

Трансформеры • Encoder

- Разберем архитектуру трансформера
- Рассмотрим Scaled Dot Product Attention
- Посмотрим на Bert

- независимые от контекста: bag of words, tf-idf
- основанные на близости распределений: word2vec, GloVe, fasttext
- RNN-based подходы – SOTA* до появления трансформеров

Основа – BiLSTM сеть

Идея – учитывает и характеристику слова, и контекста, в котором оно расположено.

I am talking to my ... (friend | dog | mom | window)

Какое слово следующее?

$P(\text{next}=\text{"friend"} \mid \text{"I am talking to my"})$

$P(\text{next}=\text{"dog"} \mid \text{"I am talking to my"})$

$P(\text{next}=\text{"mom"} \mid \text{"I am talking to my"})$

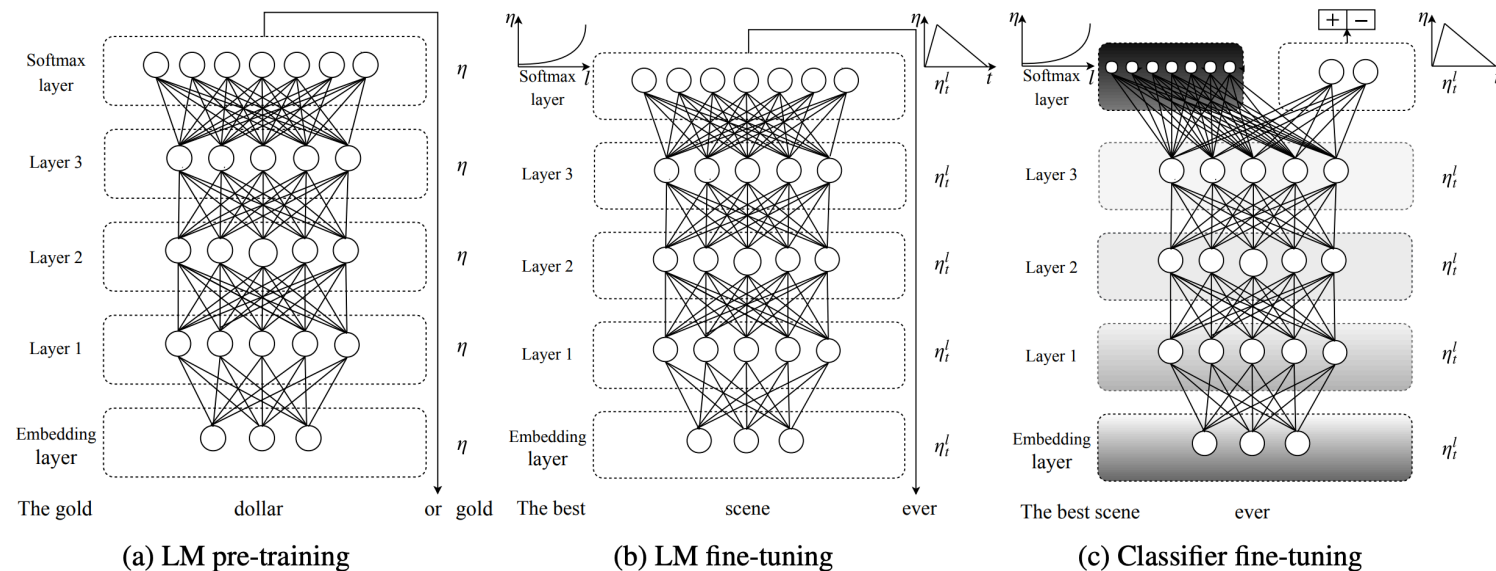
$P(\text{next}=\text{"window"} \mid \text{"I am talking to my"})$



Universal Language Model Fine-tuning (ULMFiT)

Основа – **регуляризованная BiLSTM** сеть

Идея – обучена на wiki, позволяет решать задачи, связанные с текстом, после fine-tuning'a



