

курс «Глубокое обучение»

Трансформер

Александр Дьяконов

25 ноября 2021 года



План

seq2seq

attention

attention / self-attention – матричная запись

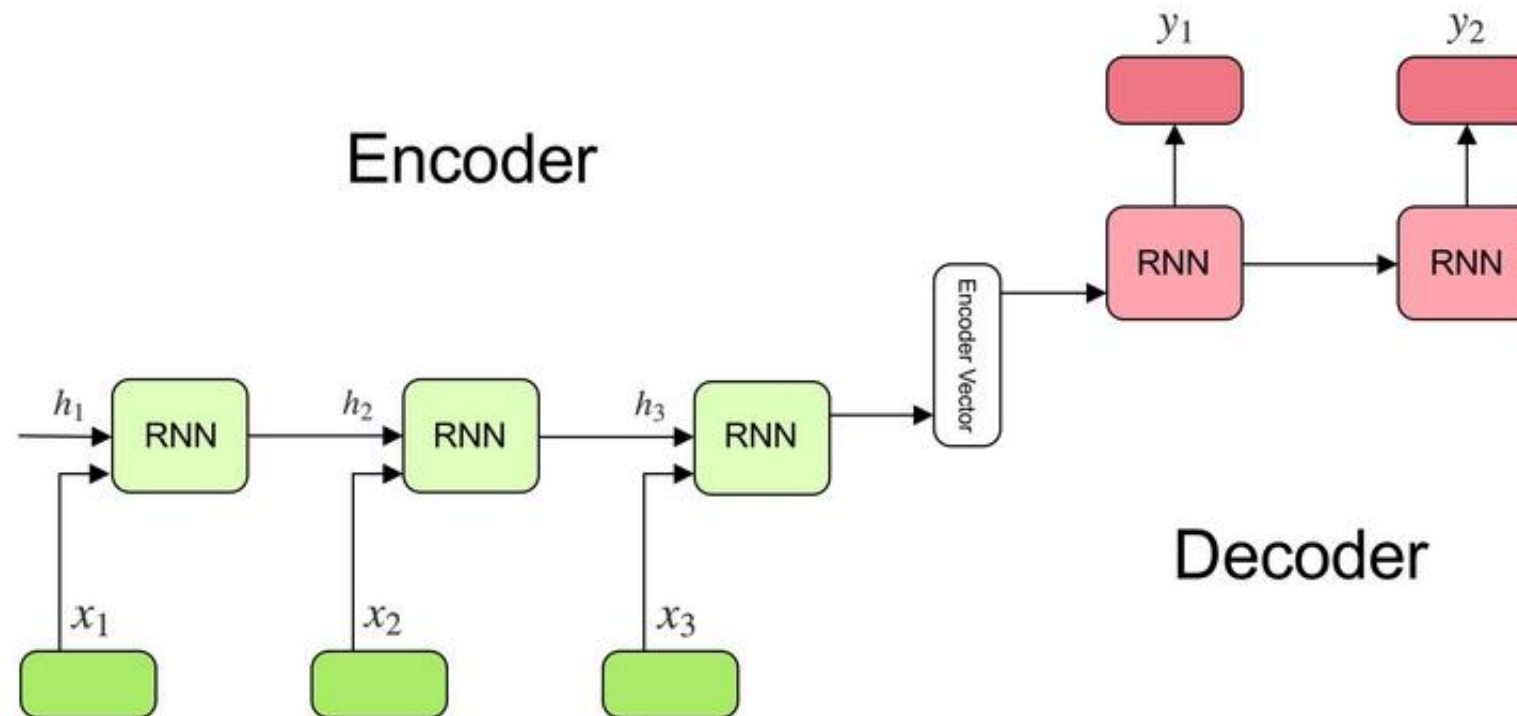
Transformer

Особенности обучения трансформера

BERT

Рис. из <https://towardsdatascience.com/@remykarem>

Модель seq2seq



<http://www.davidsbatista.net/blog/2020/01/25/Attention-seq2seq/>

Sutskever I. «Sequence to Sequence Learning with Neural Networks», 2014 // <https://arxiv.org/abs/1409.3215>

Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation

Kyunghyun Cho et. al. «Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine

Translation» <https://www.aclweb.org/anthology/D14-1179/>

Модель seq2seq: как переводить последовательность → последовательность

Многослойная (4 слоя) LSTM

размерность представления = 1000

входной словарь = 160,000

выходной словарь = 80,000

кодировщик (encoder) – декодировщик (decoder)

кодировщик: входная последовательность → вектор

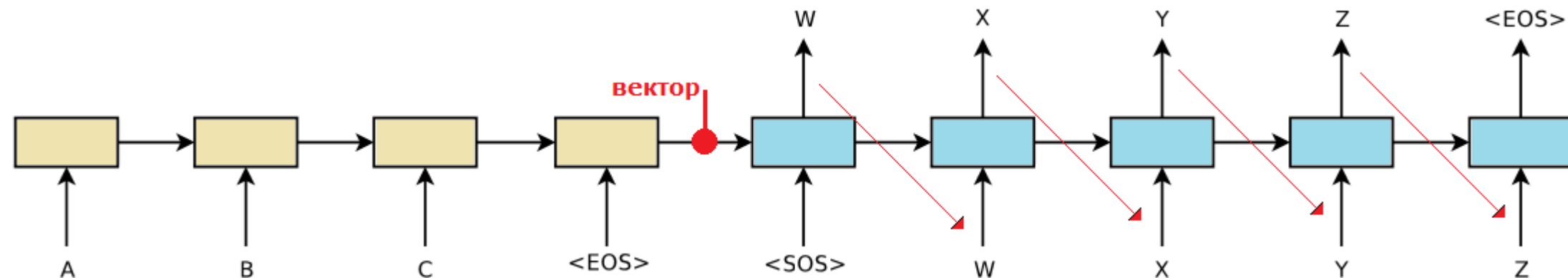
декодировщик: вектор → целевая последовательность

Это разные LSTM, у них разные параметры!

**Интересно: в задаче перевода качество повышалось
инвертирование порядка входа!**

Модель seq2seq

здесь декодировщик называют также **языковой моделью**



при работе (inference) – подаём на вход сгенерированное
при обучении – среднее ошибок на всех выходах (ex negative log prob)

тонкости:

на рисунке в декодировщике передаётся только его внутреннее состояние
выход кодировщика передаётся лишь первому элементу
можно его передавать всем! Чтобы информация о входе была у всех

Модель seq2seq

Внутреннее представление предложений!

