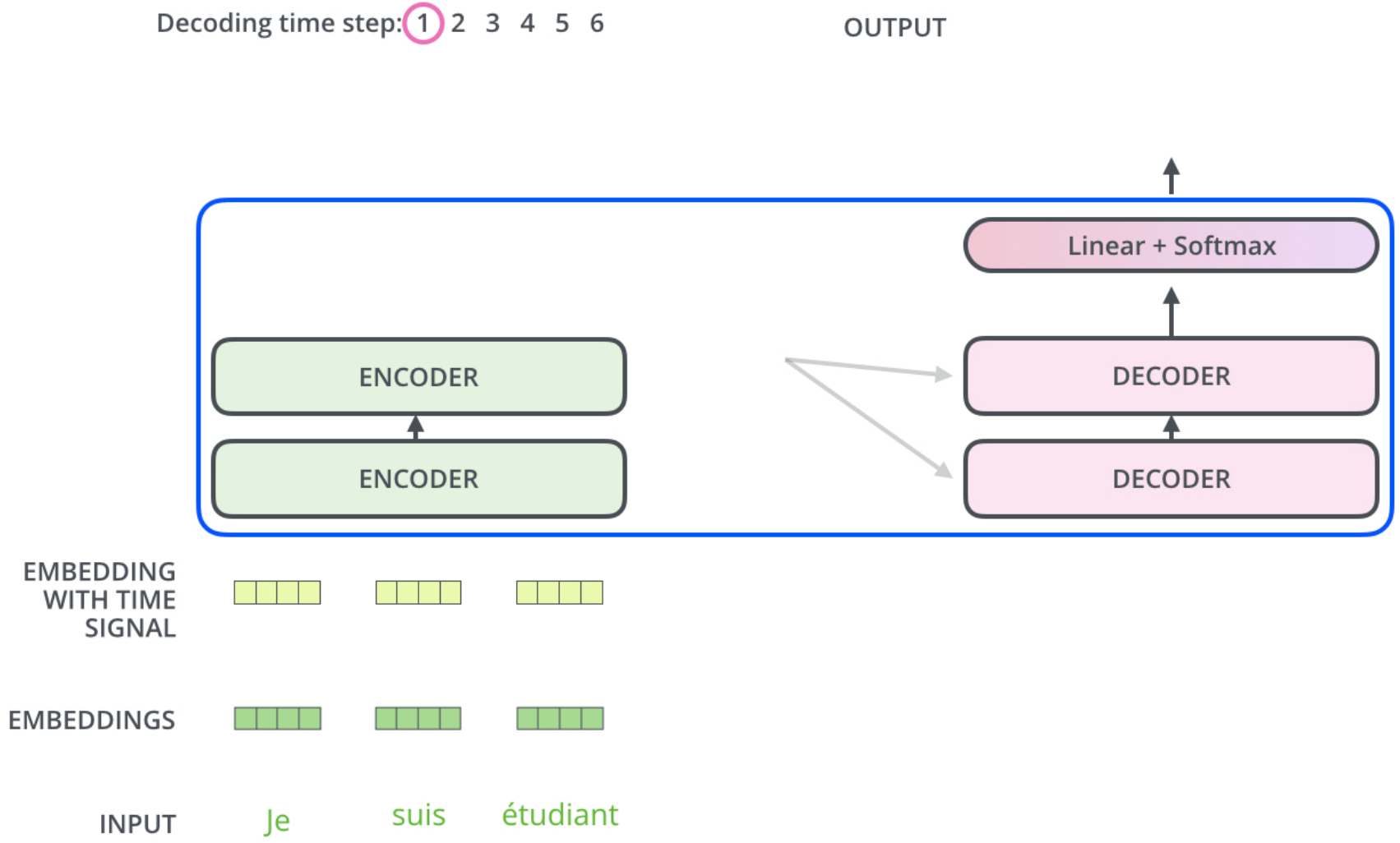
Encoder-Decoder (Cross) attention



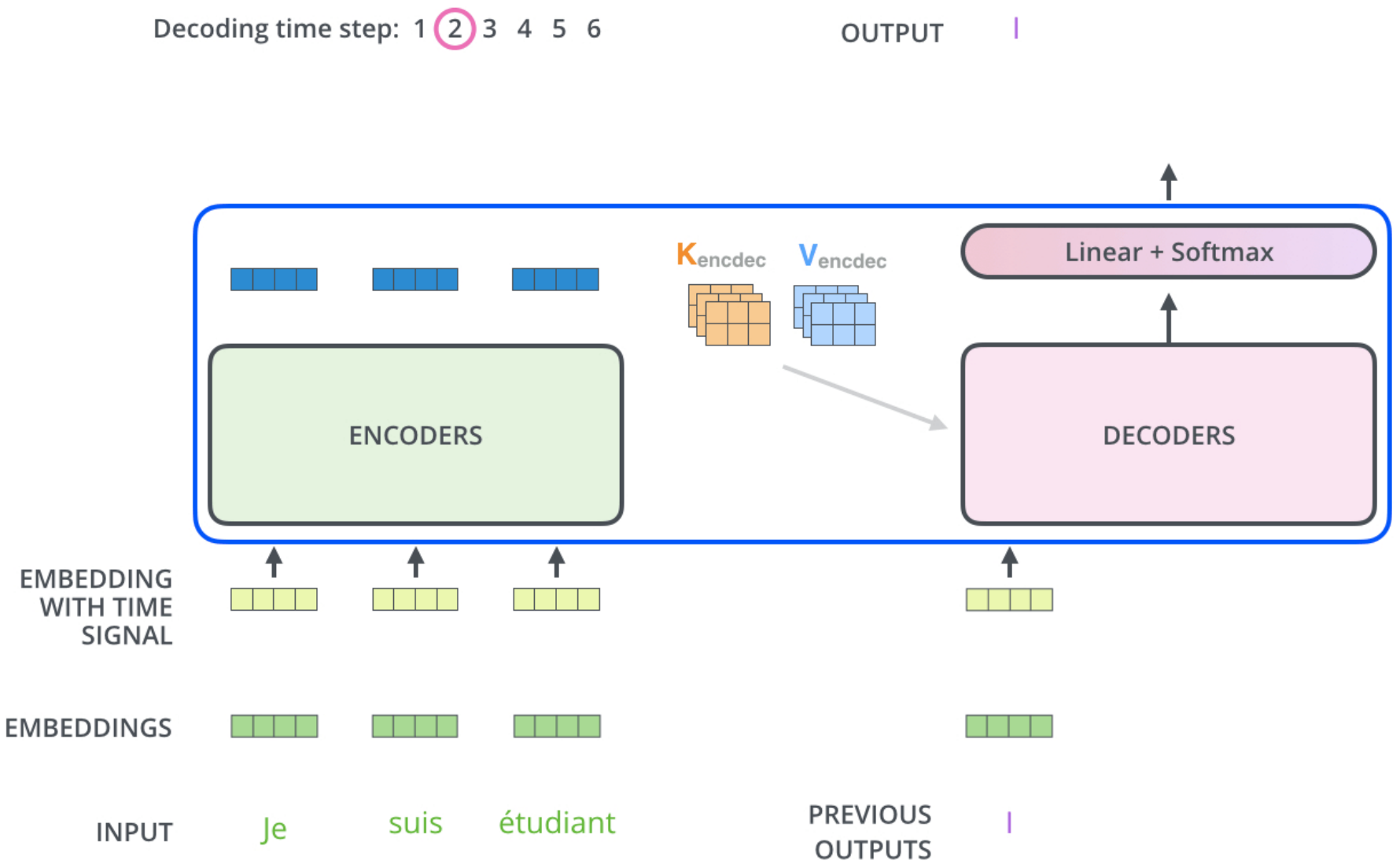
Encoder-Decoder (Cross) attention



