НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

Компьютерная лингвистика и информационные технологии

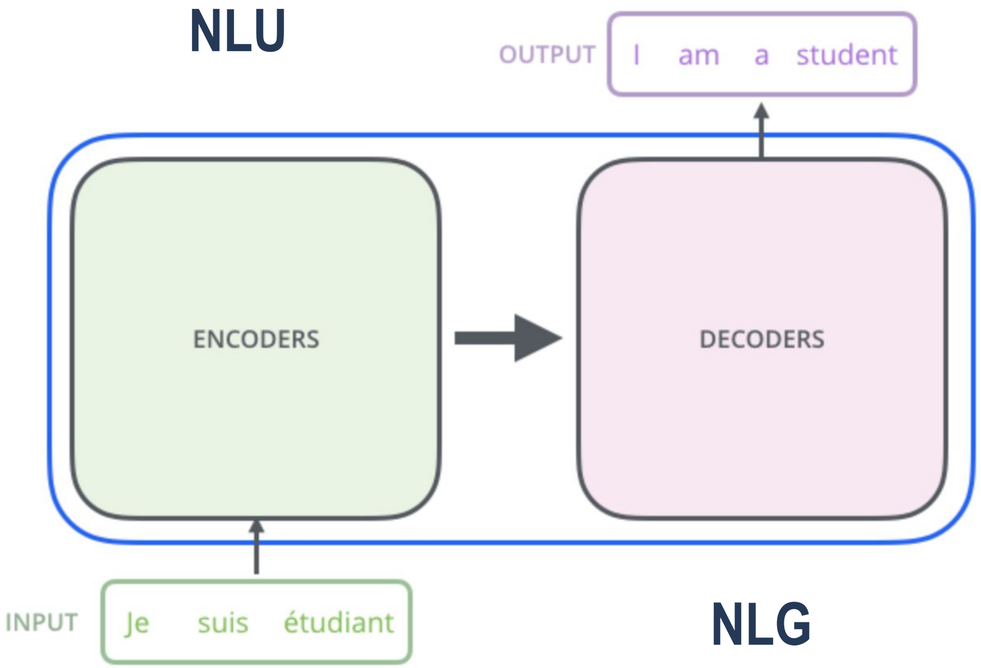
BERT, трансформеры
(на основе материалов Jay Alammar и Е. Артемовой)

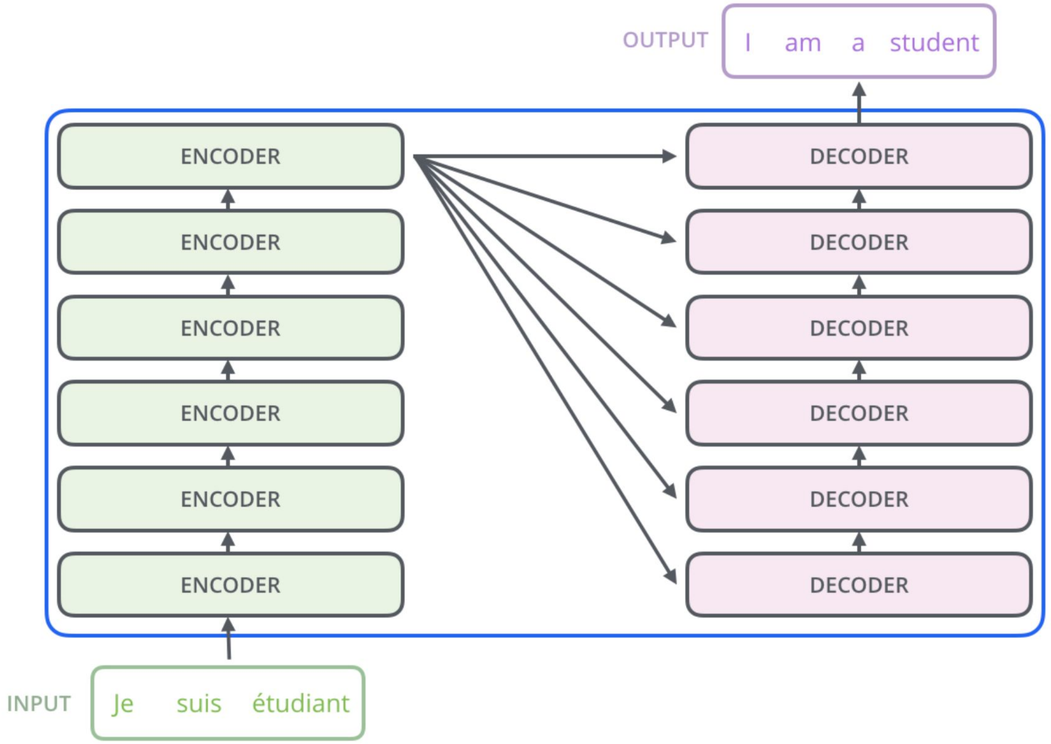
BERT

- BERT (Devlin et al., 2019);
- На основе архитектуры трансформер (Vaswani et al., 2017) для NMT;
- Новая парадигма: **обучение** большой языковой модели с использованием **training objectives** -> **дообучение** на целевой задаче;
- Прорыв в области АОЕЯ
- http://jalammar.github.io/illustrated-transformer/

Трансформер

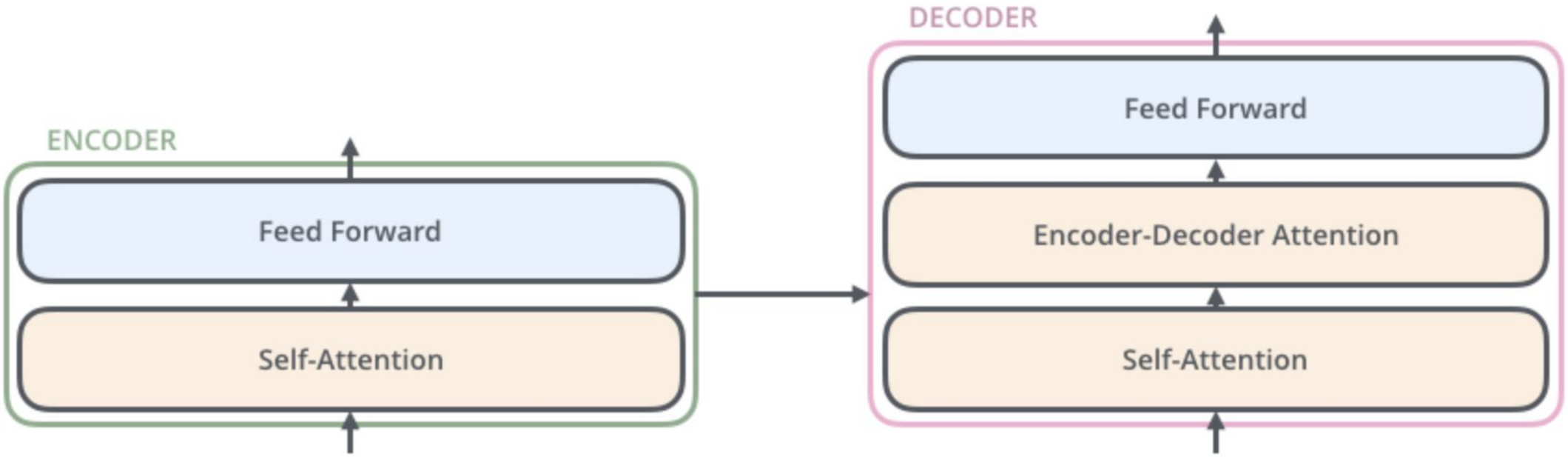
Энкодер-декодер на основе трансформера





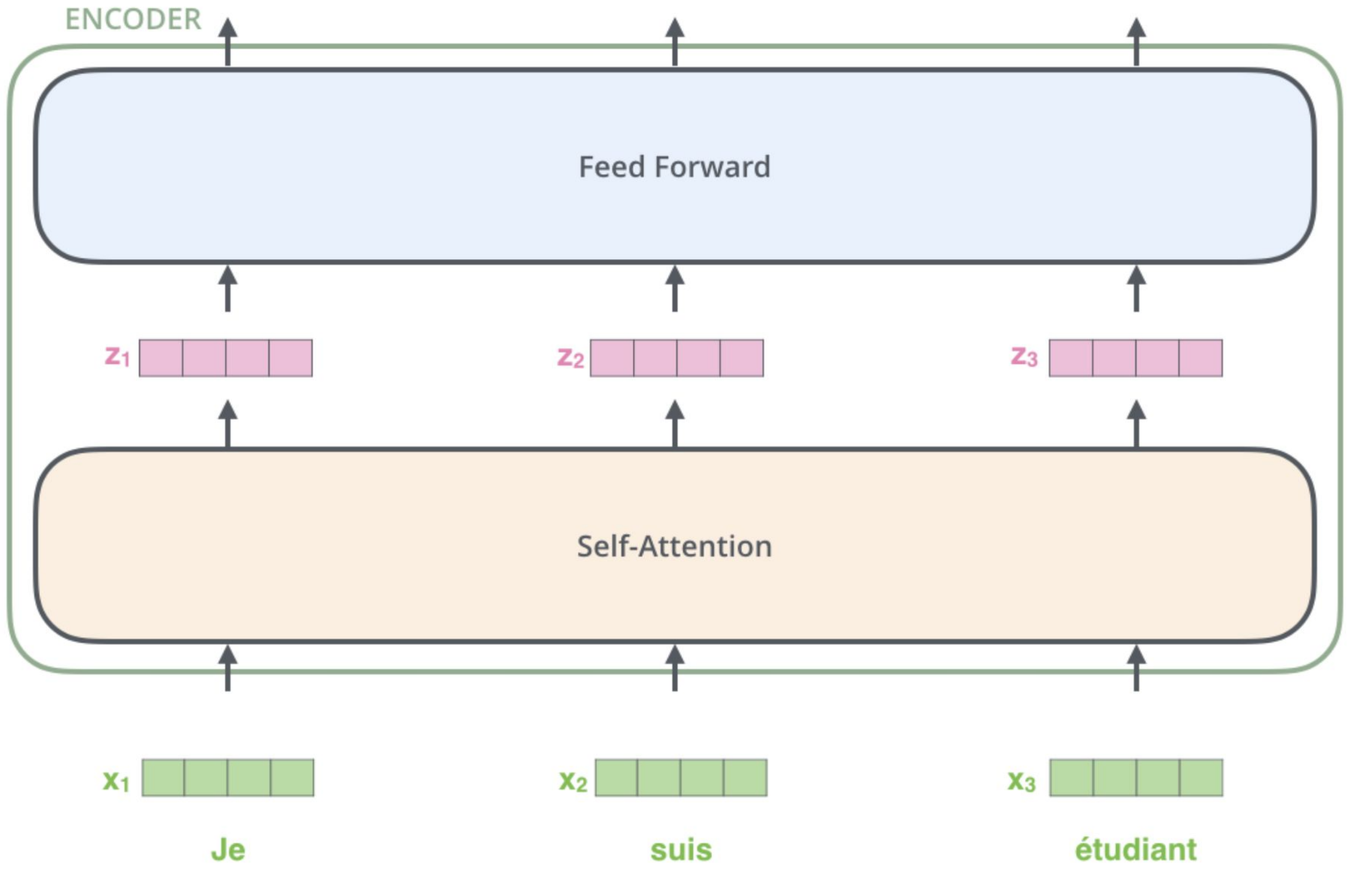
The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:

Новый механизм внимания



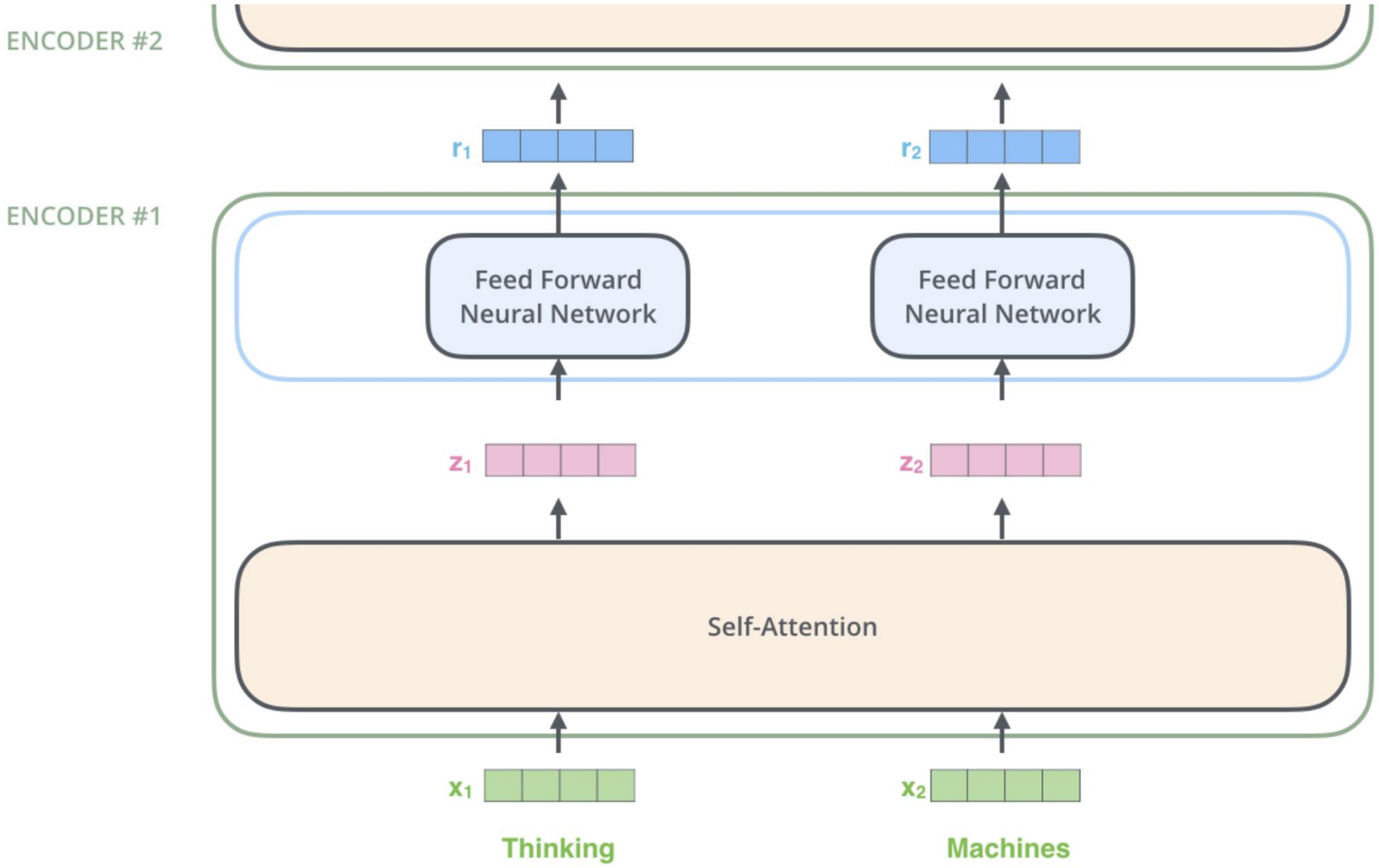
Блок энкодера

- Для простоты – на вход получаем предобученные пословные эмбеддинги
- Хотим, чтобы эмбеддинги z_i хорошо кодировали контекст



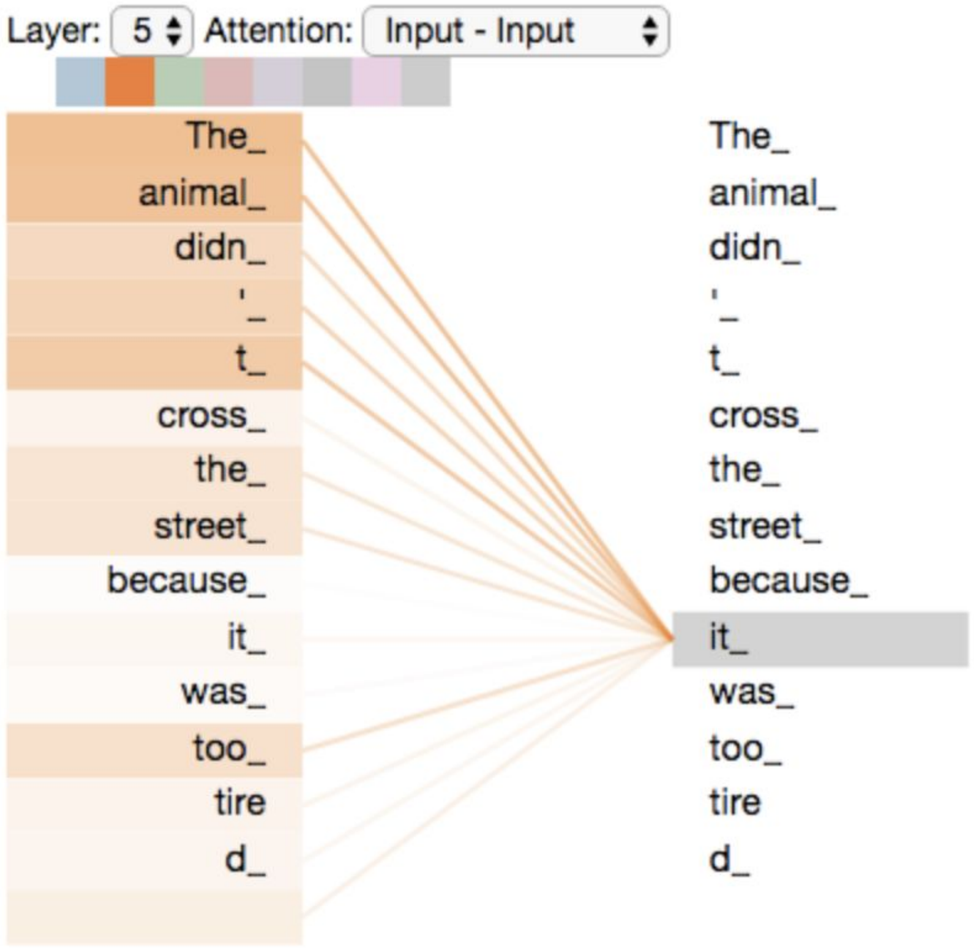
Блок энкодера

- К каждому входному элементу после self-attention независимо применяется FFN
- Каждый блок применяется последовательно