Attention Is All You Need • Transformer

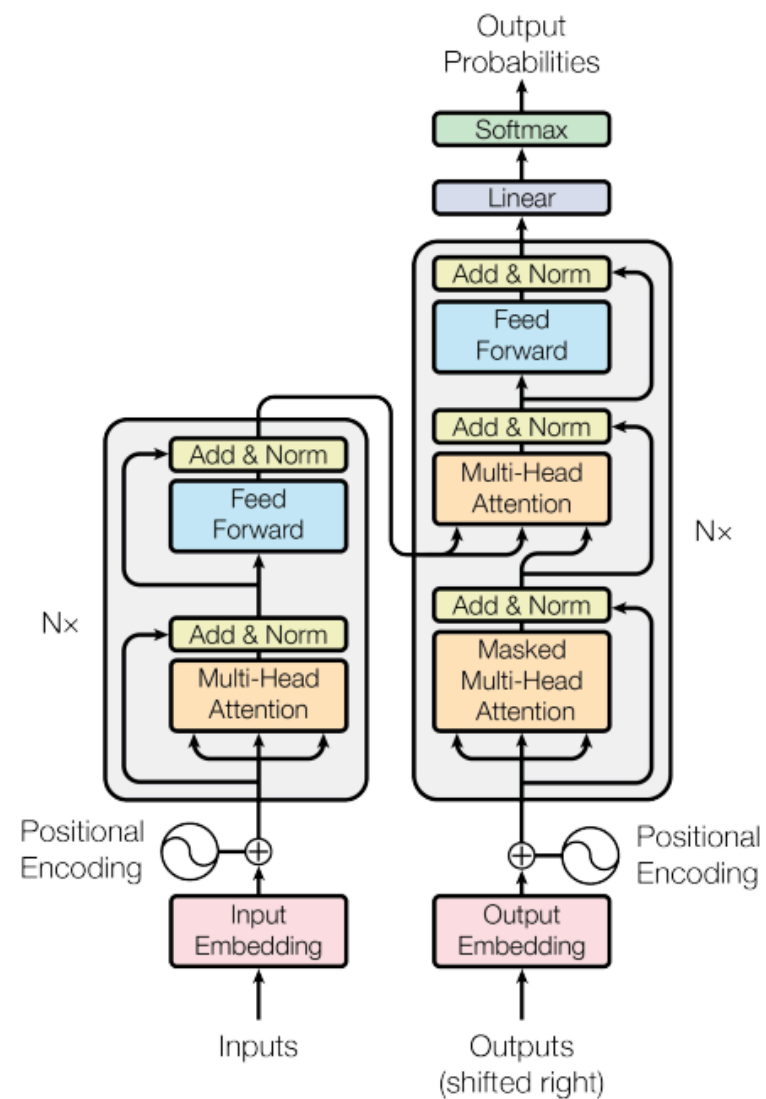