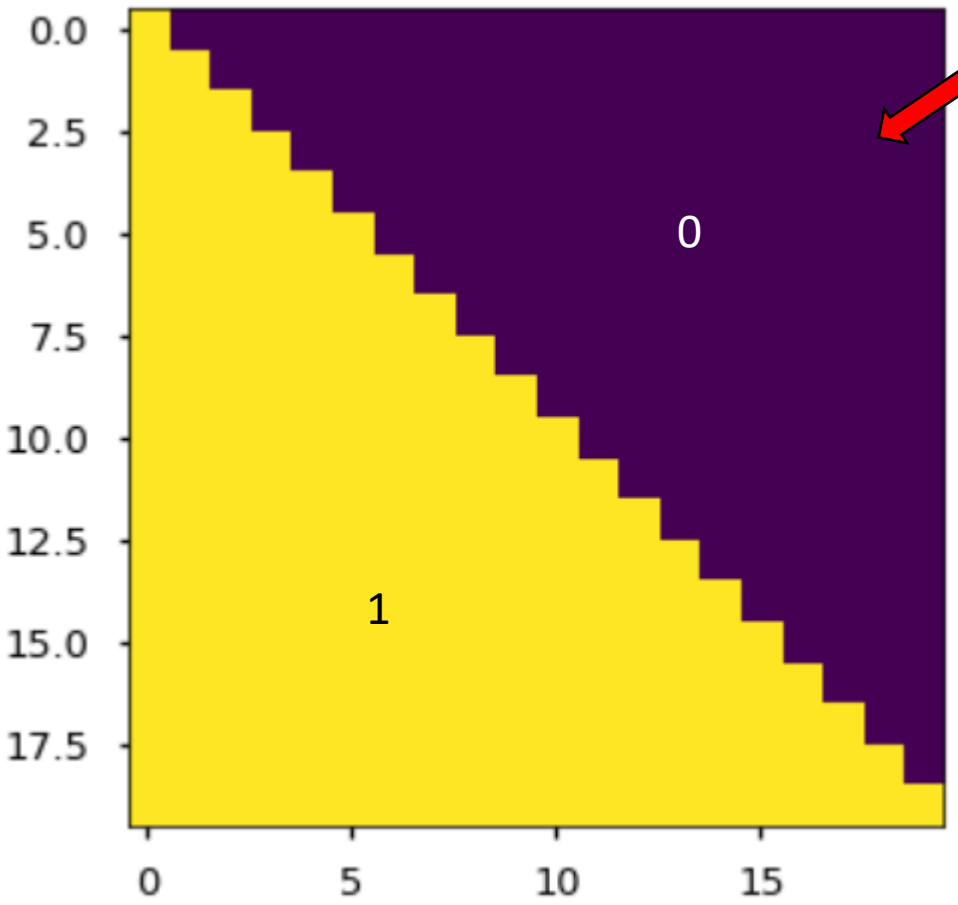
Работа энкодера



Работа декодера

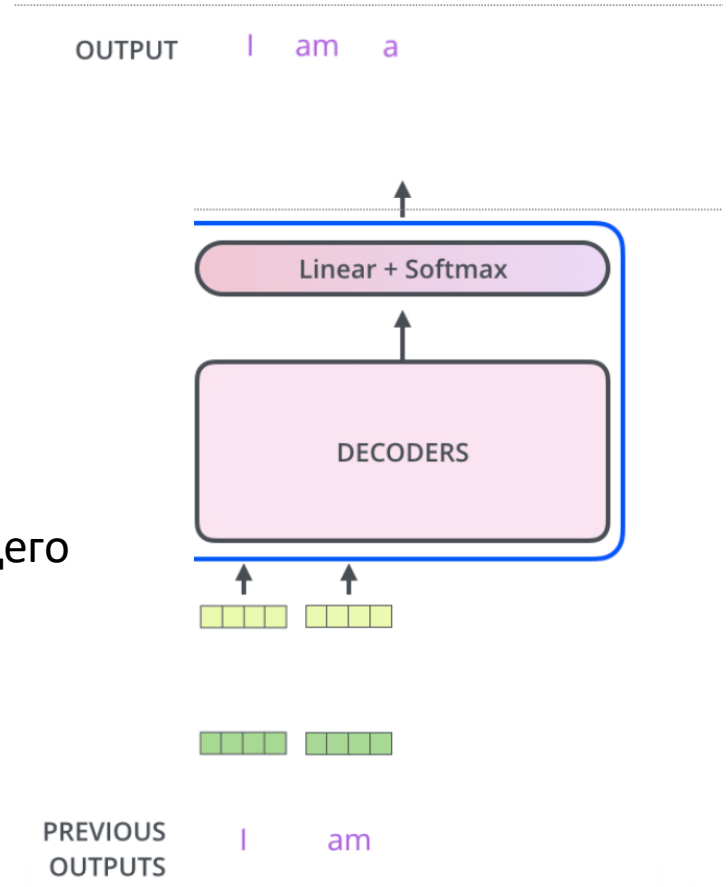


Обучение декодера