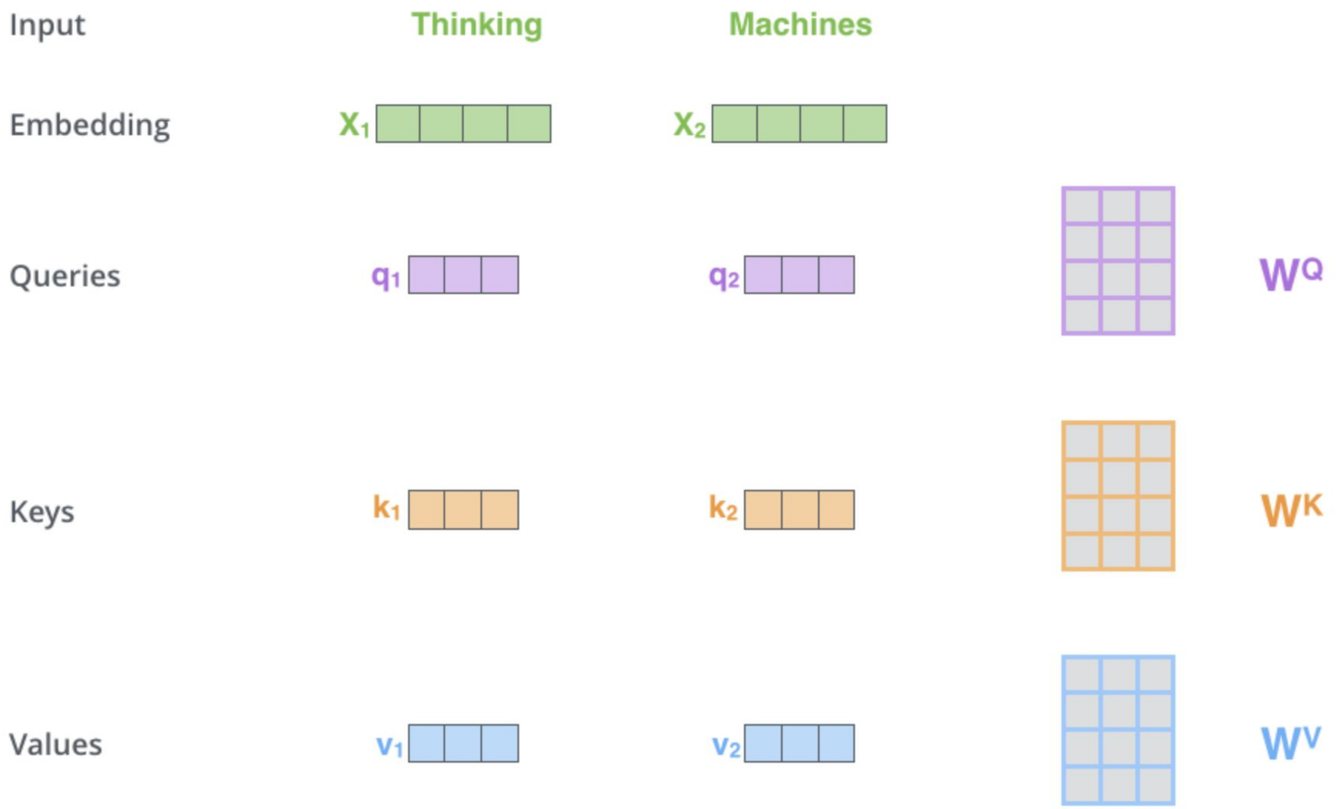
(Self-)Attention

- Для каждого элемента последовательности можем определить “значимость” других элементов в контексте
- Attention как метод интерпретации
- Хотим распространить “значимость” элементов последовательности вверх по сети



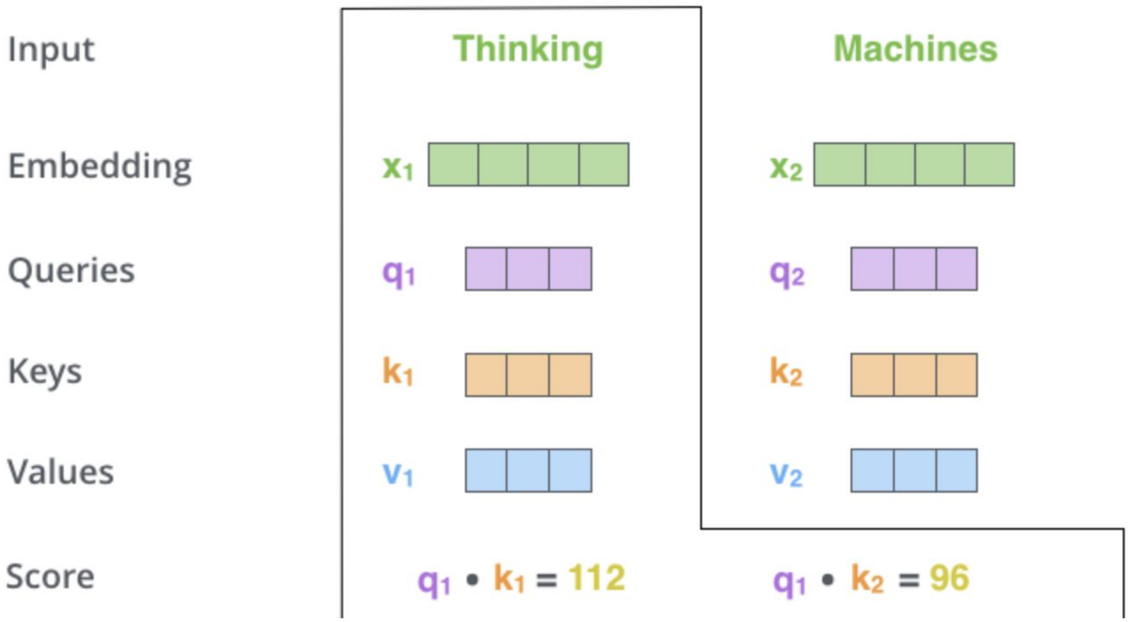
Self-Attention

- Для каждого входного вектора хотим получить ключ, запрос и значение (key, value and query)
- Имеем три обучаемые матрицы: W^Q, W^K, W^V
- Умножаем входные векторы на эти матрицы, получаем векторы q, k и v (линейное преобразование)



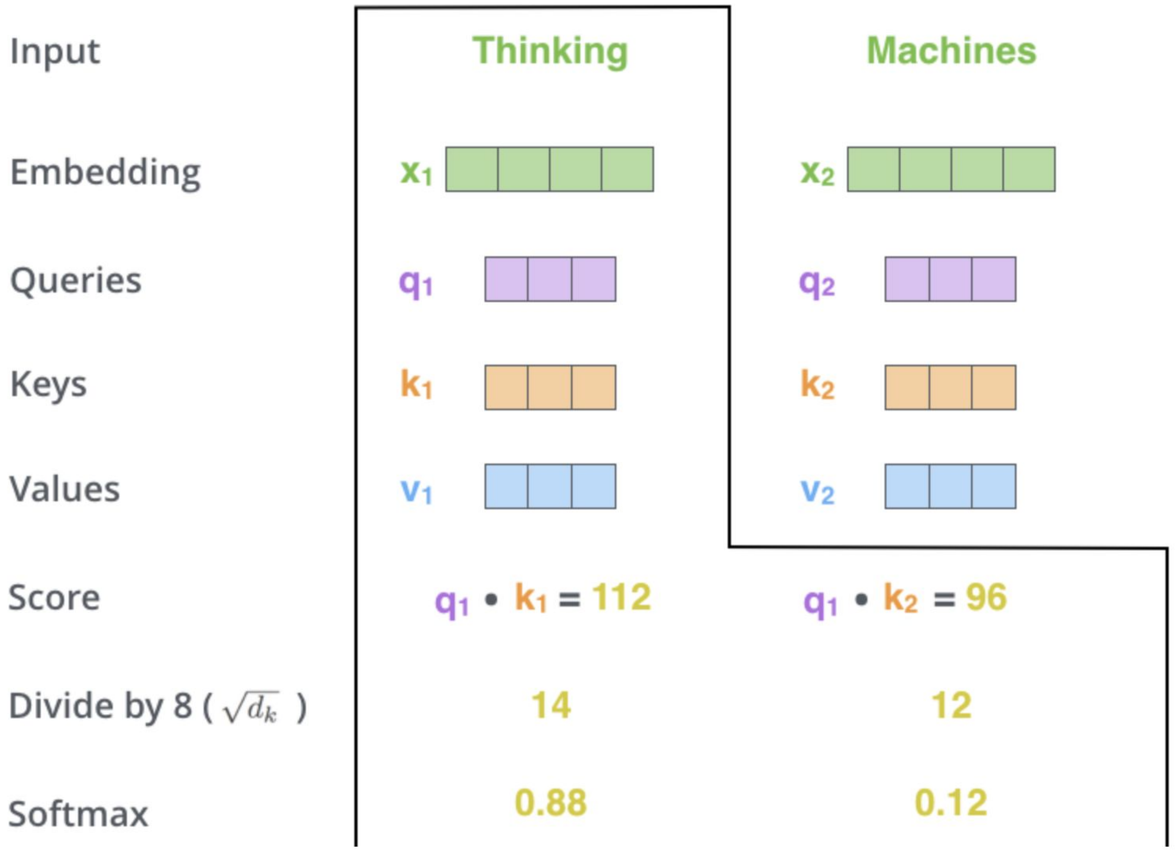
Self-Attention

- Воспринимаем элемент последовательности как запрос (q)
- Перебираем все комбинации q_i * k_j и считаем промежуточные скоры, включая сам элемент



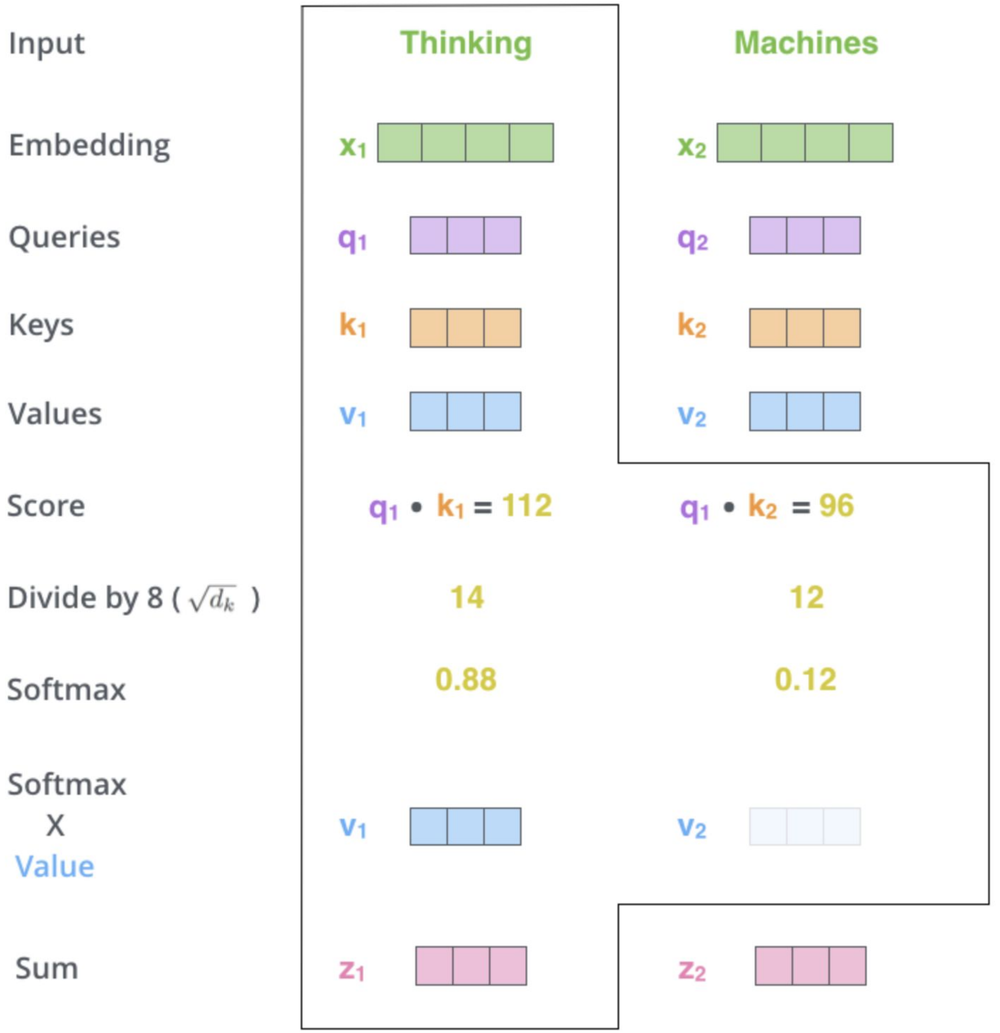
Self-Attention

- Скоры (результаты скалярных произведений на шаге 2) делим на корень из d_k
- d_k – гиперпараметр (возьмем за данность)
- Берем софтмакс по вектору скоров, чтобы получить веса внимания