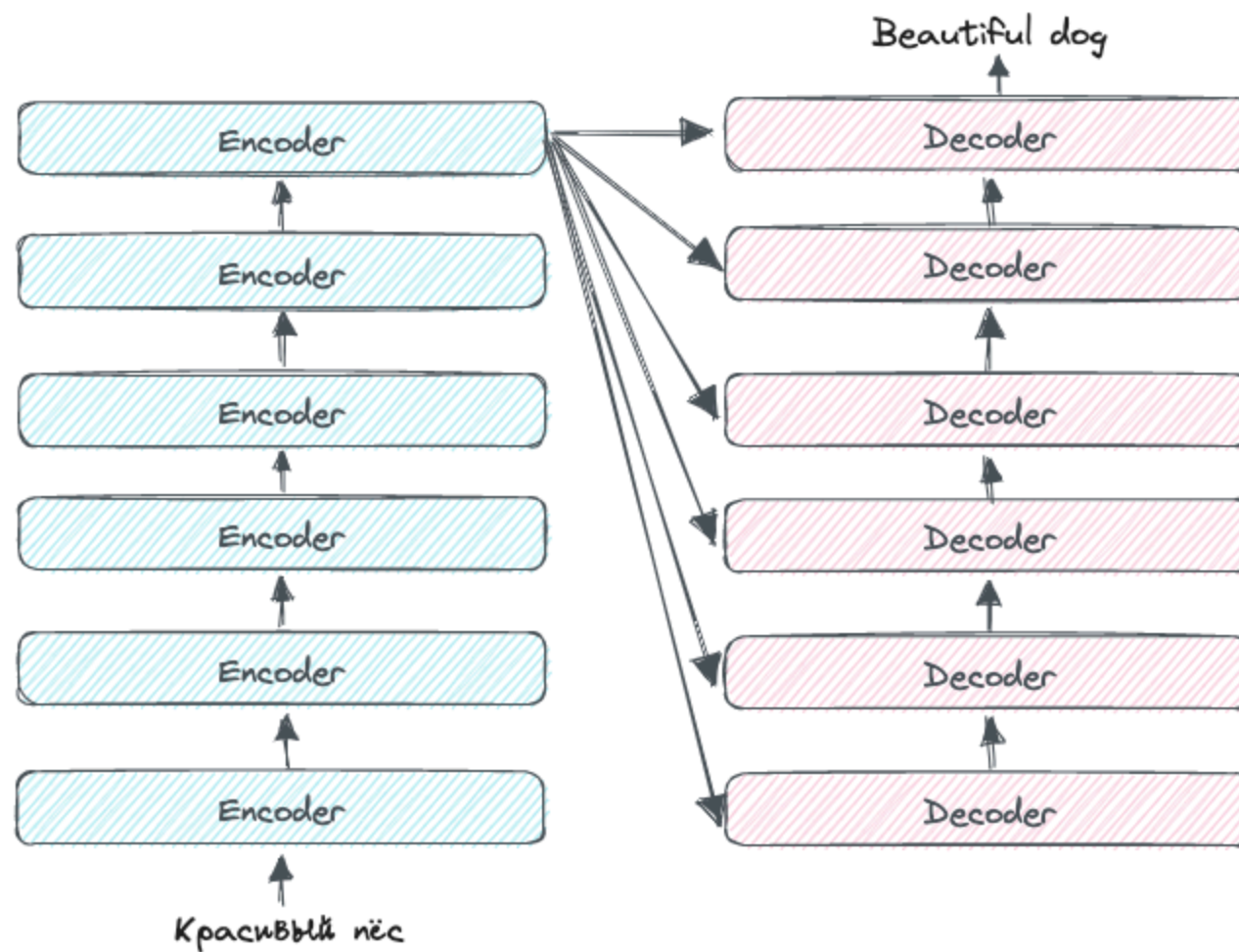
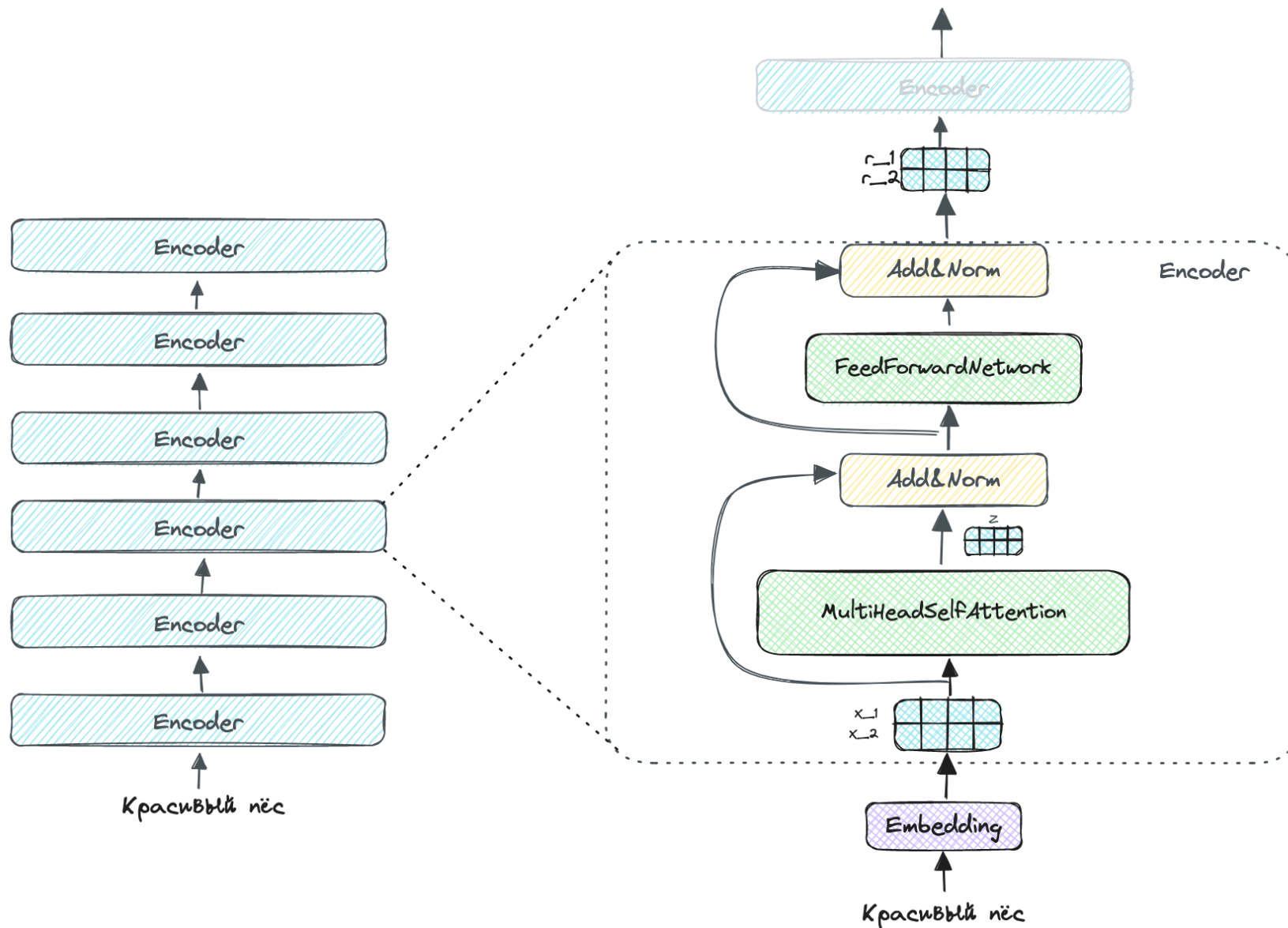