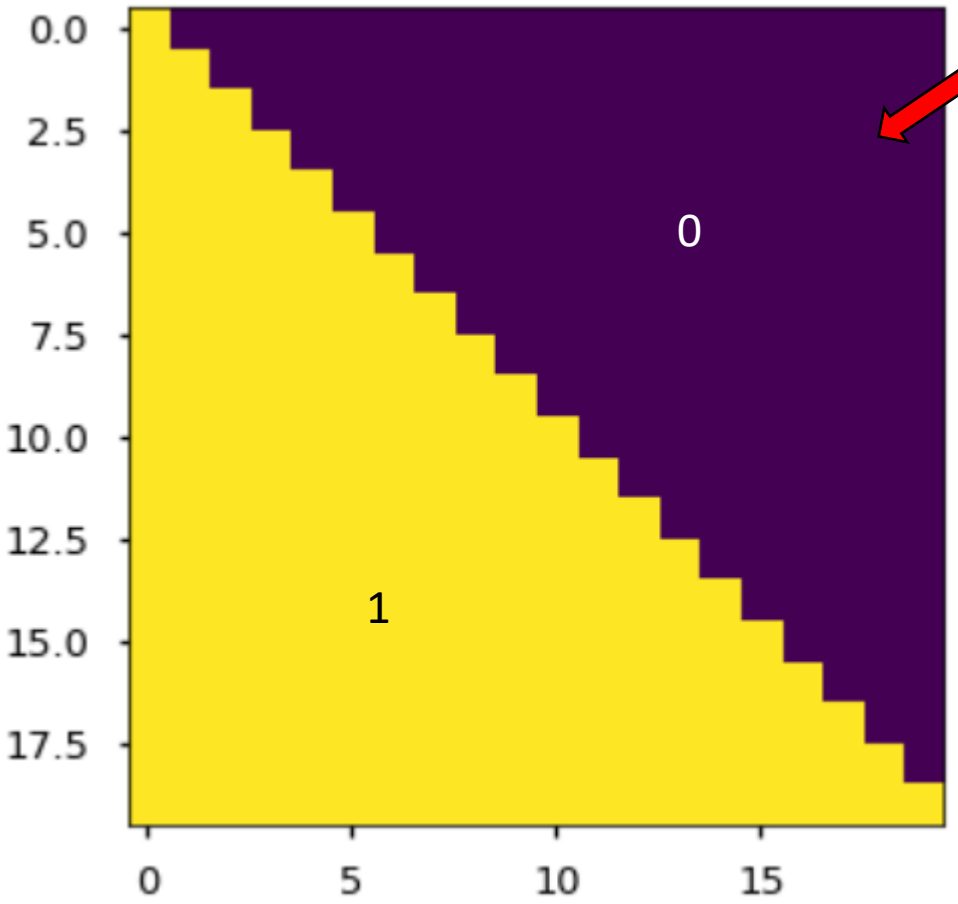


Zero attention

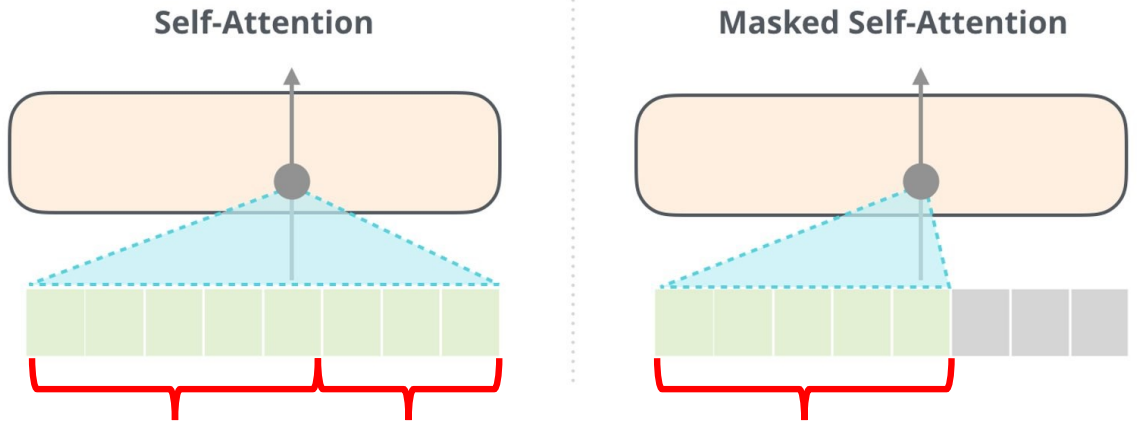
Проблема:
В декодере не можем использовать информацию из будущего



Обучение декодера



Zero attention



all tokens (left + right)

up to and including

Masked Scores
(before softmax)