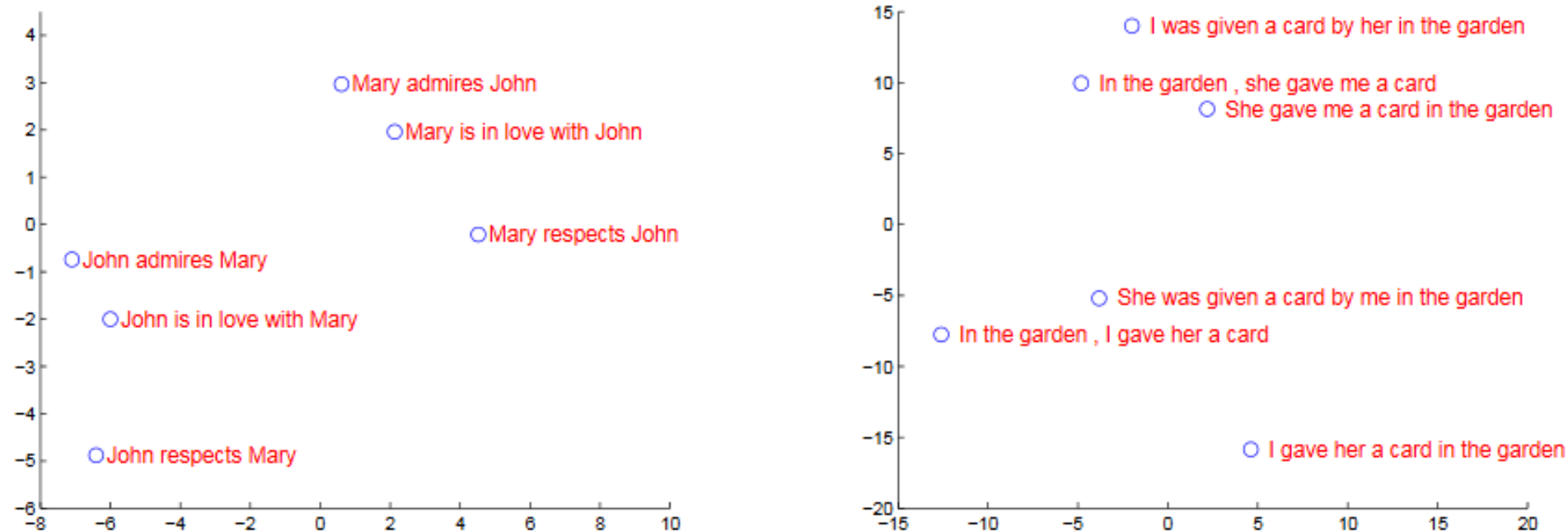


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

left-to-right beam-search decode потом будет

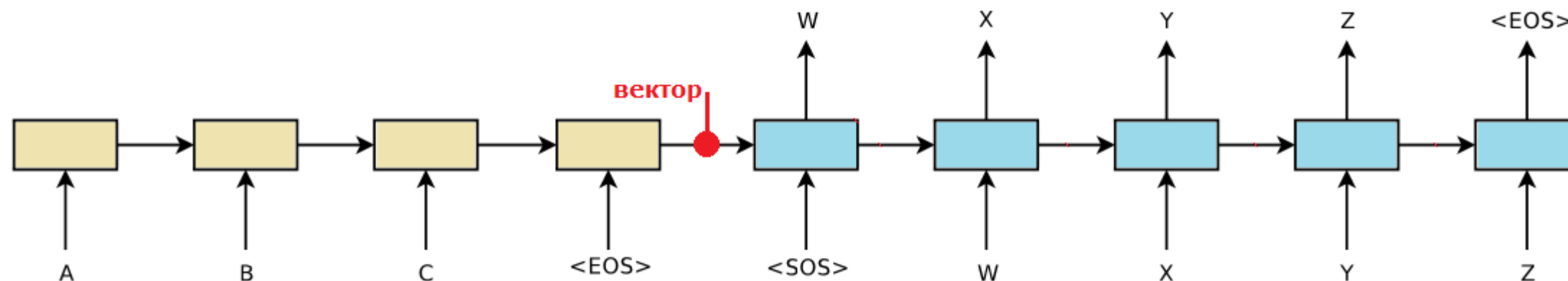
если выбираем лучшего следующего, не обязательно максимизируем качество

Обучение 10 дней

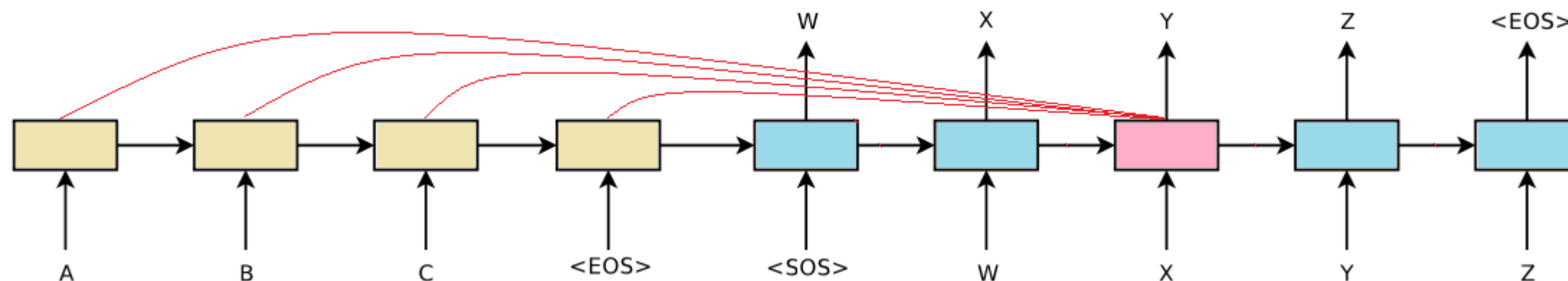
Тоже хороши ансамбли

Обобщения seq2seq

На одном нейроне вся информация о тексте... плохо
(особенно для длинных последовательностей)



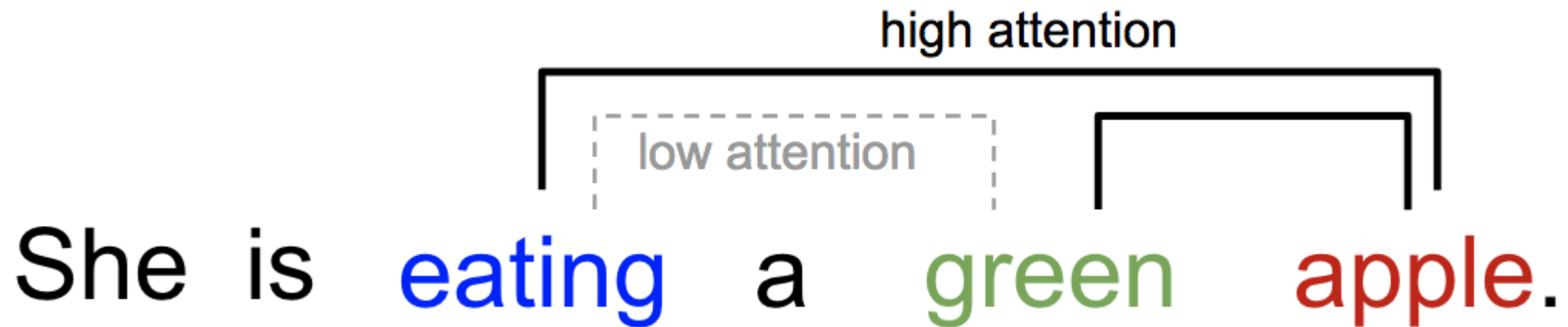
Решение – механизм внимания



Bahdanau et al. 2015 «Neural Machine Translation by Jointly Learning to Align and Translate»
// ICLR 2015 <https://arxiv.org/pdf/1409.0473.pdf>

Механизм внимания

Концепция: есть взаимосвязи между словами



**кодировщик передаёт в декодировщик не только одно состояние,
а состояния всех токенов!**

– но для этого нужен механизм пулинга посл-ти состояний любой длины
выход – пулинг по схожести

<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#born-for-translation>

Механизм внимания

Не будем пытаться закодировать всё предложение одним вектором!