Figure 1: The Transformer - model architecture.

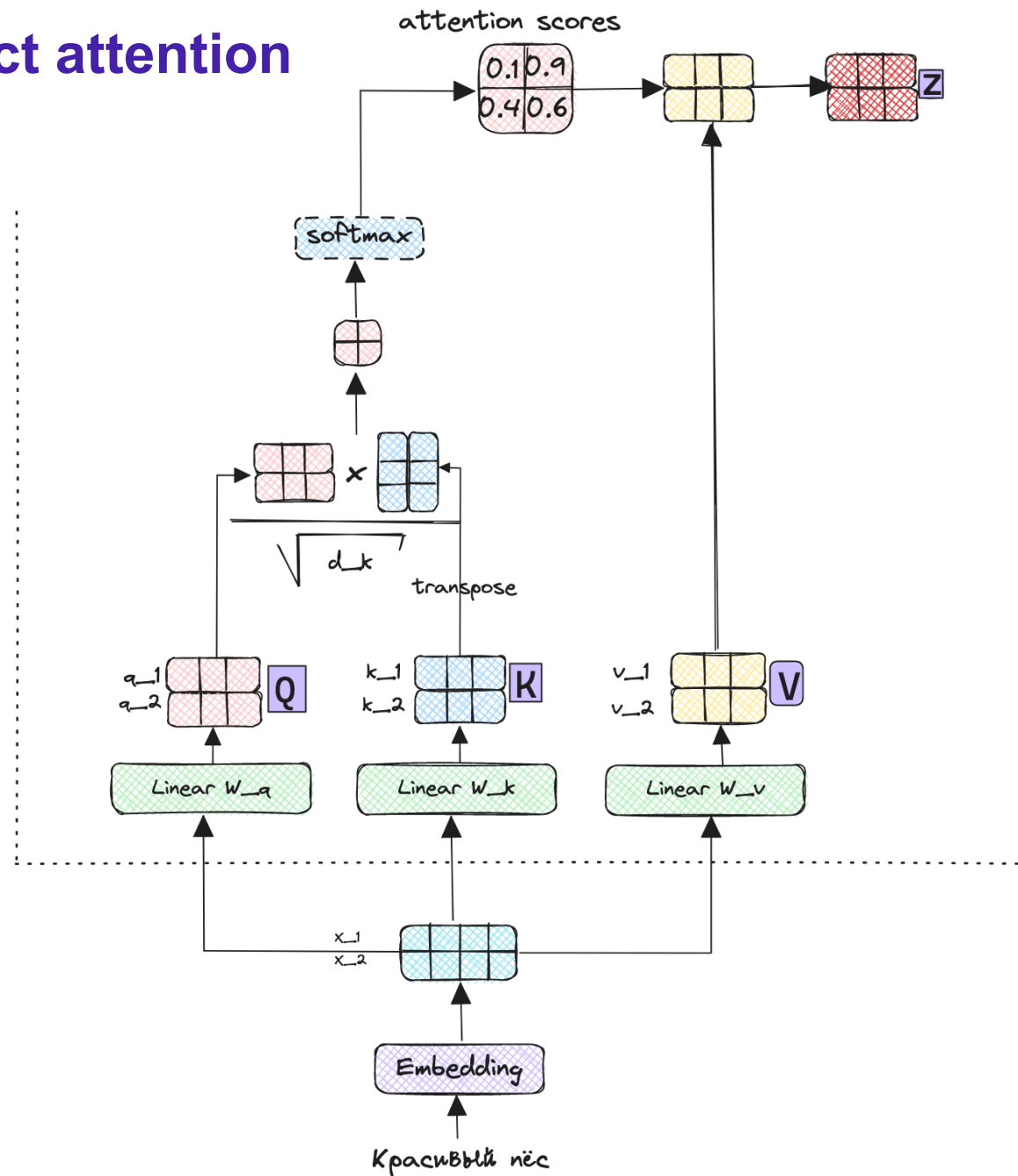
Transformer



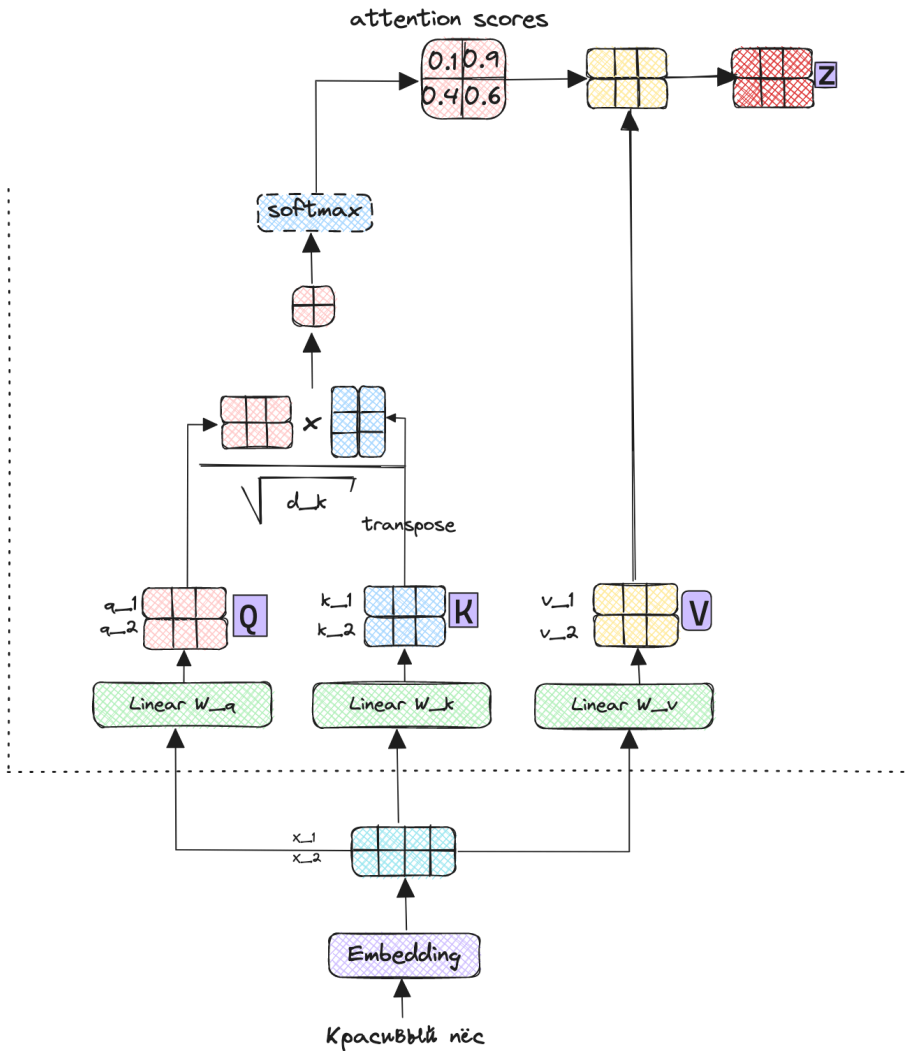
Encoder



Scaled dot product attention



В коде все проще



```
class ScaledDotProductAttention(nn.Module):
    def __init__(self, input_dim, output_dim):
        super().__init__()