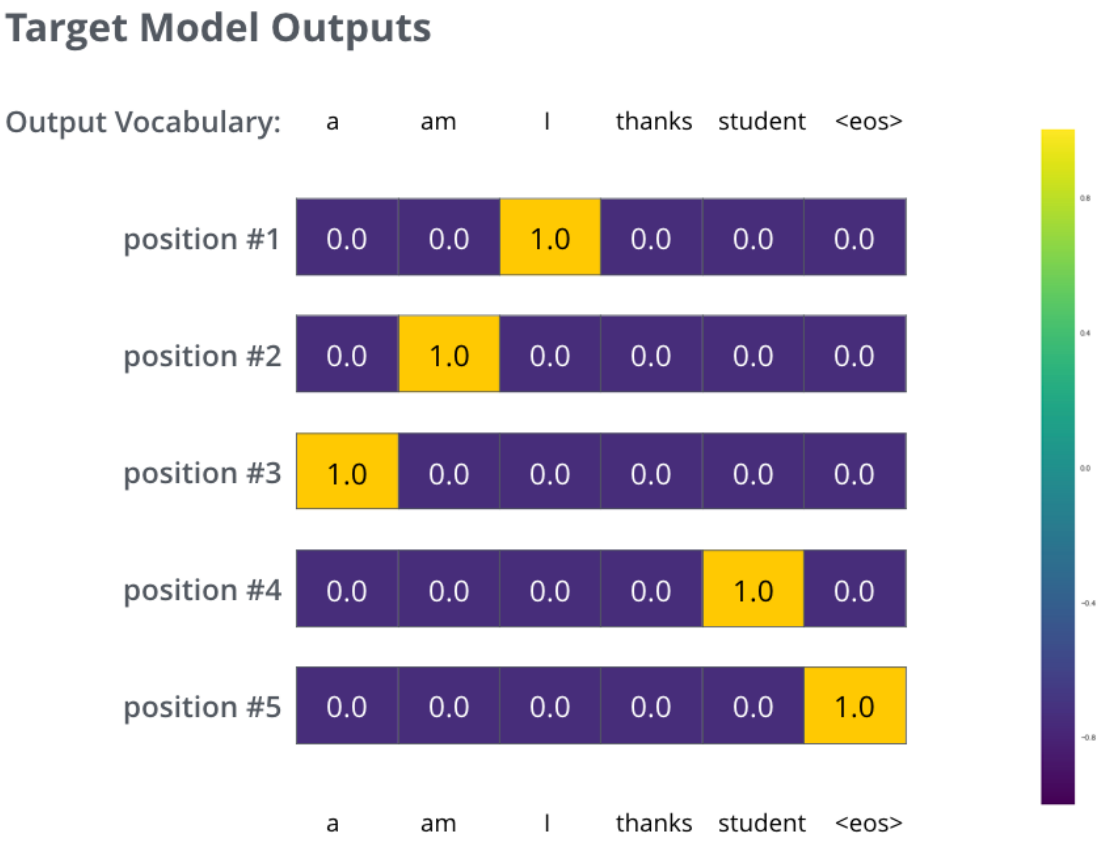
0.11	-inf	-inf	-inf
0.19	0.50	-inf	-inf
0.53	0.98	0.95	-inf
0.81	0.86	0.38	0.90

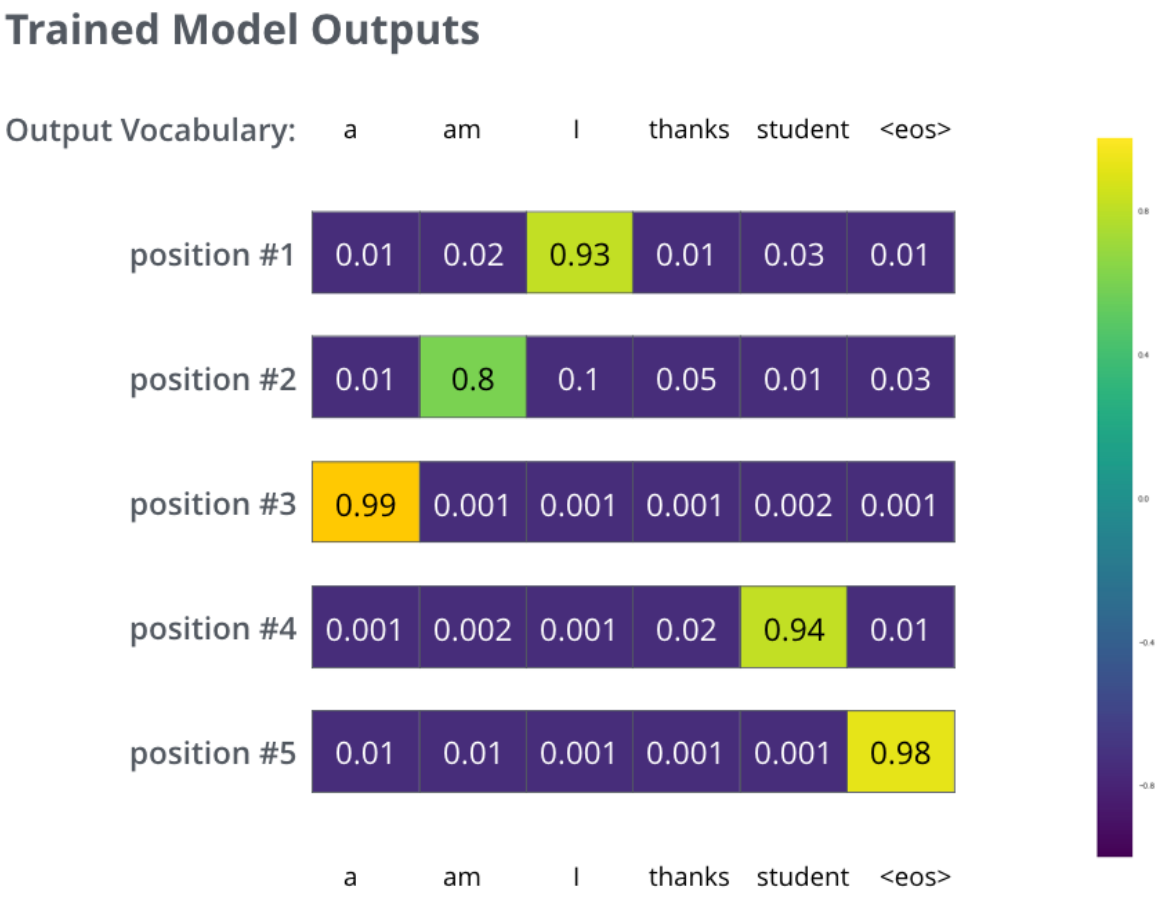
Softmax
(along rows)