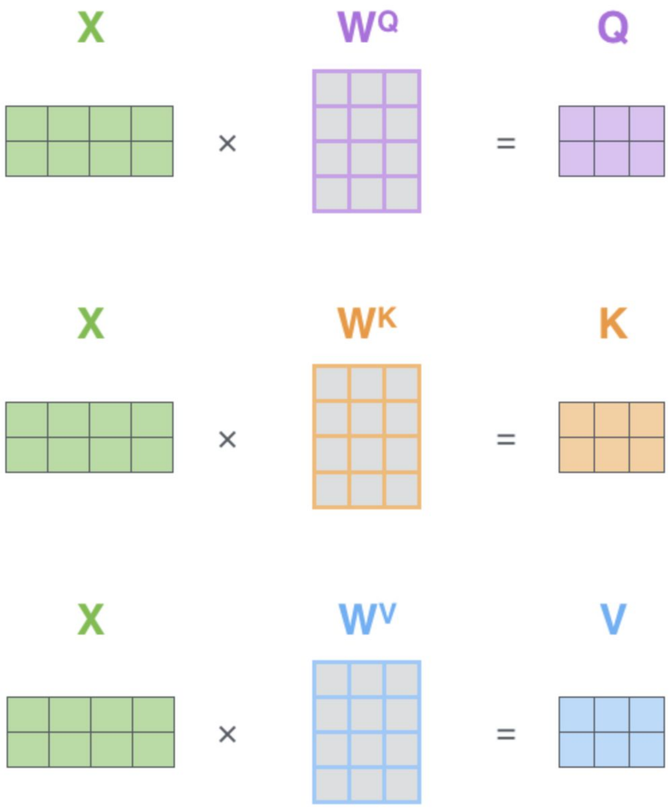


Self-Attention

- Умножаем все векторы v_i на полученные веса внимания
- Все полученные значения складываем



Self-Attention

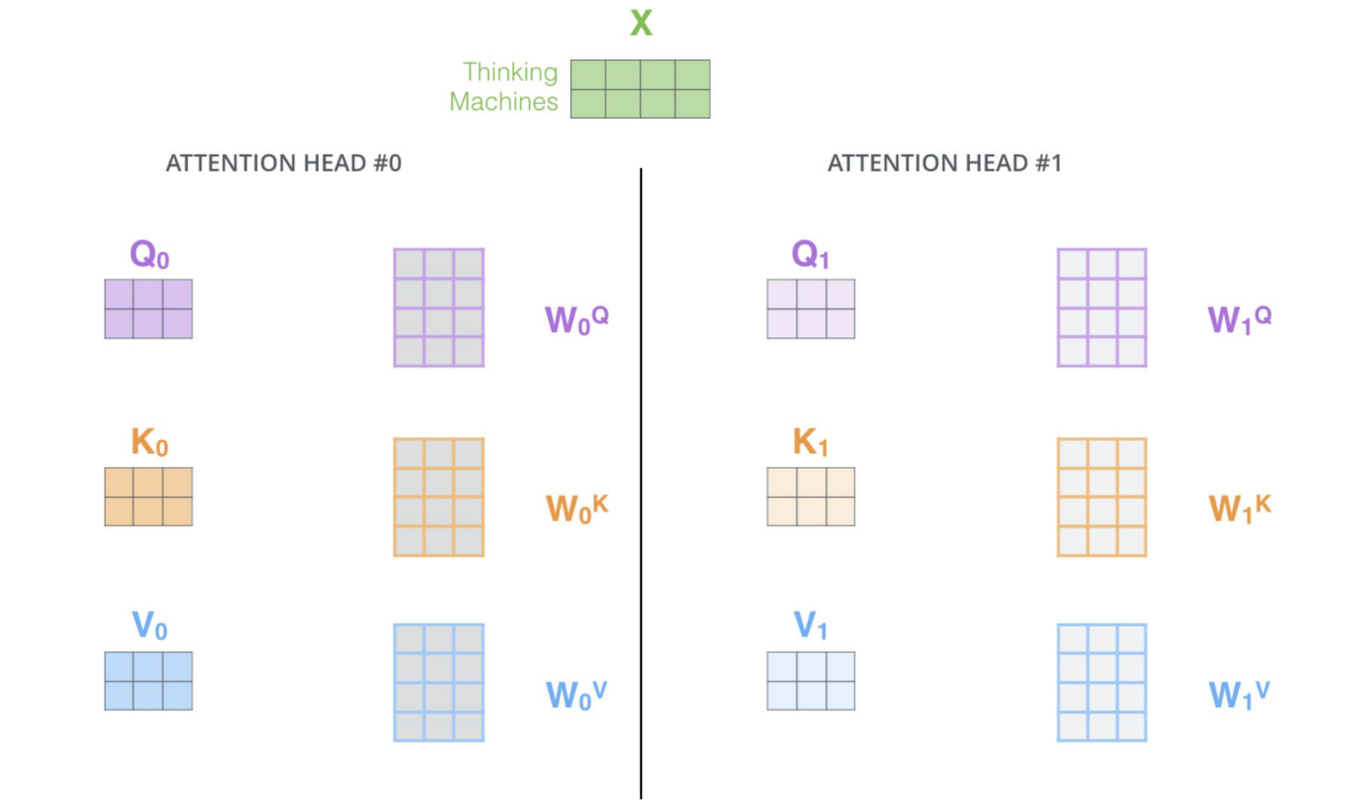


$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

The self-attention calculation in matrix form

Змей Горыныч

- Имеем k голов, для каждой i-ой головы получаем вектор z_i
- Голова – отдельный механизм self-attention (набор из 3-х матриц)
- Считаем self-attention для каждой головы независимо
- Ширина и глубина модели



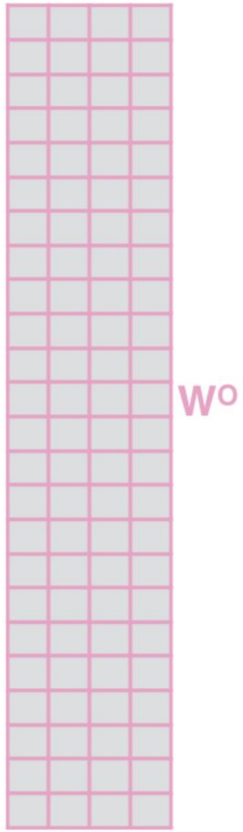
With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the WQ/WK/WV matrices to produce Q/K/V matrices.

Змей Горыныч

1) Concatenate all the attention heads

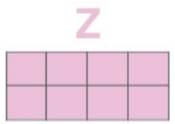
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



Змей Горыныч

1) This is our input sentence*

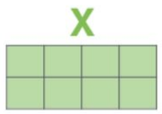
2) We embed each word*

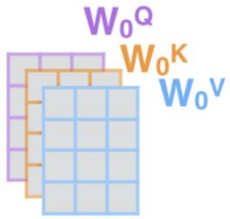
3) Split into 8 heads. We multiply X or R with weight matrices

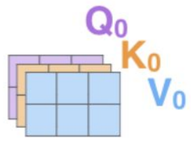
4) Calculate attention using the resulting Q/K/V matrices

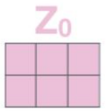
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

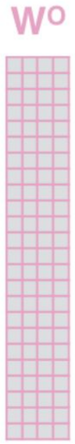
Thinking Machines

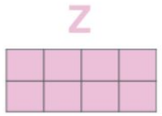








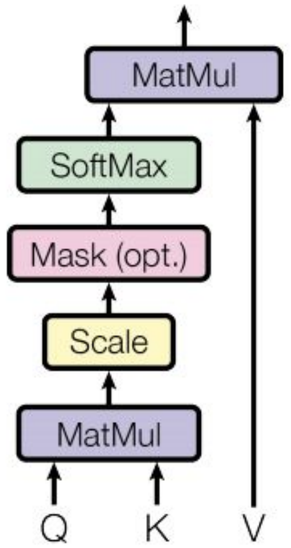




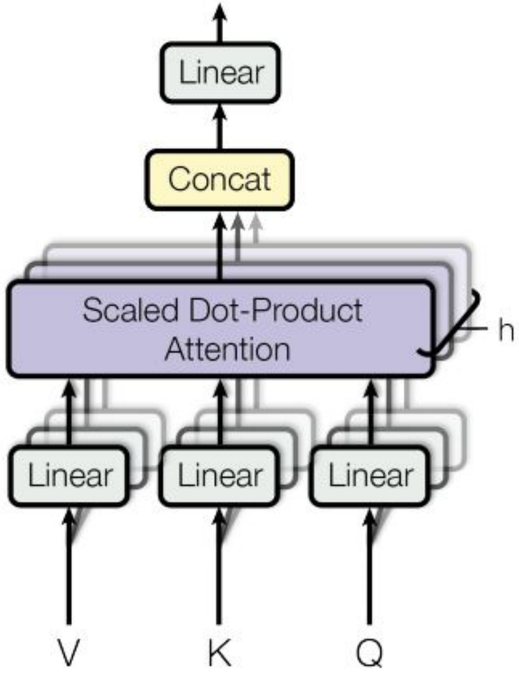
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Scaled Dot-Product Attn vs Multi-Head Attn