Добавляется контекстный вектор (конкатенируется)

$$c_i = \sum_j \alpha_{ij} h_j$$

веса (softmax)

$$\alpha_{ij} = \exp(e_{ij}) / \sum_k \exp(e_{ik})$$

Насколько соответствуют состояния

$$e_{ij} = a(s_{i-1}, h_j)$$

Учитываются не только слова ДО, но и ПОСЛЕ!
Конкатенация состояния ДО и состояния ПОСЛЕ

Bidirectional RNN (BiRNN)

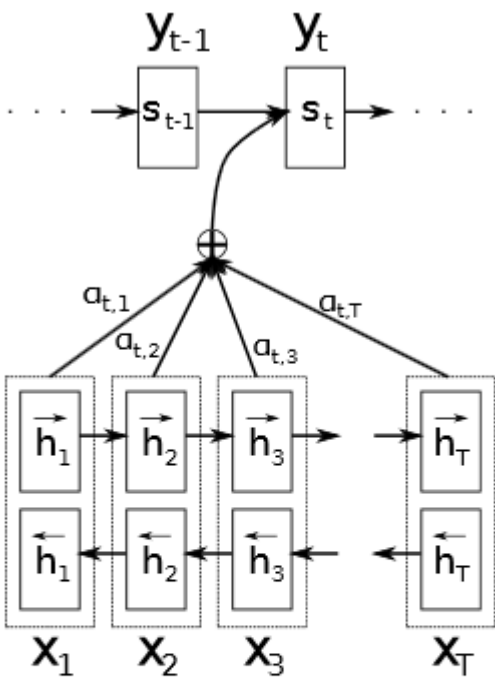


Figure 1: The graphical illustration of the proposed model trying to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) .

Механизм внимания

соответствие $e_{ij} = a(s_{i-1}, h_j)$ может быть:

Basic dot-product https://arxiv.org/pdf/1508.04025.pdf	$a(s, h) = s^T h$
Multiplicative attention https://arxiv.org/pdf/1508.04025.pdf	$a(s, h) = s^T W h$
Additive attention https://arxiv.org/pdf/1409.0473.pdf	$a(s, h) = w^T \tanh(W_1 s - W_2 h) = w^T \tanh(W[s; h])$
Scaled dot-product http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf	$a(s, h) = s^T h / \sqrt{d}$
Content-base attention https://arxiv.org/abs/1410.5401	$a(s, h) = \cos(s, h)$

+ разные нормировки по размерности

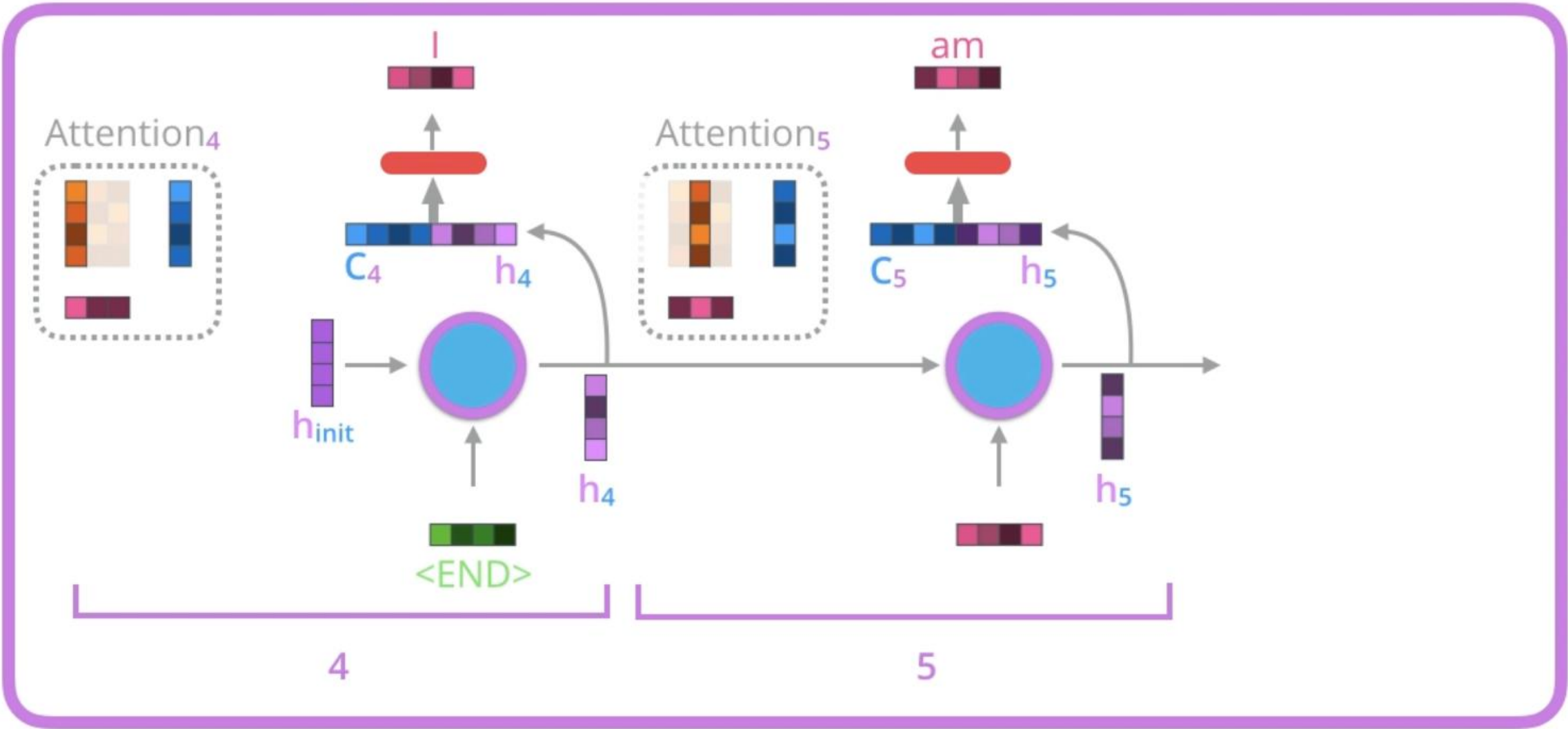
Thang Luong, Hieu Pham, Christopher D. Manning «Effective Approaches to Attention-based Neural Machine Translation» <https://www.aclweb.org/anthology/D15-1166.pdf>

Механизм внимания

Encoding Stage



Attention Decoding Stage



полученный в л/к состояний кодировщика вектор конкатенируем с текущим состоянием

Механизм внимания: получаем интерпретацию и выравнивание (alignment)