Scores

1	0	0	0
0.48	0.52	0	0
0.31	0.35	0.34	0
0.25	0.26	0.23	0.26

Обучение трансформера

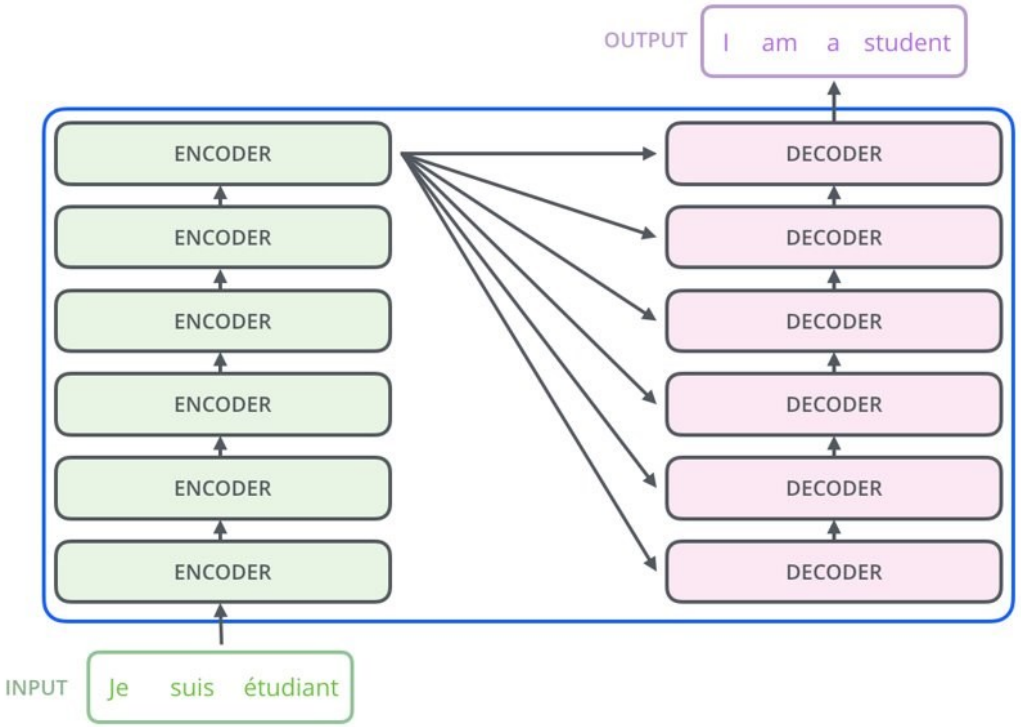




Таргет – слова и их позиции в предложении. При обучении считаем ошибку между ответом модели и таргетом. Далее эту ошибку распространяем по всей модели от декодера (целевой язык) до энкодера (исходный язык)