        self.query = nn.Linear(input_dim, output_dim)
        self.key = nn.Linear(input_dim, output_dim)
        self.value = nn.Linear(input_dim, output_dim)

    def forward(self, embedding, mask=None):
        Q = self.query(embedding)
        K = self.key(embedding)
        V = self.value(embedding)
        d_k = torch.tensor(K.size(-1))

        scores = torch.bmm(Q, K.transpose(-2, -1))

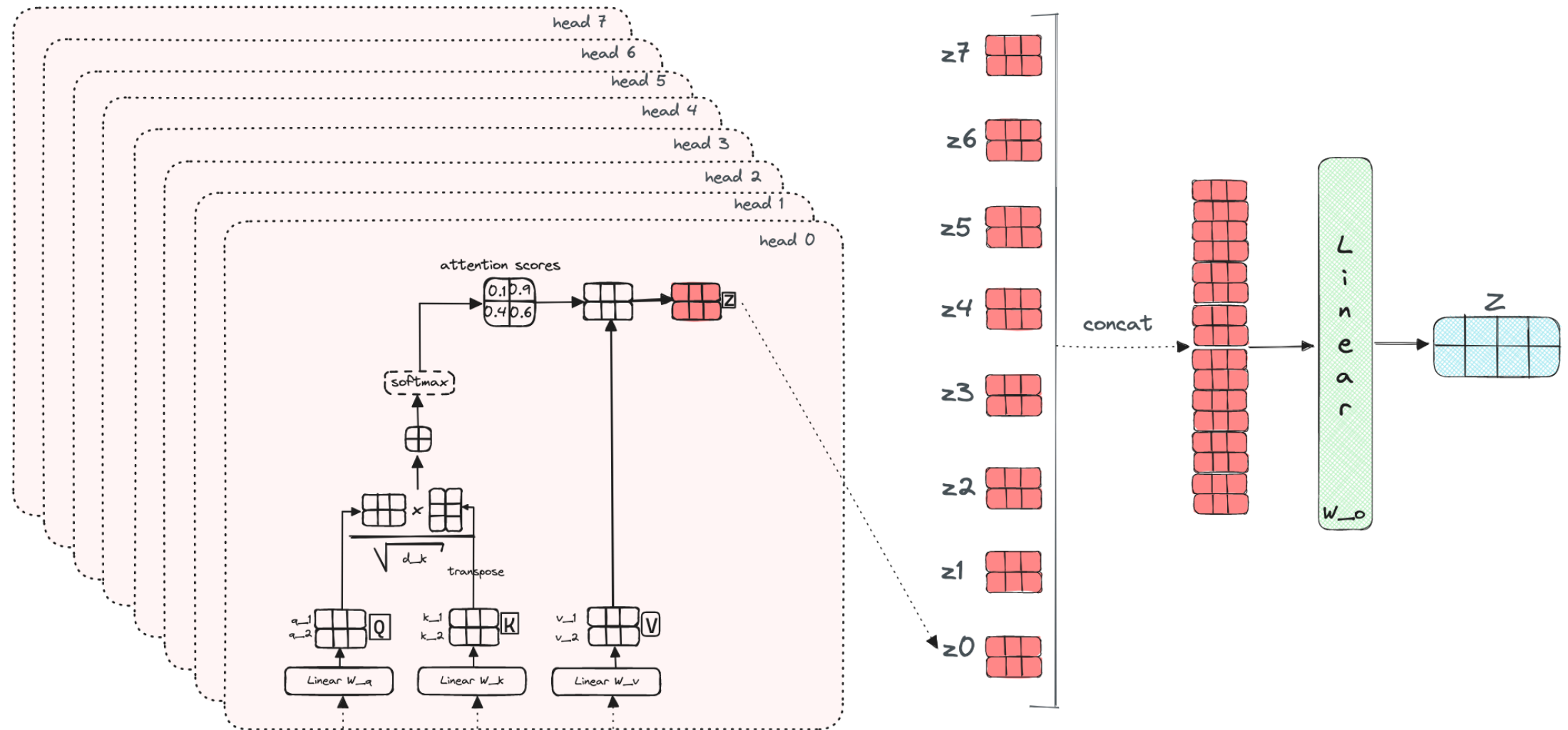
        scores = torch.div(scores, torch.sqrt(d_k))

        attention_scores = F.softmax(scores, dim=-1)

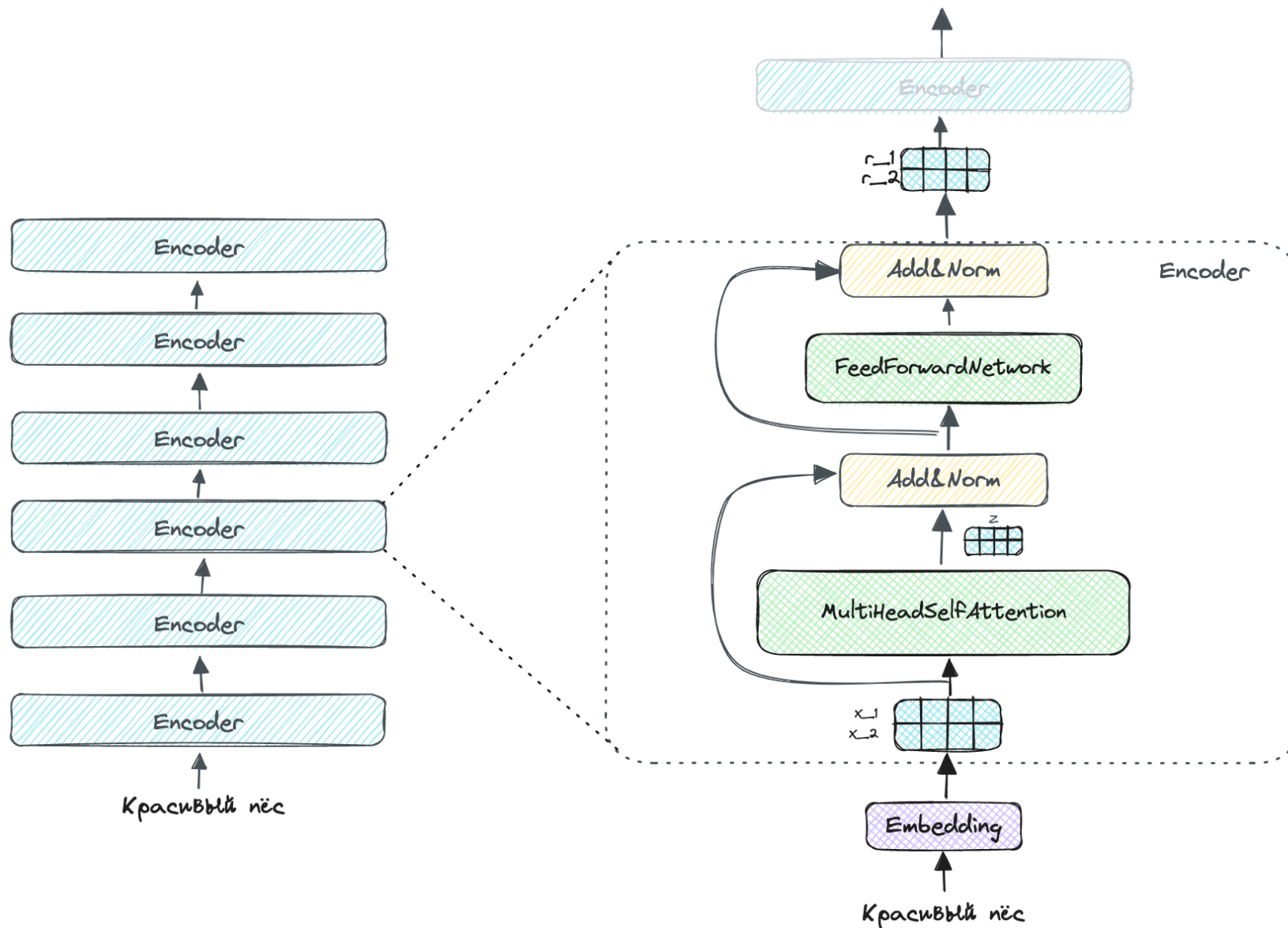
        Z = torch.bmm(attention_scores, V)

        return Z
```