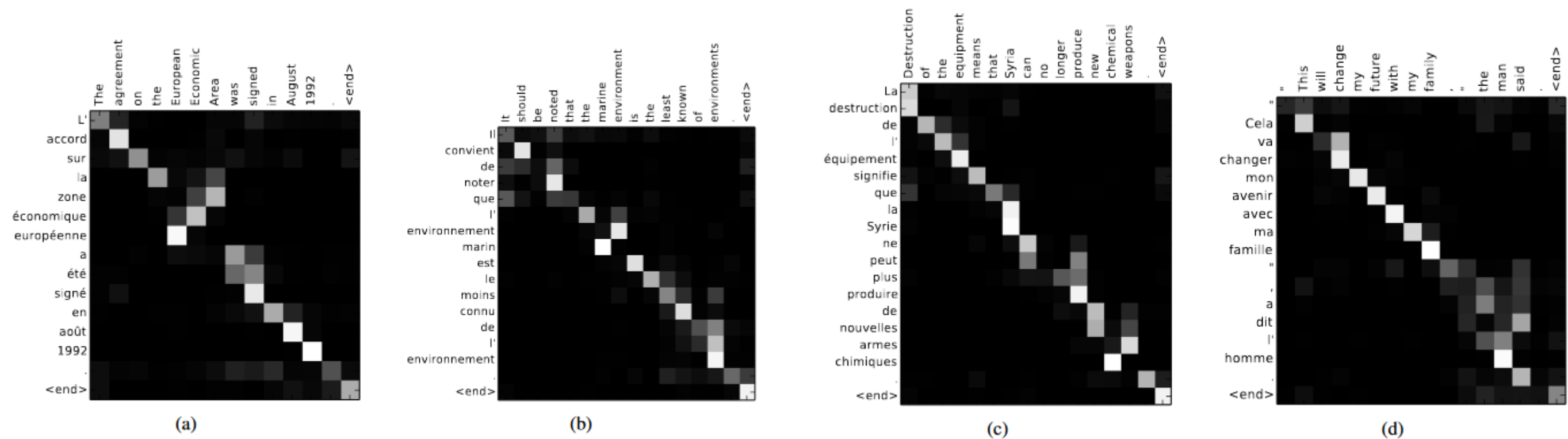
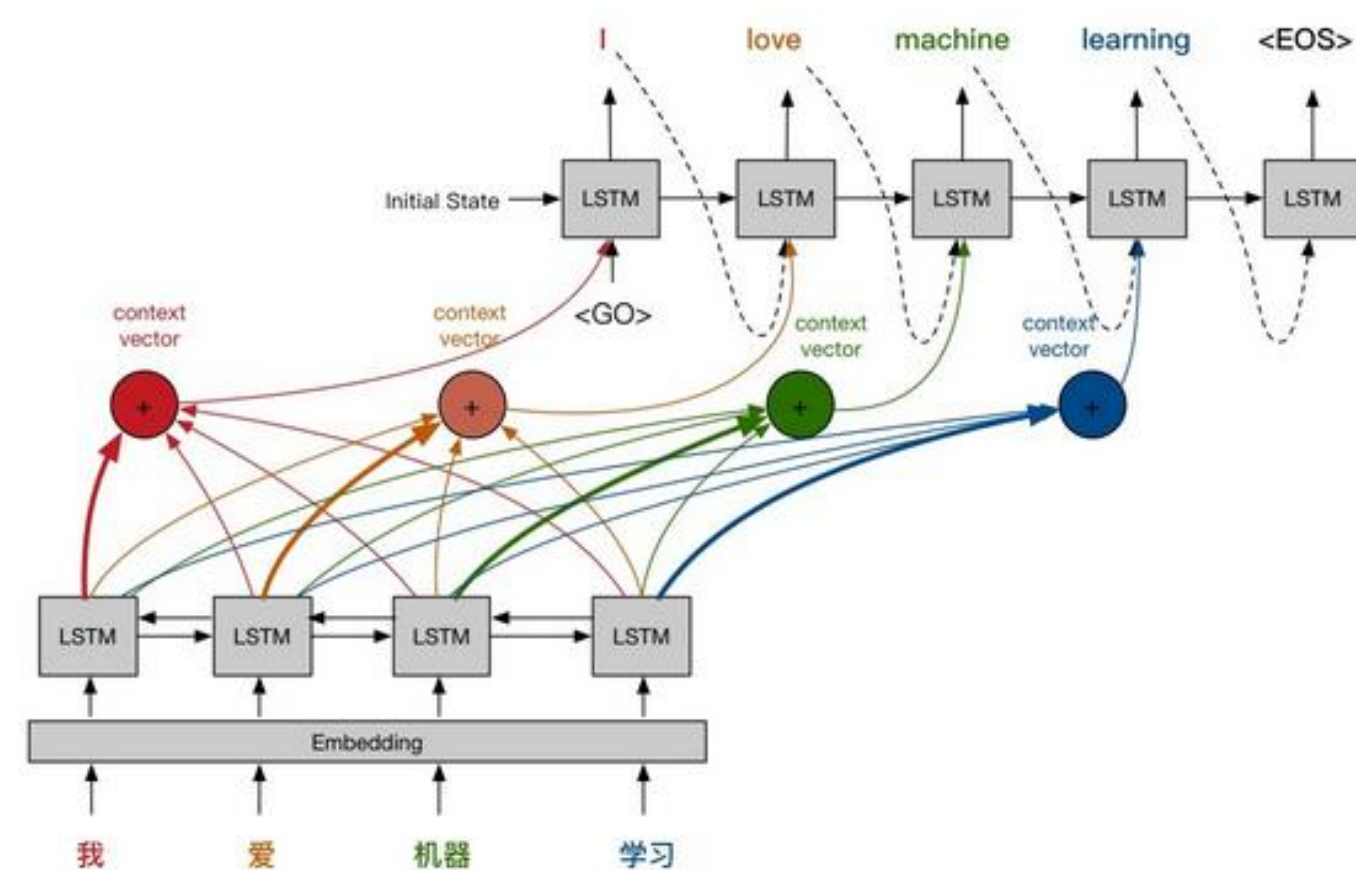
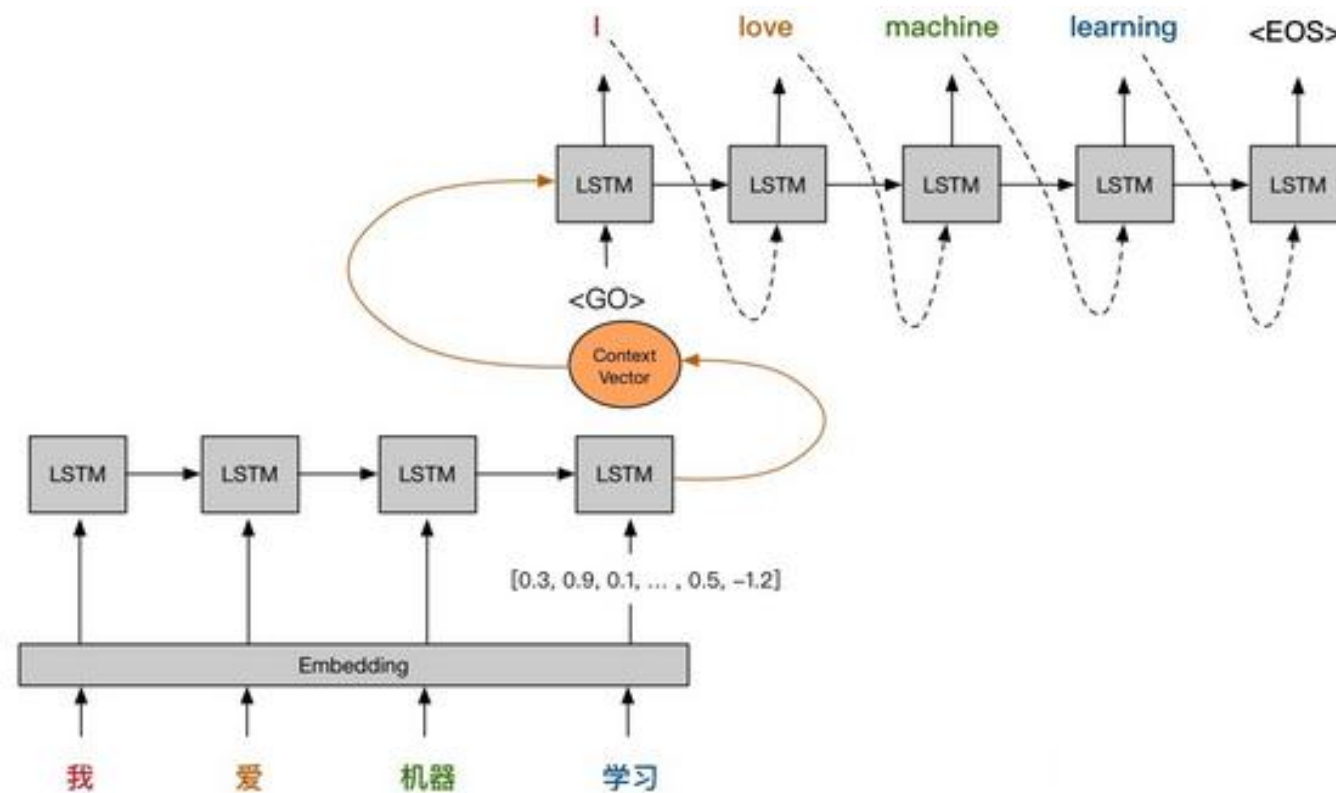