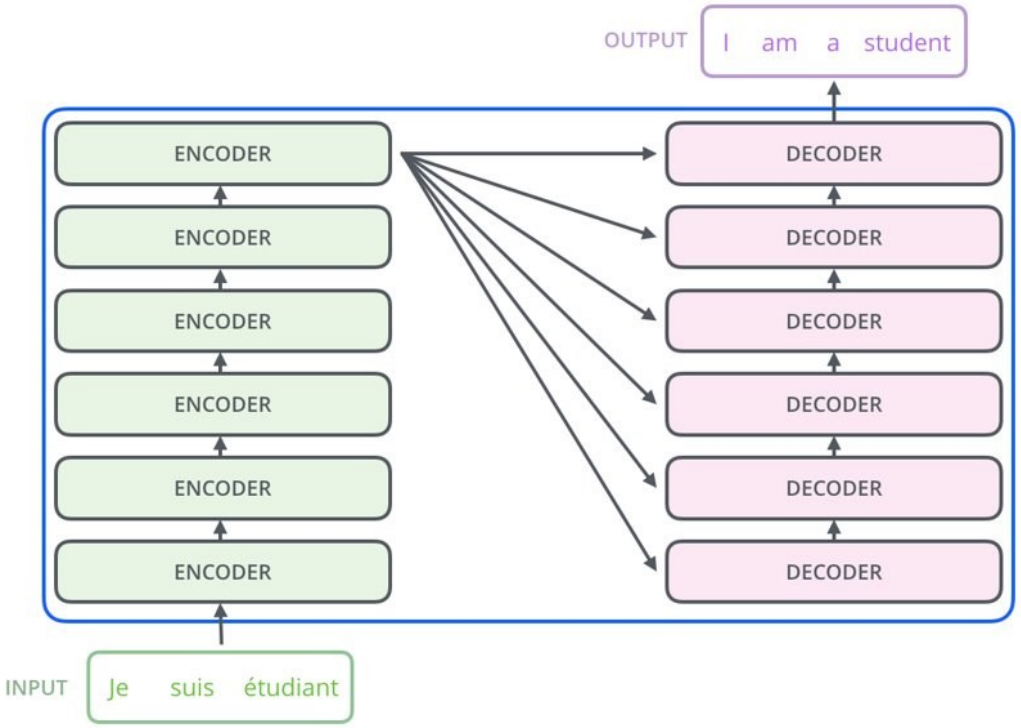
BERT vs GPT

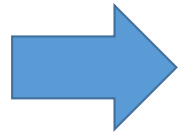
Трансформер – разработан для задачи перевода



BERT vs GPT

Трансформер – разработан для задачи перевода

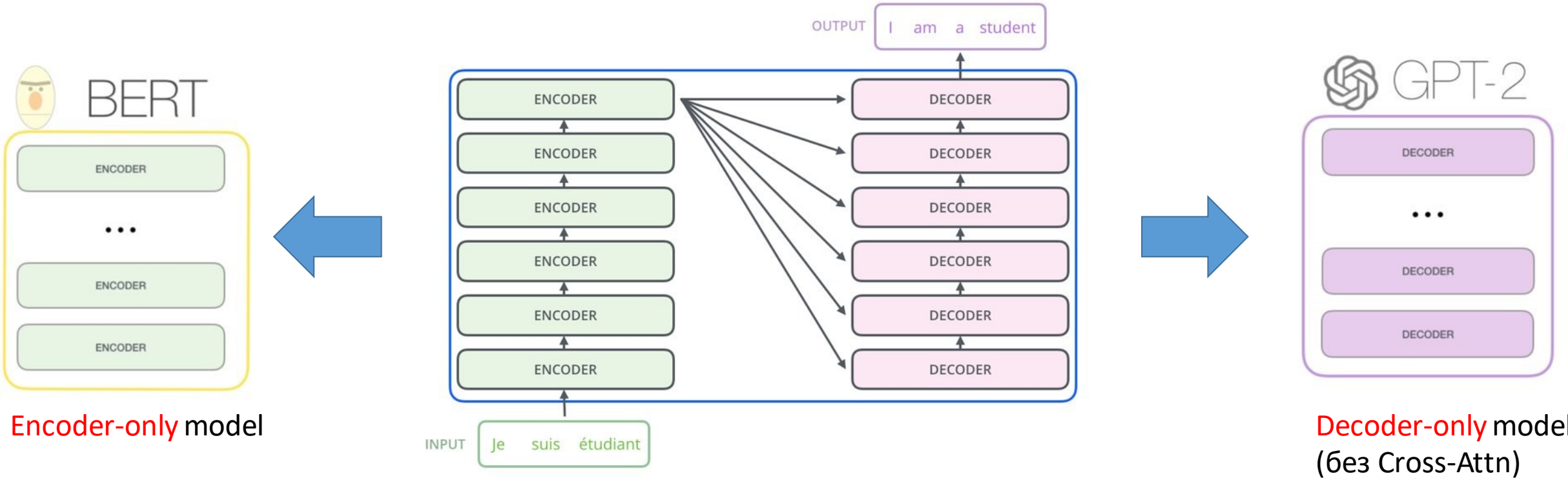




Decoder-only model
(без Cross-Attn)

BERT vs GPT

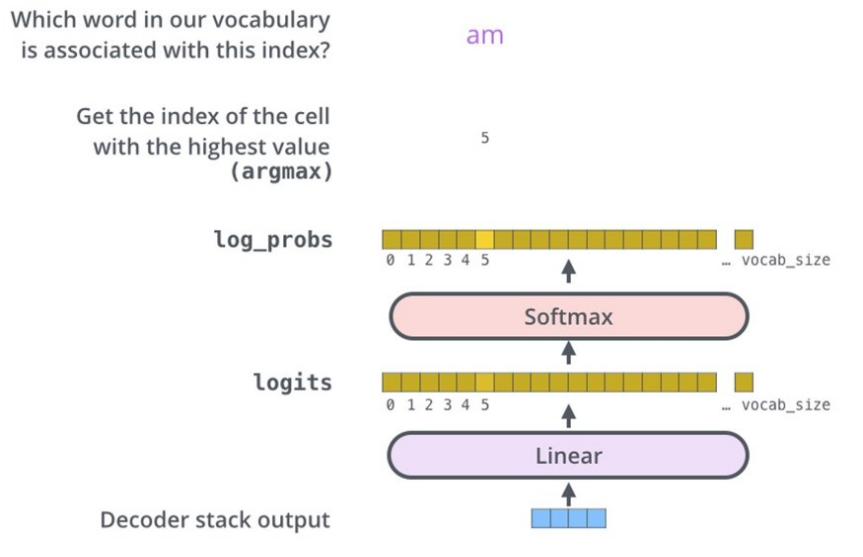
Трансформер – разработан для задачи перевода



Encoder-only model

Decoder-only model
(без Cross-Attn)

Обучение BERT: Masked Language Modeling (MLM)



Выбираем случайным образом из предложения 15% токенов и заменяем их токеном [MASK].
Делаем предсказание для маскированного токена – какое слово было заменено на маску?