Scaled Dot-Product Attention



Multi-Head Attention

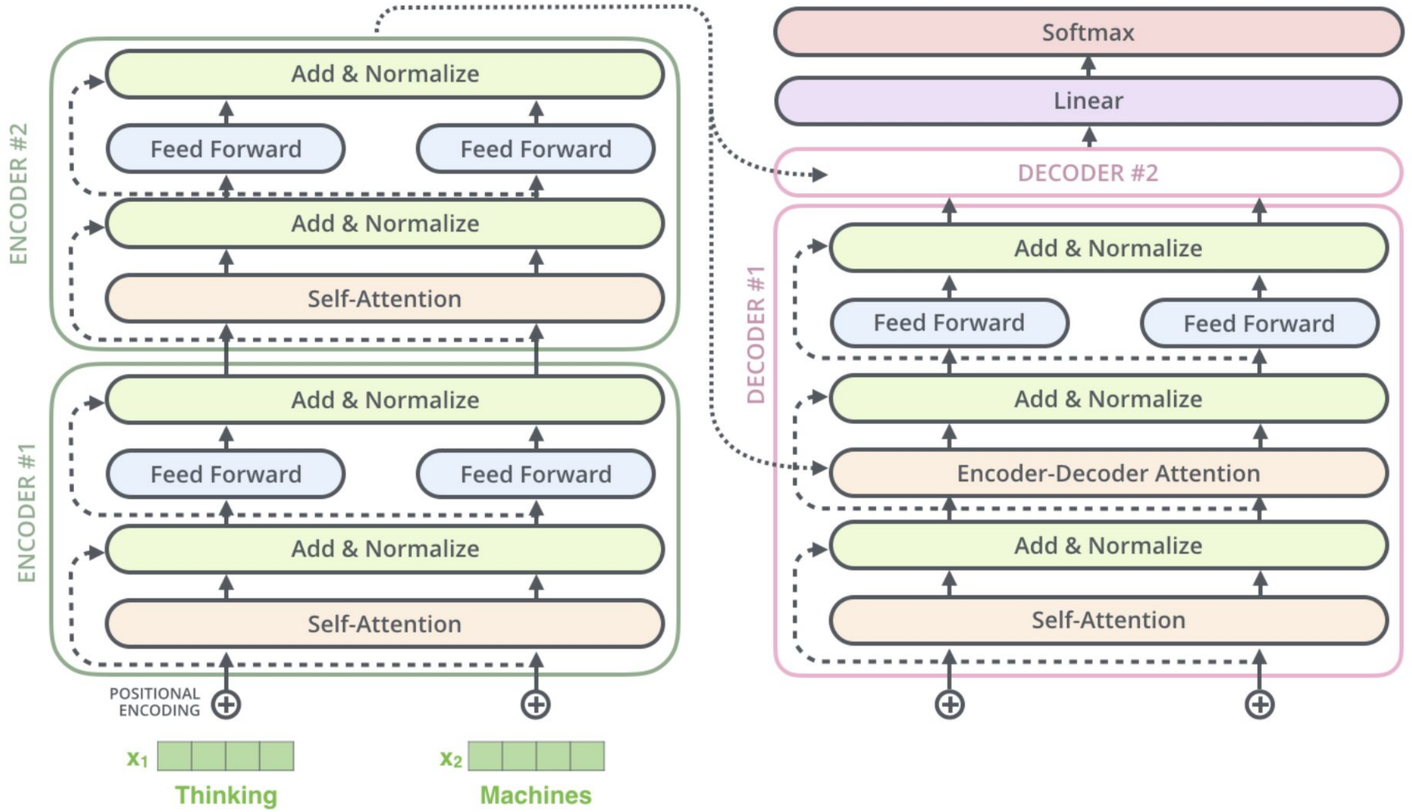


$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

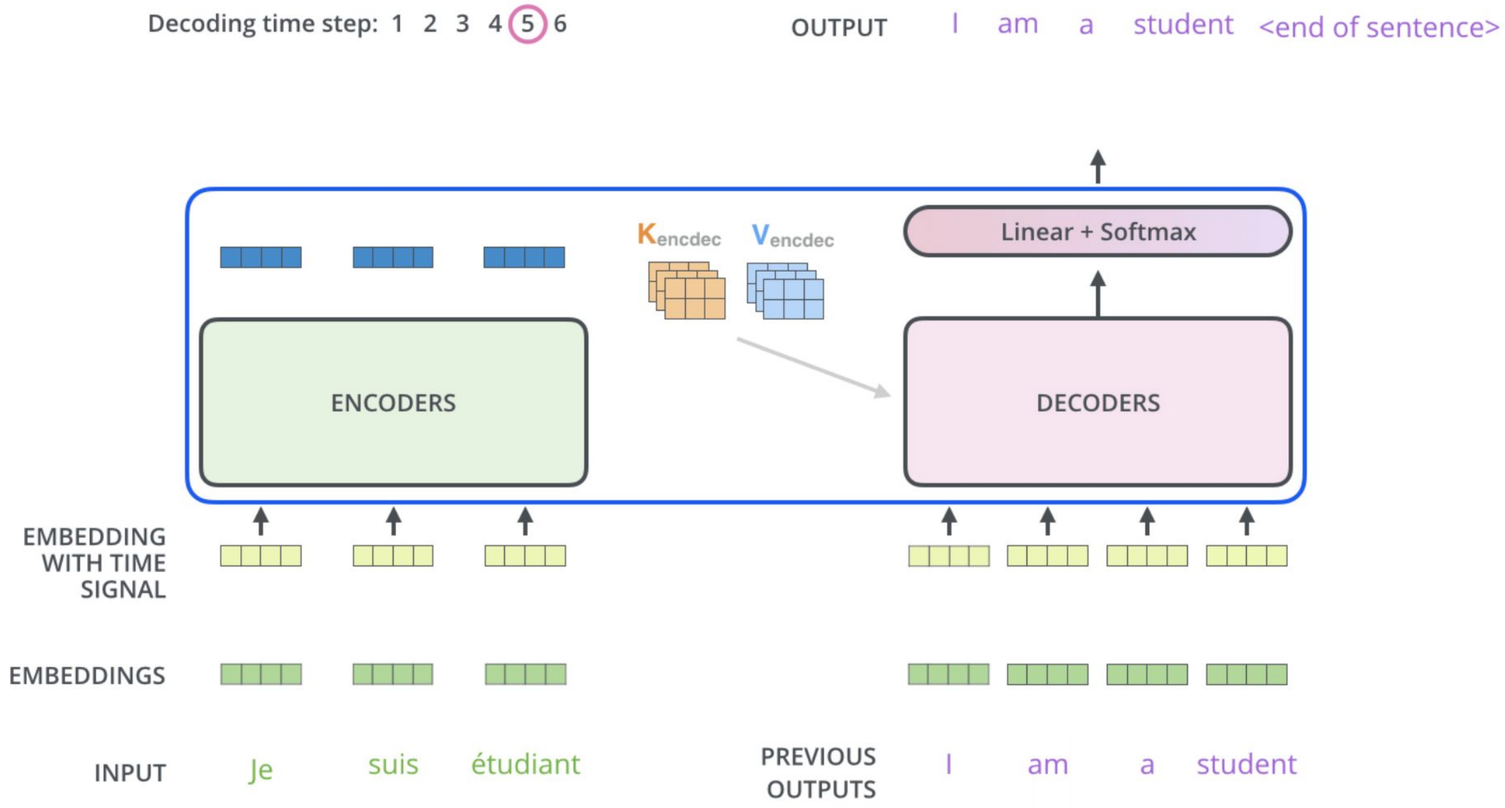
Декодер

- Encoder-Decoder Attention: декодер получает выход с последнего блока энкодера
- Encoder: k, v
- Decoder: q
- Хотим сформировать вектор контекста (по аналогии с RNN)



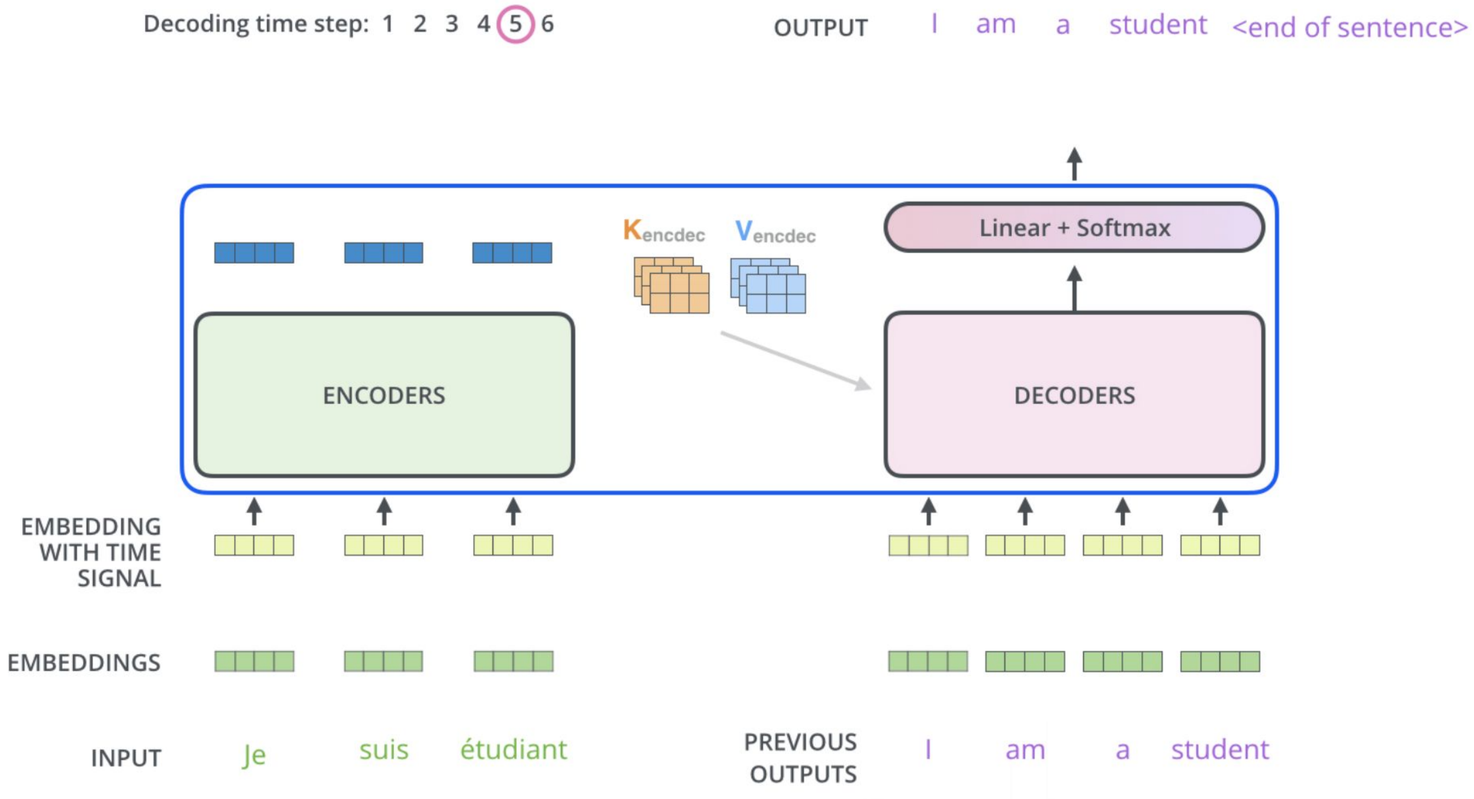
Декодер

- Верим, что выход 6-ого блока энкодера есть хорошая репрезентация контекста
- Матрицы Kencdec и Vencdec обучаются, чтобы получить векторы k и v для декодера (общие для каждого блока декодера)