MultiHead Attention

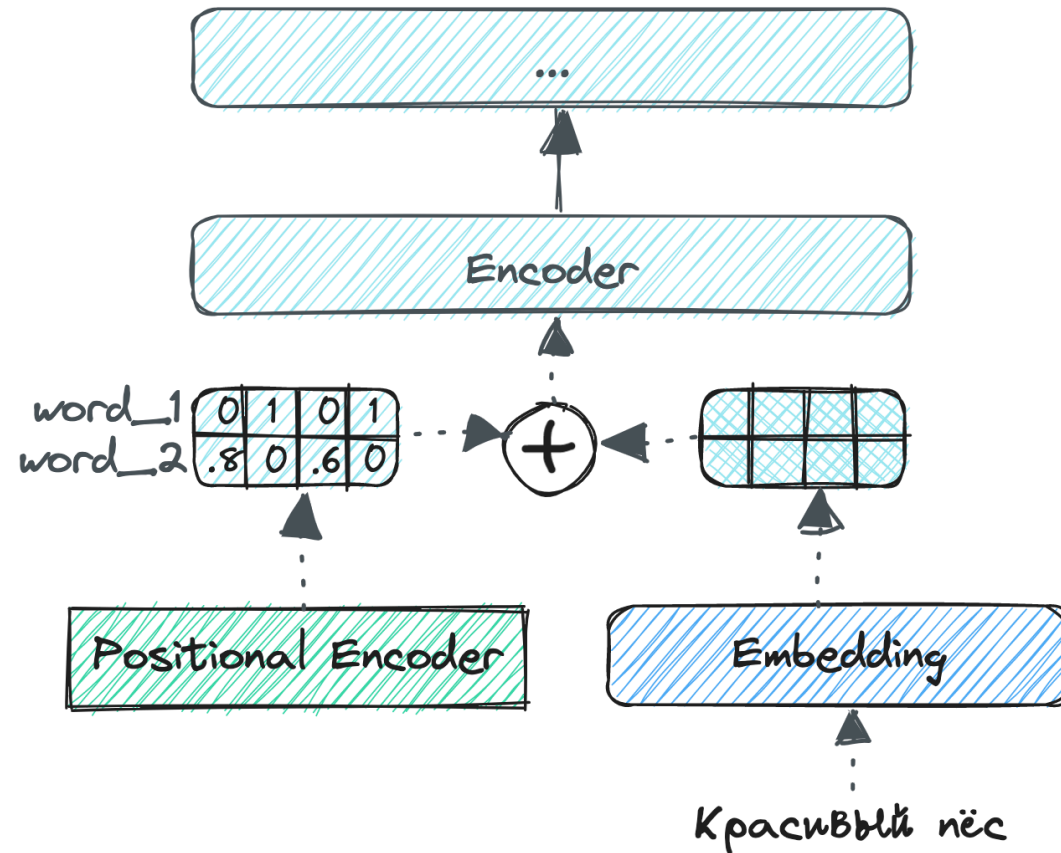


Encoder



Positional Encoding

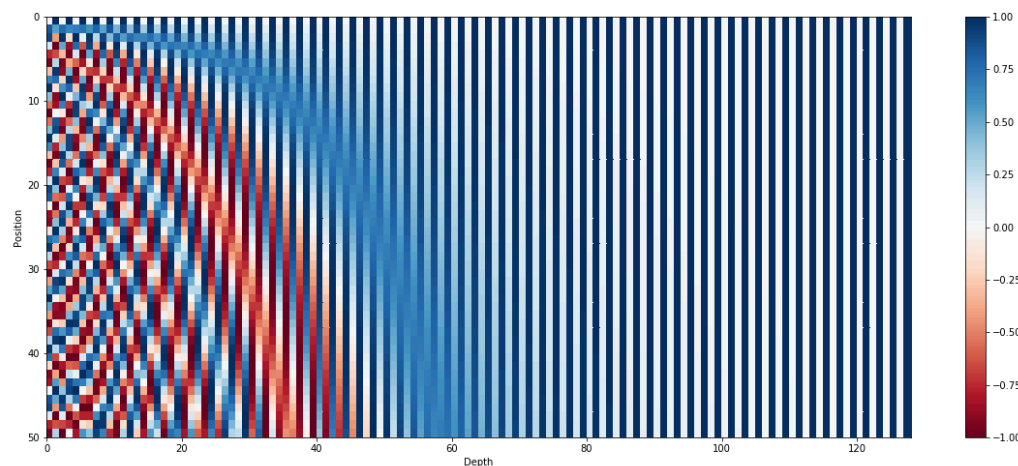
Такой подход позволяет учитывать позиции слов в предложениях (еще и относительно друг друга). Полученный вектор добавляем к эмбедингу слова.