Figure 3: Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j -th source word for the i -th target word (see Eq. (6)), in grayscale (0: black, 1: white). (a) an arbitrary sentence. (b–d) three randomly selected samples among the sentences without any unknown words and of length between 10 and 20 words from the test set.

Bahdanau D. и др. «Neural Machine Translation by Jointly Learning to Align and Translate» // <https://arxiv.org/abs/1409.0473>

seq2seq vs attention



Внимание – техника вычисления взвешенной суммы значений (values) по запросу (query)
~ техника получения описания (representation) фиксированного размера по запросу

<https://zhuanlan.zhihu.com/p/37290775>

Плюсы механизма внимания (Attention)

- улучшает качество перевода (и не только)
 - решает проблему «узкого горла»
 - появляется интерпретируемость
- решает проблему «затухания сигнала» / исчезающего градиента
 - получаем выравнивание (alignment) «бесплатно» в переводе

Виды внимания

Self-Attention / intra-attention	к разным позициям одной и той же входной последовательности
Global / Soft	ко всему входу
Local / Hard	к части входа

<http://proceedings.mlr.press/v37/xuc15.pdf>

Виды внимания: Self-Attention

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

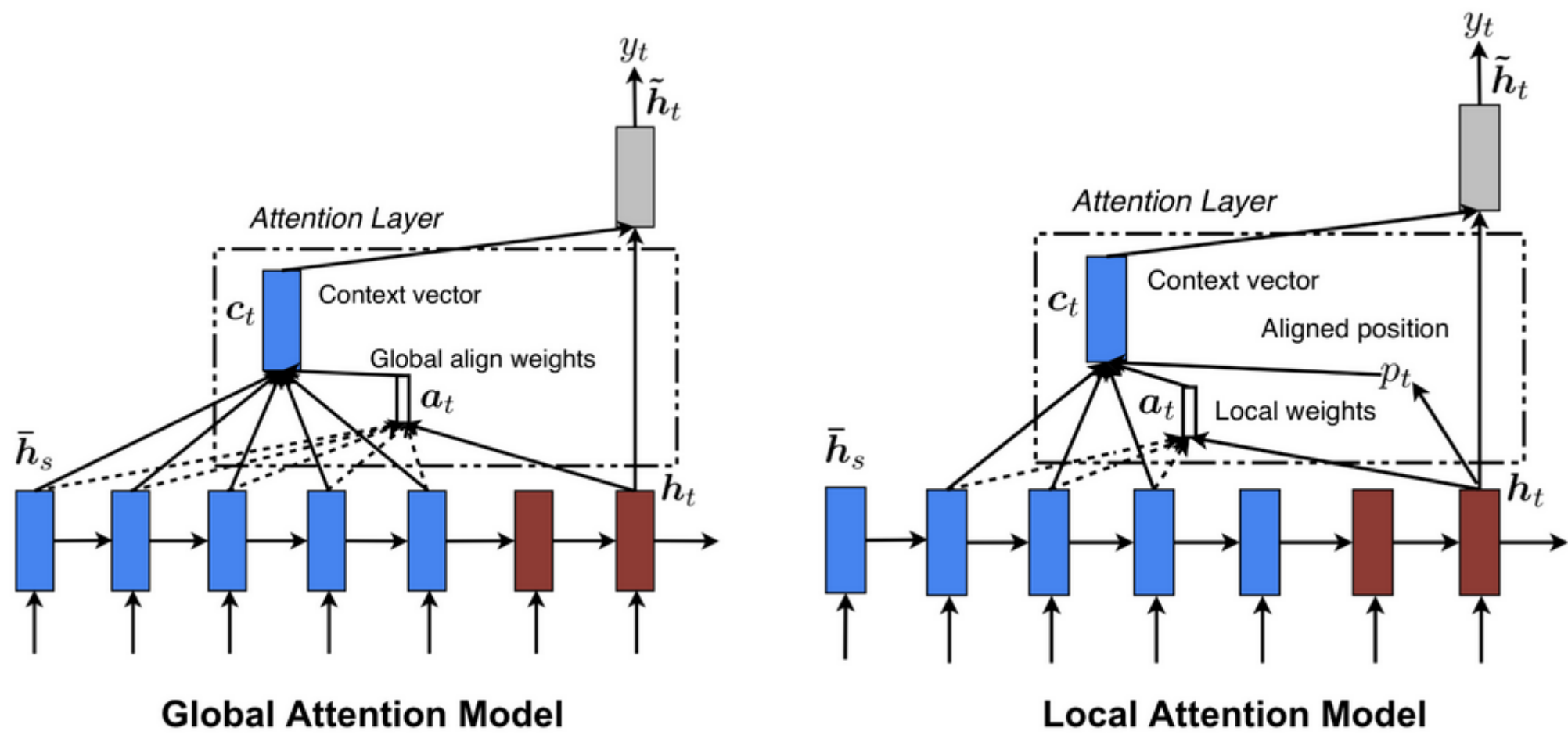
The FBI is chasing a criminal on the run .

The FBI is chasing a criminal on the run .