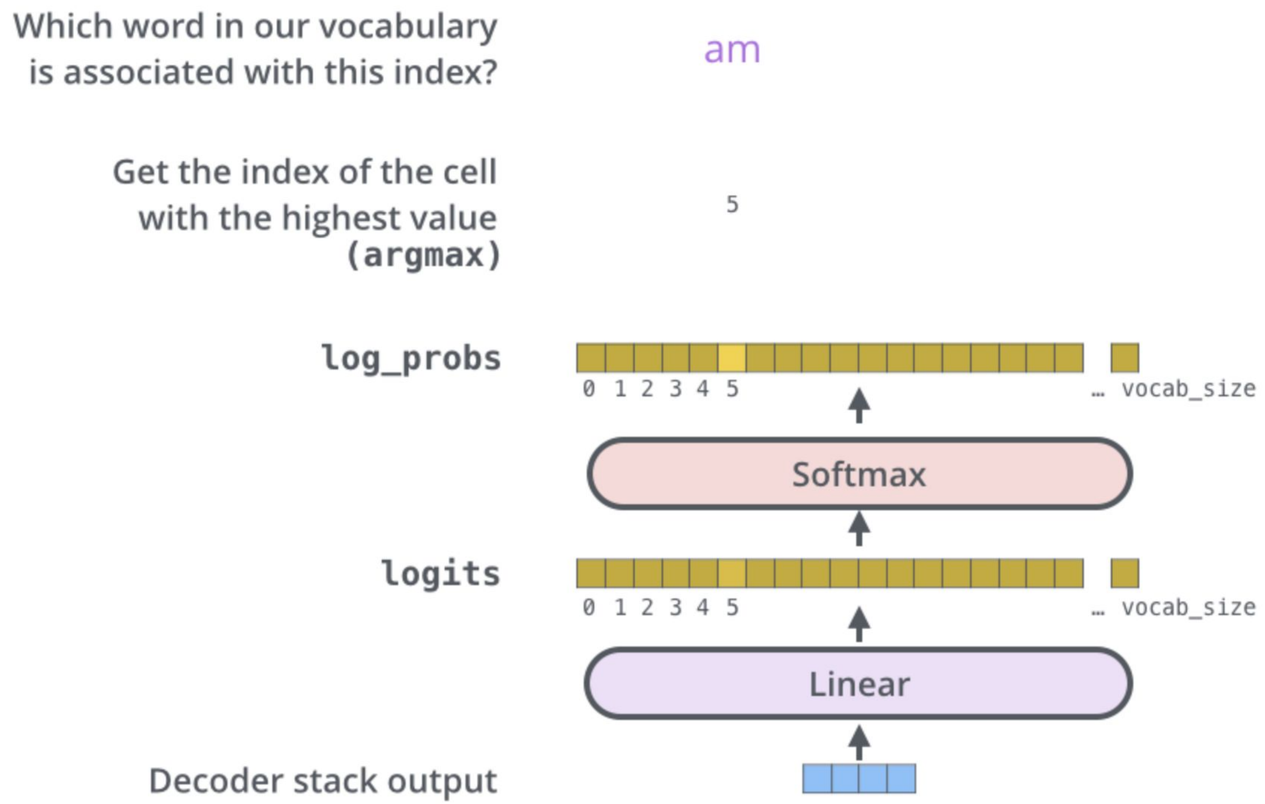
Декодер

- Декодер генерирует элементы последовательности друг за другом (как ЯМ)
- Энкодер может видеть все элементы последовательности, в то время как декодер в момент времени видит подмножество элементов (маска)



Декодер

- Для каждого предсказываемого элемента последовательности после линейного преобразования получаем набор чиселок – logits (по самому правому элементу в момент времени)
- Берем софтмакс



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

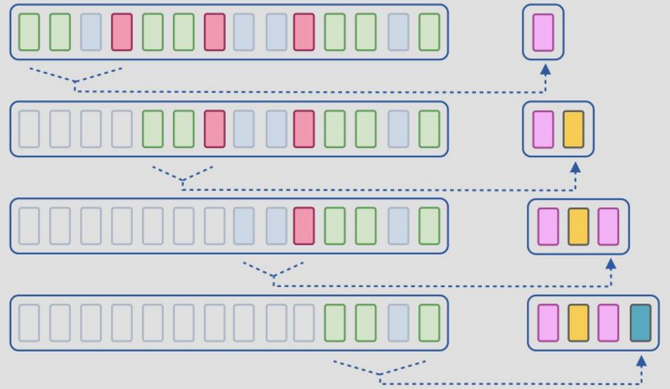
Byte Pair Encoding

Subword Tokenization

- Чего мы хотим от токенизации?
- Представим слова как набор “подслов” (са
  токены)
- В чем отличие от fastText?
- Алгоритм сжатия данных

Попарное битовое кодирование (Byte pair encoding, BPE)

- Считаем частоты пар символов
- Склеиваем самую частую пару символов и превращаем ее в новый символ
- Продолжаем повторять операцию фиксированное число раз



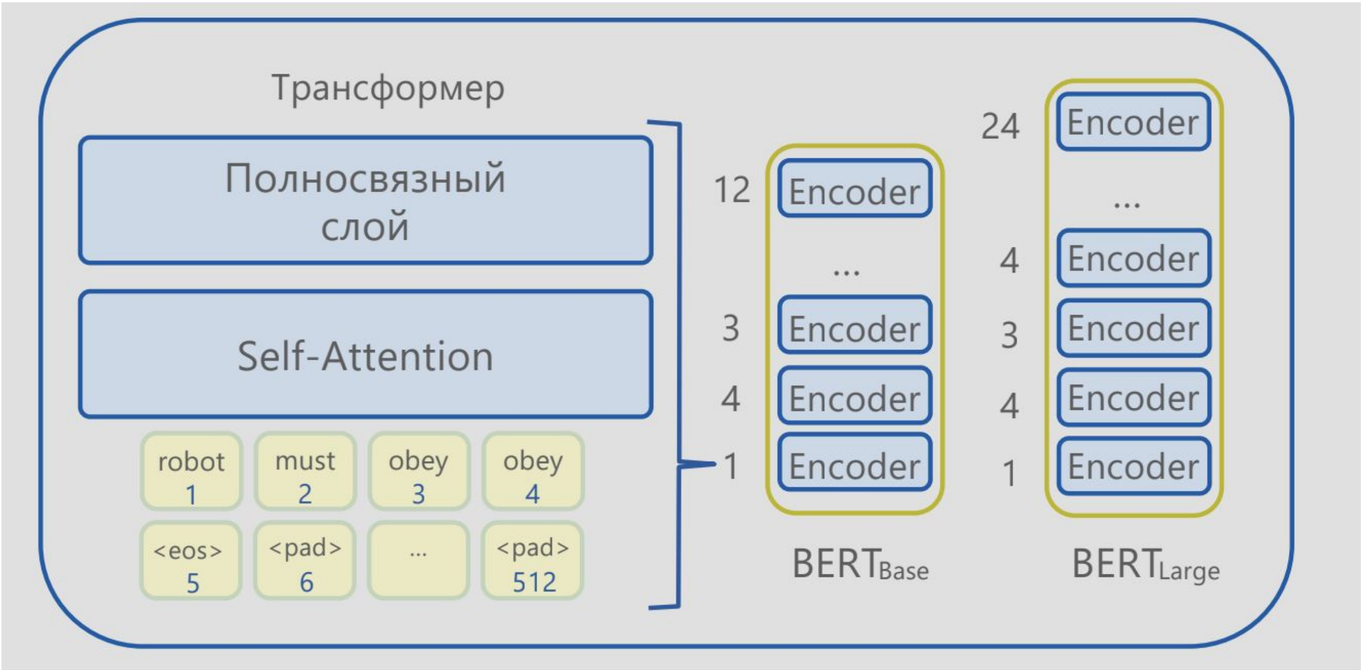
```
text = 'на дворе трава на дворе дрова'

group_subtokens(text)
```

```
['[CLS]', 'на', 'дворе', 'т', '##рава', 'на', 'дворе', 'др', '##ова', '[SEP]']
```