Sentence:

The doctor ran to the emergency room to see [MASK] patient.

Mask 1 Predictions:

38.3% **his**
36.9% **the**
8.1% **another**
7.3% **a**
6.0% **her**

Обучение BERT: Next Sentence Prediction (NSP)

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

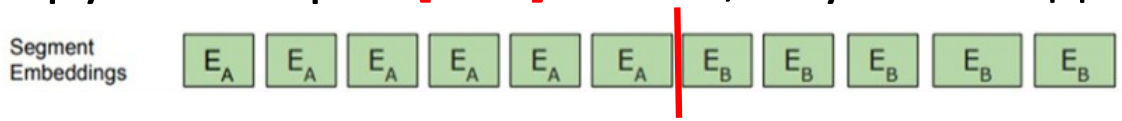
Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

Обучаемся на контекст – где должны быть высокие веса

- NSP – Next Sentence Prediction (Обычный бинарный классификатор)
- Два предложения конкатенируются через [SEP]-токен; обучаются дополнительные Segement Embeddings



Обучение BERT: Next Sentence Prediction (NSP)

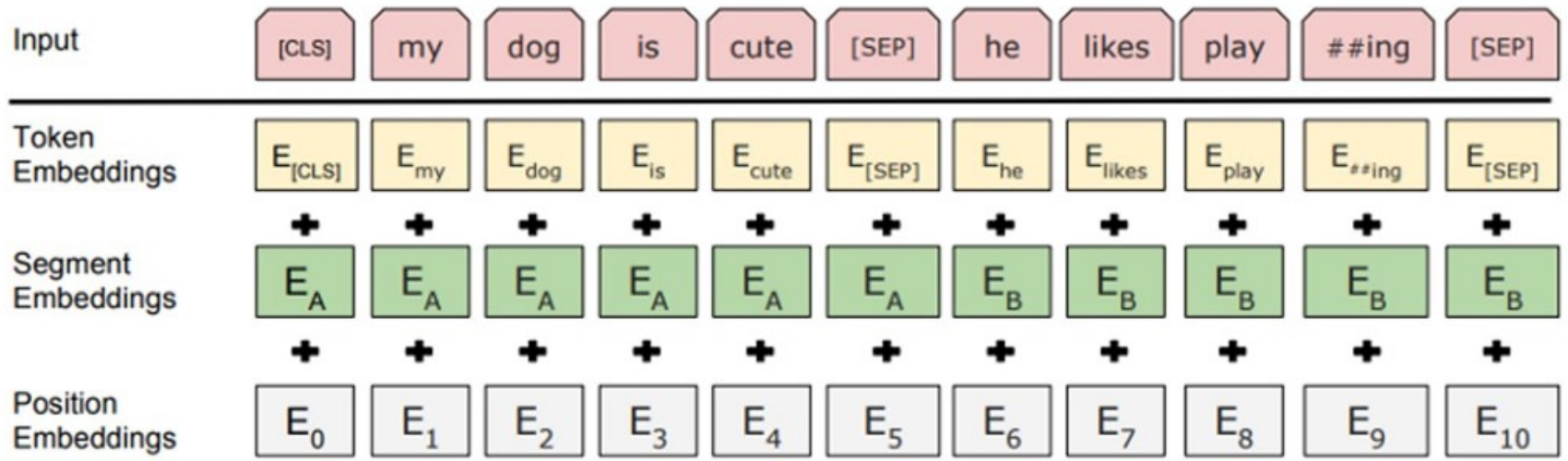


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Segement Embeddings могут кодировать в себе мета-информацию, например Geo, web-домены, итд

Почему трансформеры стали популярными?

- One-size-fits-all – разрабатывался для машинного перевода, однако показывает хорошие результаты в любых областях, где используются обычные модели
  - Можно использовать в качестве baseline-моделей в качестве старта
  - Возможность Transfer Learning
  - Легкое масштабирование – модульная архитектура - «Трансформер».
    - GPT3 ~ GPT2 * 120 раз

- Больше данных = лучше модель
- Быстро обучаются, т.к. нет рекурсии => можем работать параллельно
- Software/Hardware оптимизация (GPU, TPU, одна архитектура для разных задач)

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	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$