Figure 1: Illustration of our model while reading the sentence *The FBI is chasing a criminal on the run*. Color *red* represents the current word being fixated, *blue* represents memories. Shading indicates the degree of memory activation.

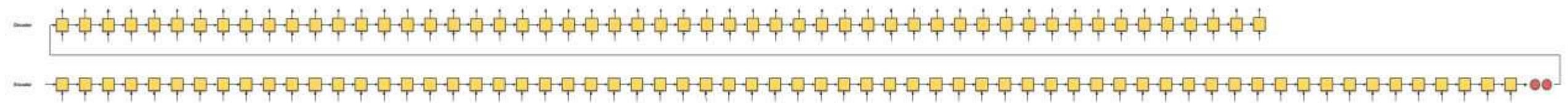
<https://arxiv.org/pdf/1601.06733.pdf>

Виды внимания: Global vs Local Attention

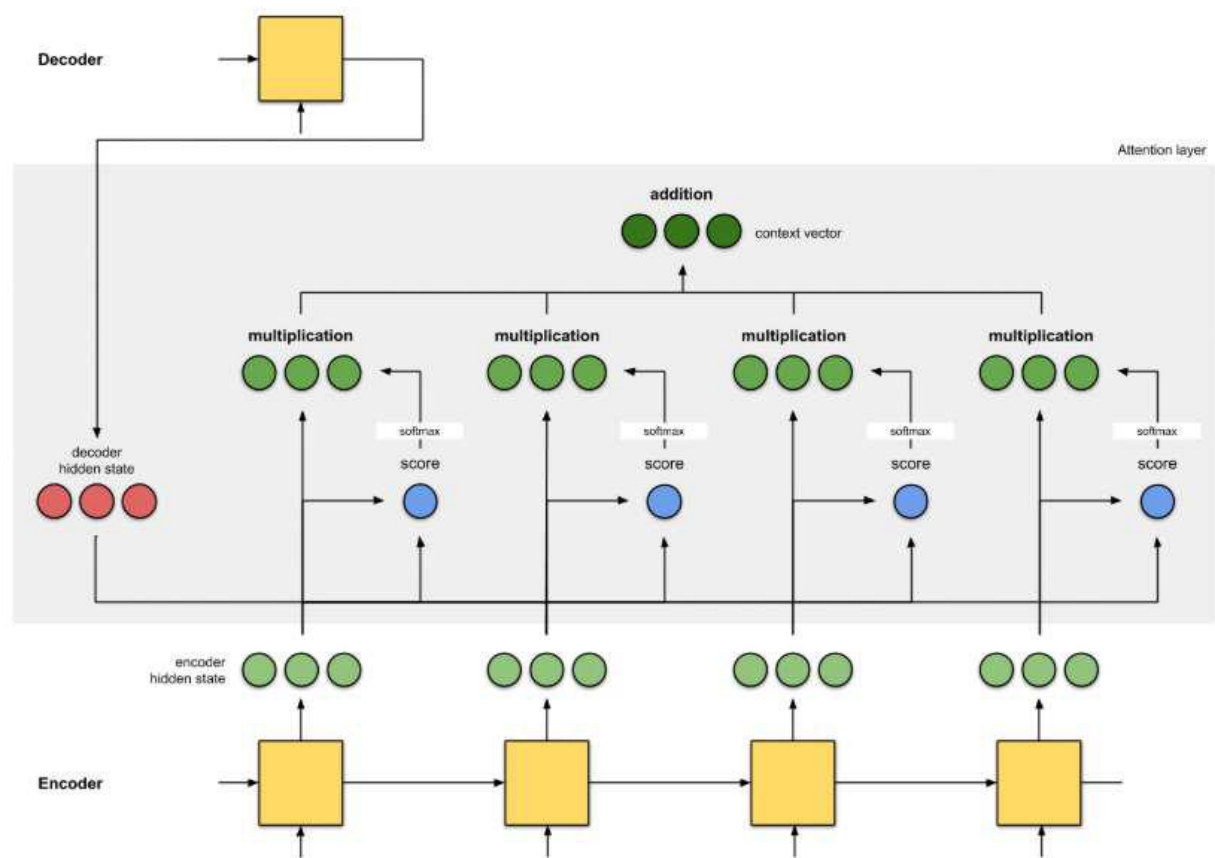


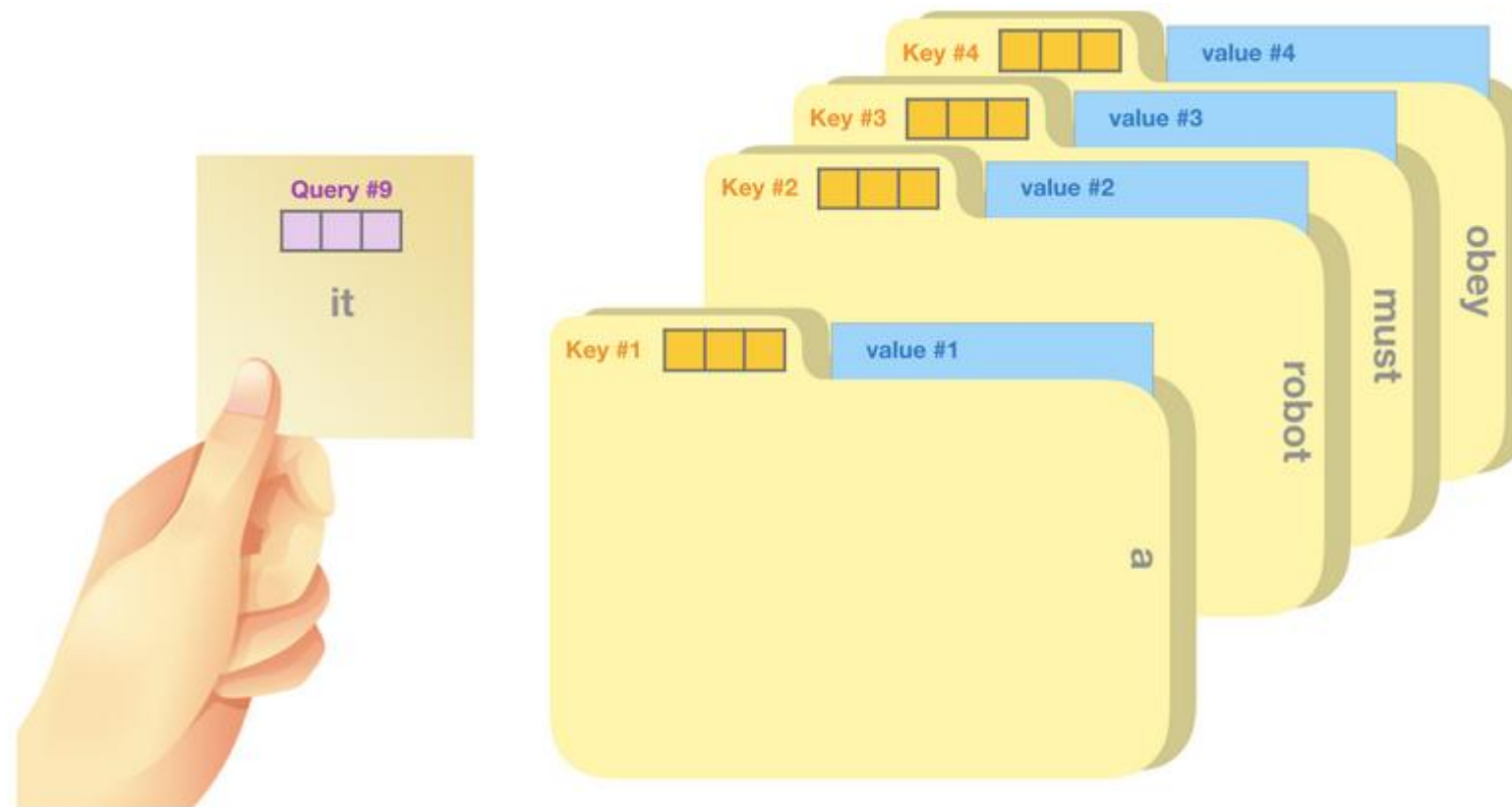
<https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#born-for-translation>