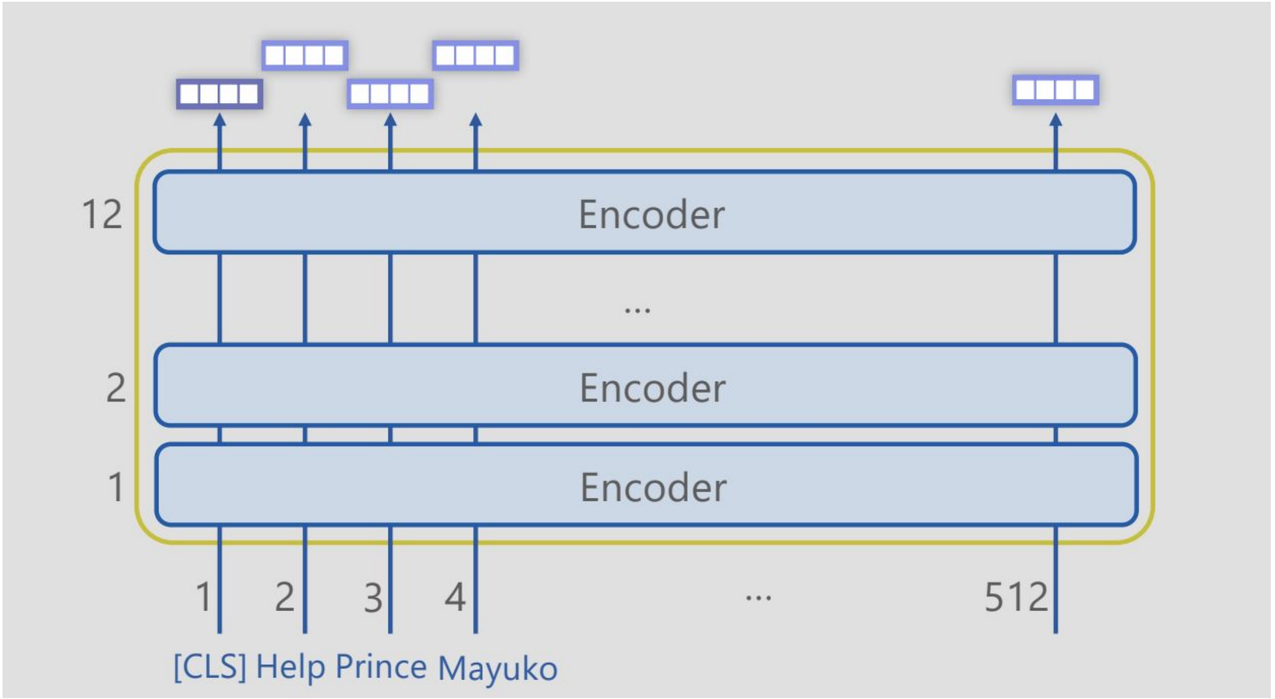
BERT

- Transformer-based Encoder
- Несколько конфигураций (12, 24)
- Управляющие токены [CLS], [MASK], [SEP] и саб-токены
- Training Objectives:
  - Masked Language Modeling (MLM)
  - Next Sentence Prediction (NSP)



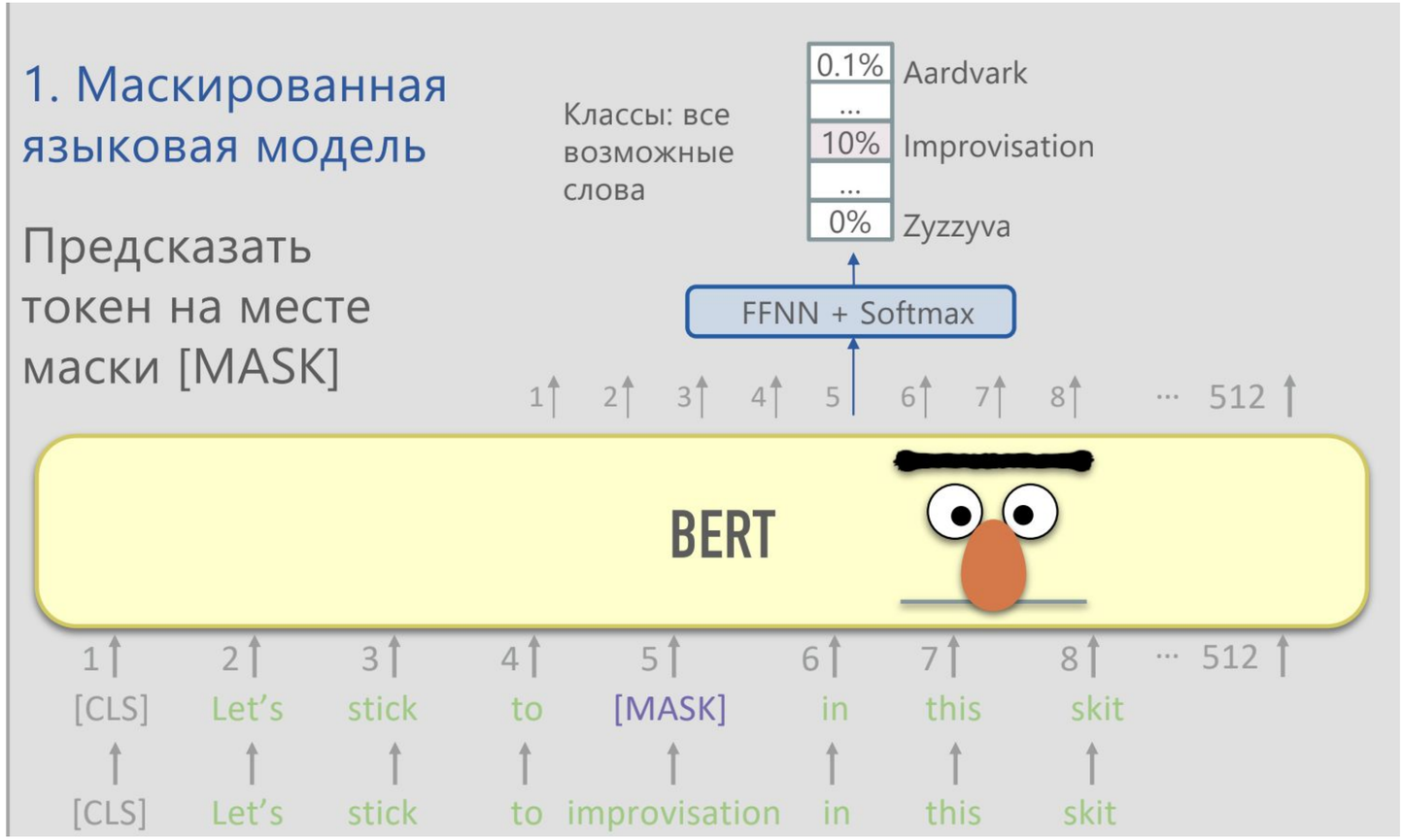
BERT

- Управляющие токены [CLS], [MASK], [SEP] и саб-токены
- Вектор токена [CLS] может использоваться как векторное представление предложения (CLS-Pooling)
- Для вектора слова можно взять эмбеддинг последнего саб-токена



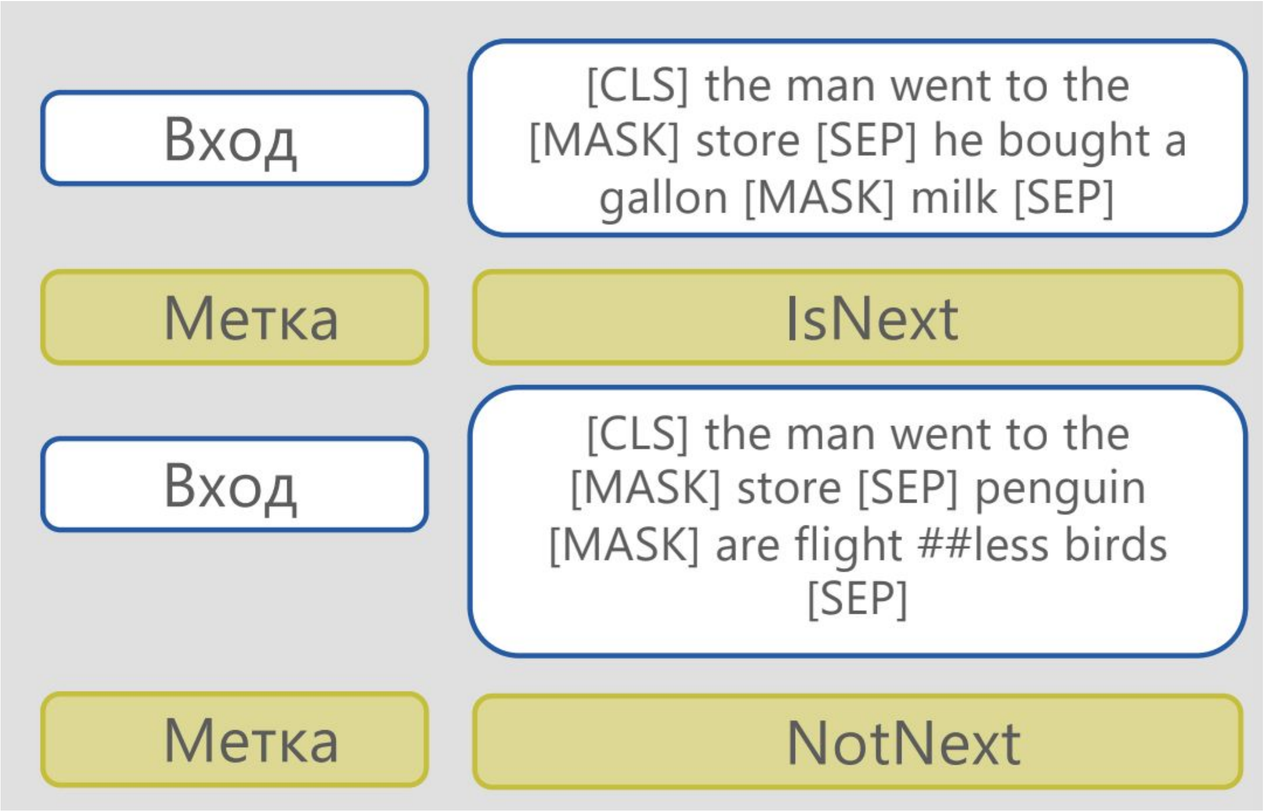
Masked Language Modeling

- 80%: токен маскируются управляющим токеном [MASK]
- 10%: токен заменяется на случайное слово
- 10%: без изменений



Next Sentence Prediction

- Бинарная классификация
- Линейный слой + софтмакс
- В целевых задачах с помощью [SEP] можем соединять question и paragraph (QA), или пары предложений (NLI)