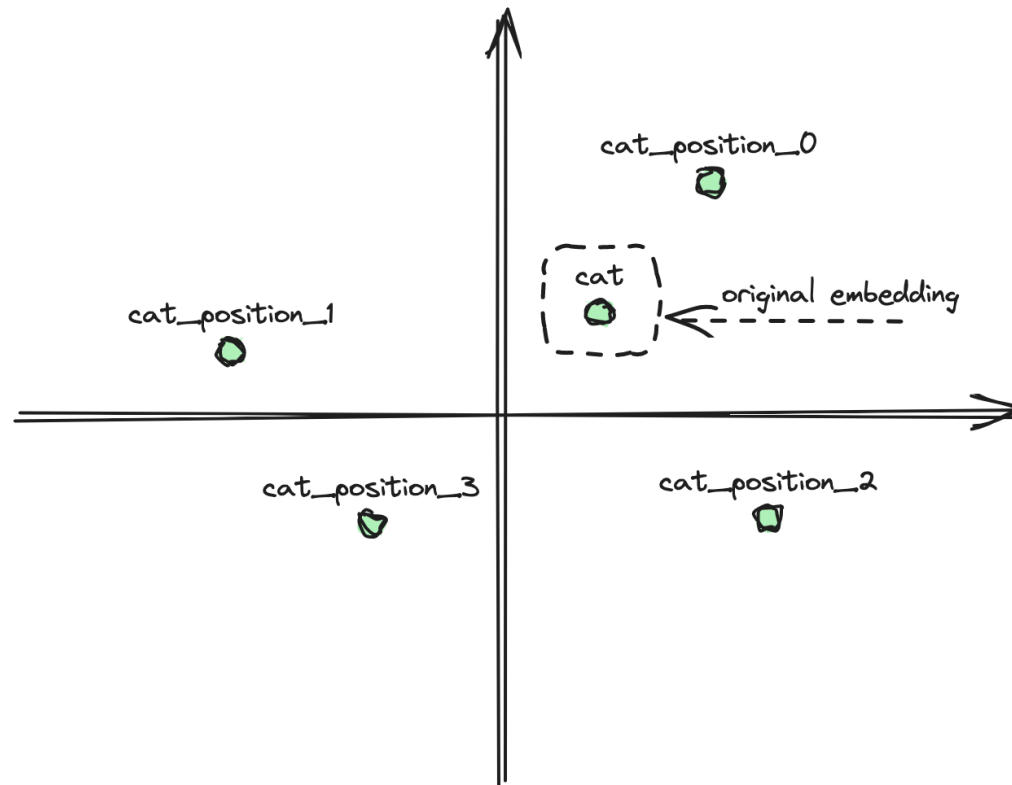
Positional Encoding

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

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

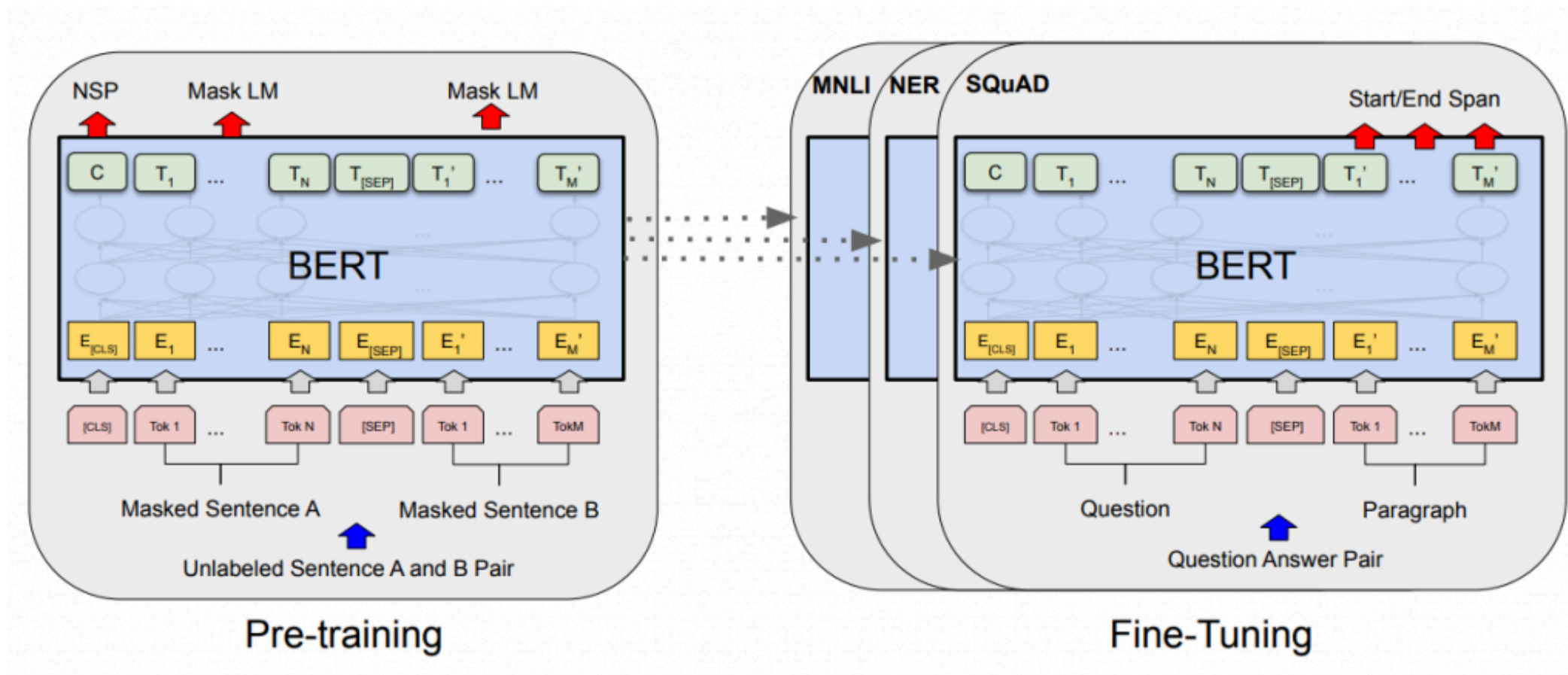


Positional Encoding



BERT • Bidirectional Encoder Representations from Transformers






NLI - Natural Language Inference

Задачи

BERT – языковая модель, т.е. он не создавался для решения фиксированной задачи. В зависимости от формата выходного слоя он может решать разные задачи.

- если выходной слой состоит из одного нейрона, то можно решить задачу бинарной классификации (или решить задачу NSP)
- если выходной слой с несколькими нейронами, то можно решить задачу многоклассовой классификации, классификации токенов и тд
- добавив некоторую логику обработки выходов можно решить задачу ответов на вопросы (подробнее можно почитать тут )
- можно прогнать тексты через предобученную модель и получить эмбединги

Большой обзор с деталями:  Large Language Models: BERT — Bidirectional Encoder Representations from Transformer

Masked Language Modelling

80% времени: заменяем случайные слова на токен **[MASK]**

10% времени: случайно заменяем одно слово другим

10% времени: ничего не делаем.

MLM: Какие слова должны стоять на месте токенов **[MASK]** ?

```
**[CLS]** the man **[MASK]** to the store **[SEP]** penguin **[MASK]** are flight ##less birds **[SEP]**
```

NSP – Next Sentence Prediction

NSP: Является ли второе предложение продолжением первого?

```
**[CLS]** the man **[MASK]** to the store **[SEP]** penguin **[MASK]** are flight ##less birds **[SEP]**
```



- трансформеры - очень мощная модель
- трансформеры адаптированы под разные задачи: текст, изображения, звук и т.д.
- нужно использовать предобученные модели
- основные библиотеки
 - `transformers` от huggingface: <https://huggingface.co/>
 - NLP-курс от 🙌: <https://huggingface.co/learn/nlp-course/chapter0/1?fw=pt>
 - `sententce transformers` <https://www.sbert.net/>
 - `deep pavlov` <https://deeppavlov.ai/>