Напоминание: внимание seq2seq



seq2seq + attention: Query, Key, Value



attention / self-attention**Парадигма Query, Key, Value (запрос, ключ, значение)**

attention / self-attention – матричная запись**идея «attention»...****values** – $V_{d \times s}$ **keys** – $K_{d \times s}$ **query** – q **размерность** – d **число примеров** – s **«self-attention»****в X перечислены объекты внимания**

$$V_{d \times s} = W_{d \times D}^V X_{D \times s}$$

$$K_{d \times s} = W_{d \times D}^K X_{D \times s}$$

$$Q_{d \times s} = W_{d \times D}^Q X_{D \times s}$$

раз-ти ключей и значений м.б. разные

$$v = (V_{d \times s} \text{softmax}(K_{s \times d}^T q_{d \times 1})_{s \times 1})_{d \times 1}$$

в матричной форме + нормировка

$$(V \text{softmax}(\alpha K^T Q))_{d \times s}$$

$$\text{head}(X | W^V, W^K, W^Q) = W^V X \text{softmax}\left(\alpha (W^K X)^T W^Q X\right), \alpha = 1 / \sqrt{d}$$

+ сделать несколько параллельных матриц

$$W_{D \times 8d} \cdot \text{concat}[\text{head}(W_t^V, W_t^K, W_t^Q)_{d \times s}, \text{axis} = 0]_{t=1}^8$$

Минутка кода: attention

```
def attention(query, key, value, mask=None, dropout=None):  
    "Compute 'Scaled Dot Product Attention'"  
    d_k = query.size(-1)  
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)  
    if mask is not None:  
        scores = scores.masked_fill(mask == 0, -1e9)  
    p_attn = F.softmax(scores, dim = -1)  
    if dropout is not None:  
        p_attn = dropout(p_attn)  
    return torch.matmul(p_attn, value), p_attn
```

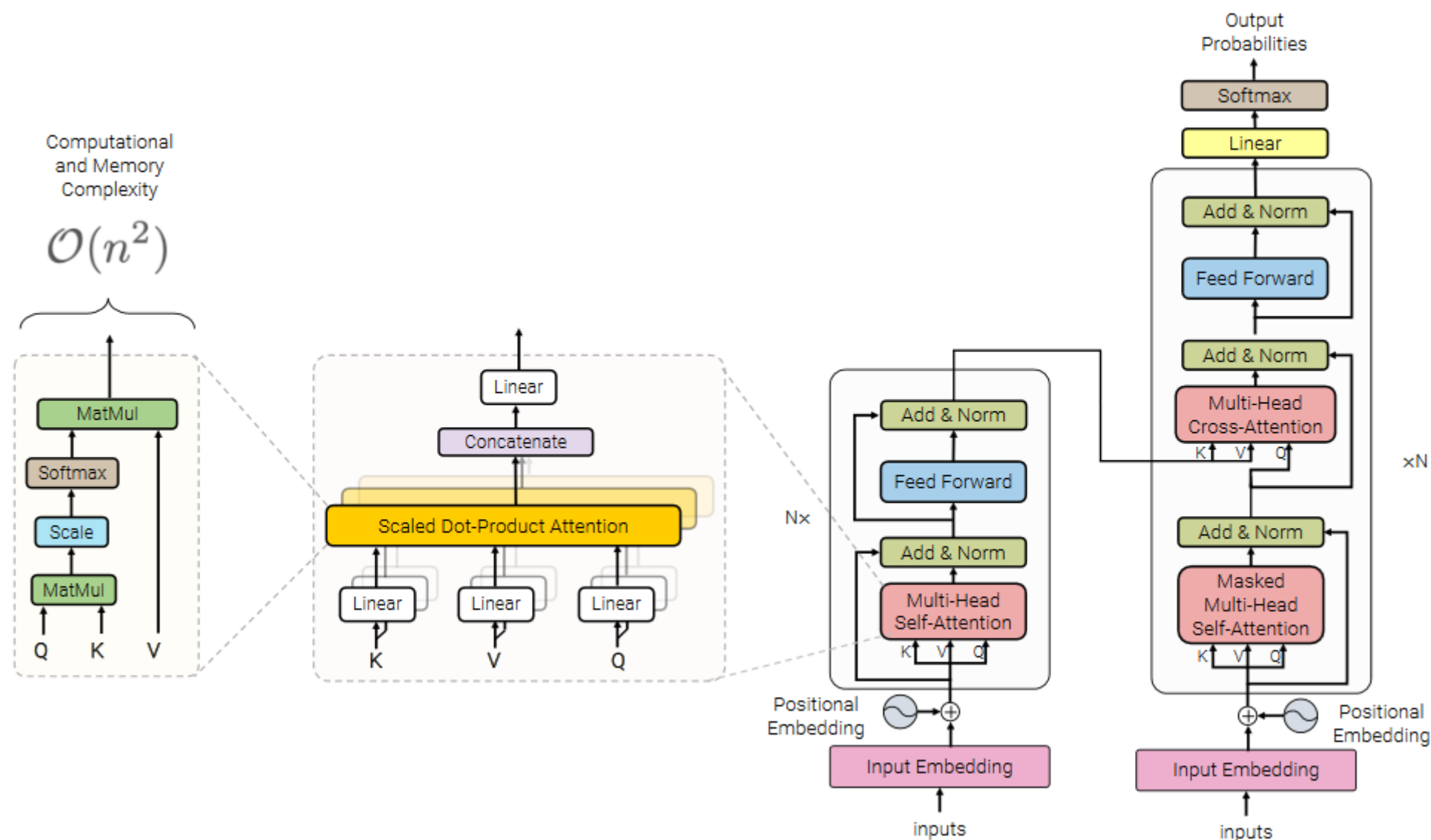
```
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4) # W_q, W_k, W_v, W
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        if mask is not None: # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)

        # Получить проекции, т.е. матрицы Q, K, V => h x d_k
        query, key, value = [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
                               for l, x in zip(self.linears, (query, key, value))]

        # само внимание.
        x, self.attn = attention(query, key, value, mask=mask, dropout=self.dropout)
        # конкатенация + линейность
        x = x.transpose(1, 2).contiguous().view(nbatches, -1, self.h * self.d_k)
        return self.linears[-1](x)
```