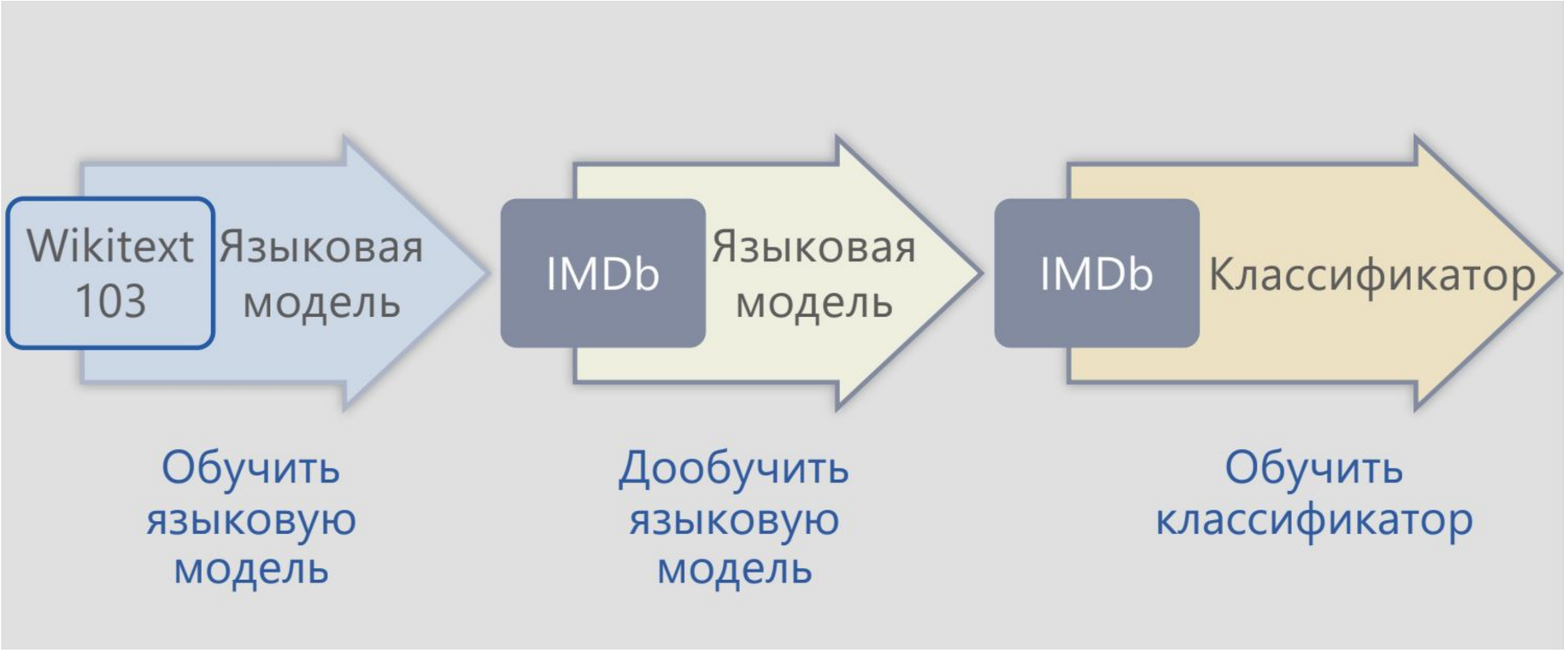
Training Objectives

Objective	Inputs	Targets
LM	[START]	I am happy to join with you today
MLM	I am [MASK] to join with you [MASK]	happy today
NSP	Sent1 [SEP] Next Sent or Sent1 [SEP] Random Sent	Next Sent/Random Sent
SOP	Sent1 [SEP] Sent2 or Sent2 [SEP] Sent1	in order/reversed
Discriminator (o/r)	I am thrilled to study with you today	o o r o r o o o
PLM	happy join with	today am I to you
seq2seq LM	I am happy to	join with you today
Span Mask	I am [MASK] [MASK] [MASK] with you today	happy to join
Text Infilling	I am [MASK] with you today	happy to join
Sent Shuffling	today you am I join with happy to	I am happy to join with you today
TLM	How [MASK] you [SEP] [MASK] vas-tu	are Comment

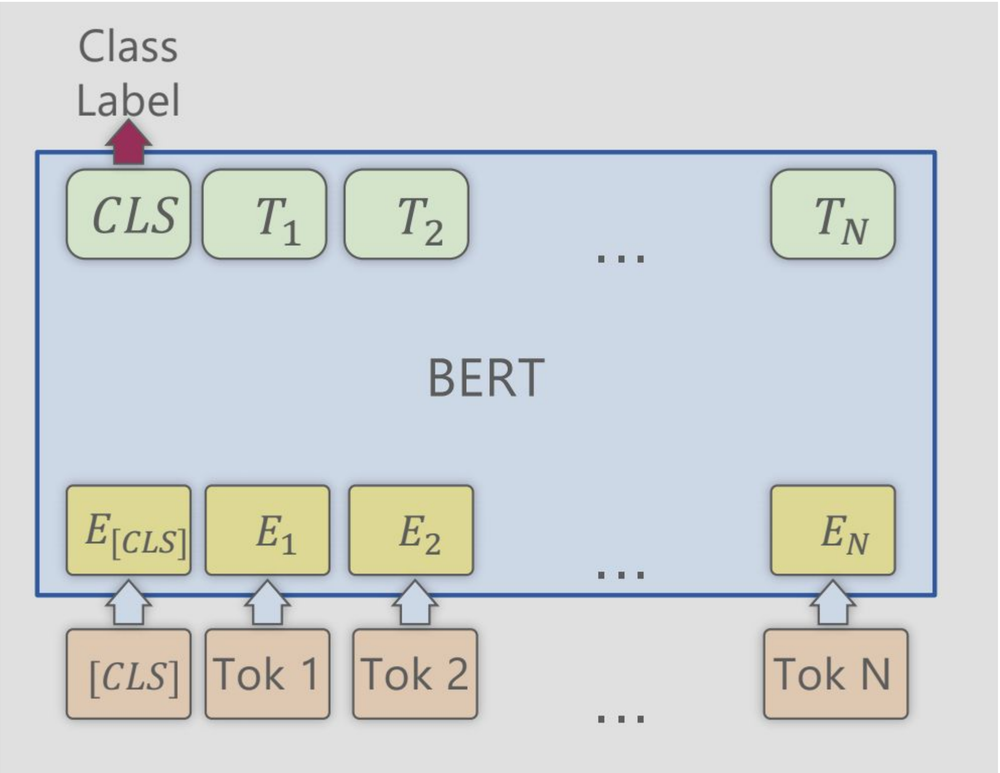
Обучение

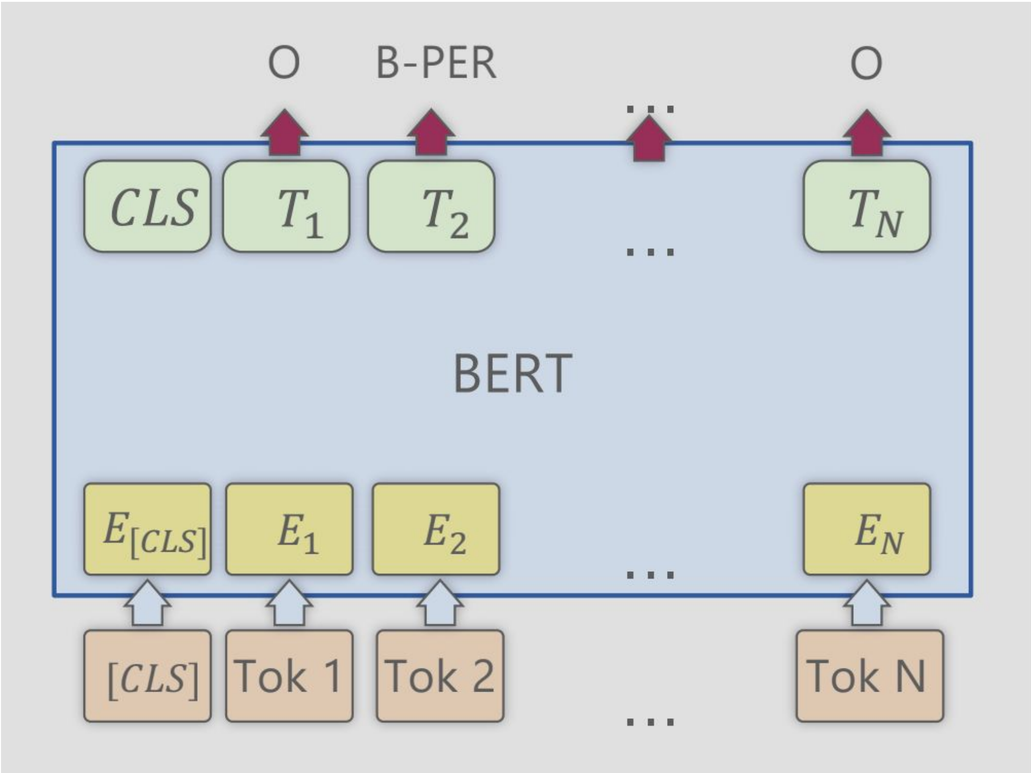

```
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
| first sequence    | second sequence |
```

Общая схема


Wikitext 103
Языковая модель
IMDb
Языковая модель
IMDb
Классификатор
Обучить языковую модель
Дообучить языковую модель
Обучить классификатор

Файн-тюнинг


Class
Label
CLS
T1
T2
...
TN
BERT
E[CLS]
E1
E2
...
EN
[CLS]
Tok 1
Tok 2
...
Tok N


O
B-PER
...
O
CLS
T1
T2
...
TN
BERT
E[CLS]
E1
E2
...
EN
[CLS]
Tok 1
Tok 2
...
Tok N

Файн-тюнинг

```
class BertForTokenClassification(BertPreTrainedModel):

    _keys_to_ignore_on_load_unexpected = [r"pooler"]

    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config, add_pooling_layer=False)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
```

Файн-тюнинг

```
class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, config.num_labels)
```

Пулинг

- [CLS]-пулинг – вектор управляющего токена CLS на последнем слое
- MEAN-пулинг – усреднение векторов слов на последнем слое
- MAX-пулинг – покомпонентный максимум векторов слов на последнем слое