Transformer: Основная идея «Parallelized Attention»



Vaswani et al. «Attention Is All You Need» <https://arxiv.org/abs/1706.03762>

Дальше картинки из <https://jalammr.github.io/illustrated-transformer/>

Transformer: главное

трансформер – стекинг трансформер-блоков
послойное уточнение представлений токенов

представления токенов не меняют размерность по слоям D
не путать с внутренними размерностями d (запросов, ключей и значений)
и исходными D_{ONE} (на вход трансформеру)

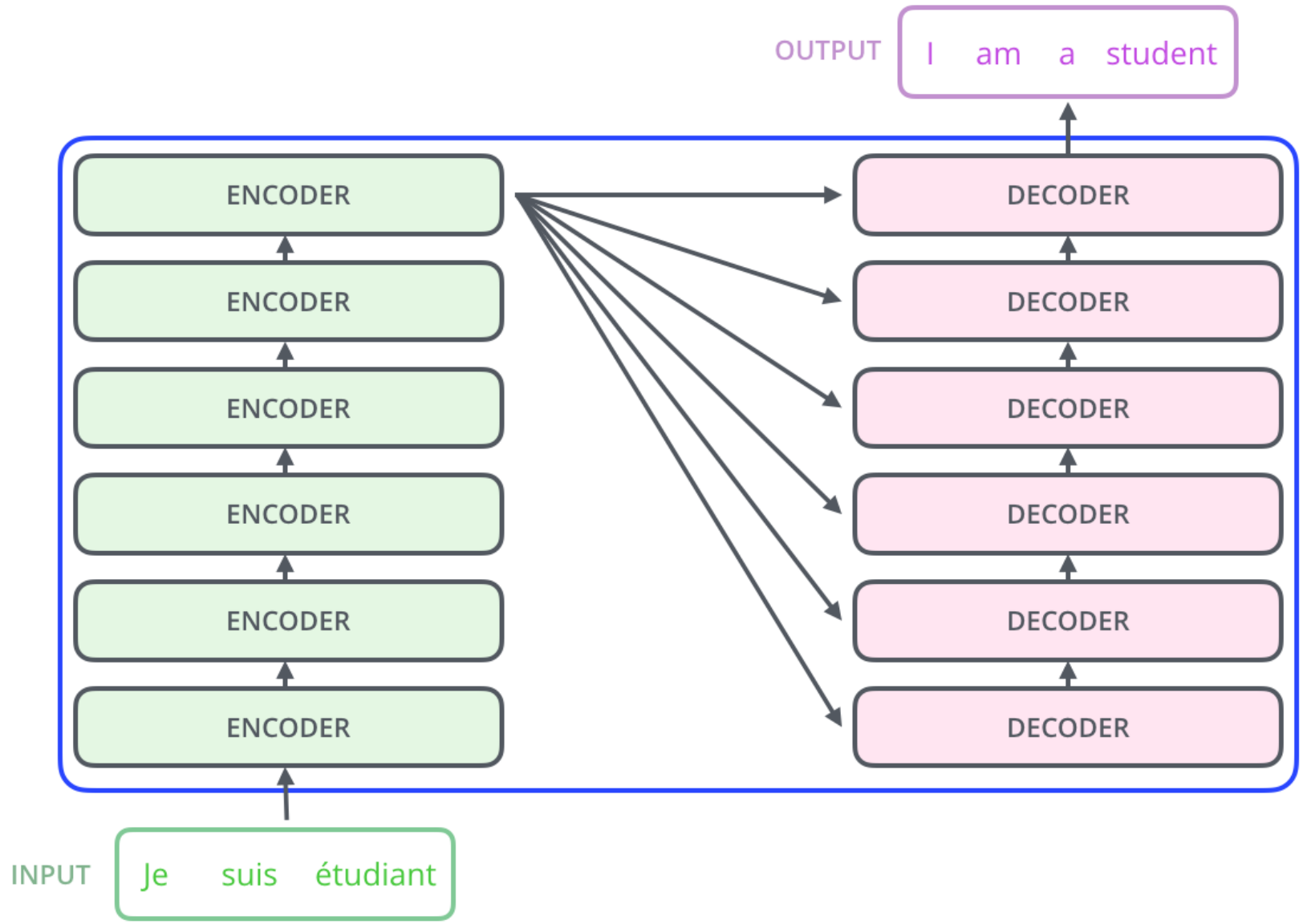
self-attention можно рассмотреть как граф + пулинг по релевантности
relevance-based pooling

нерекуррентная модель для последовательностей

глубокая модель с блоками внимания

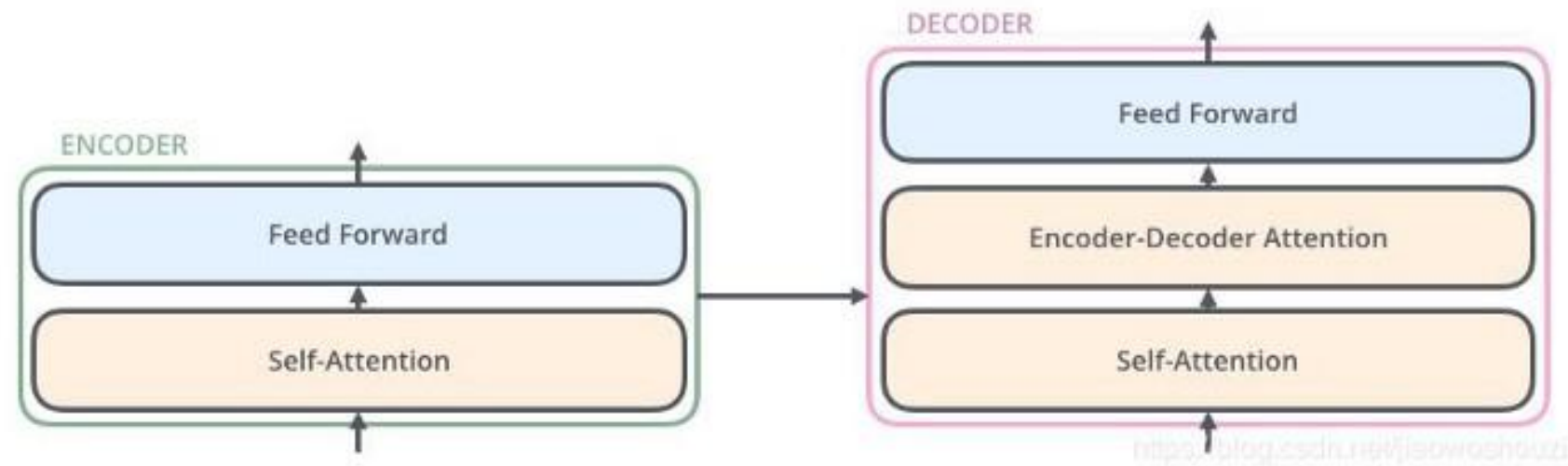
изначально для NMT
последний слой softmax + cross-entropy error

Transformer: общий вид



опять схема «кодировщик» – «декодировщик» (6 + 6 слоёв)
компоненты одинаковые по архитектуре, но веса разные

Transformer: Parallelized Attention



Encoder

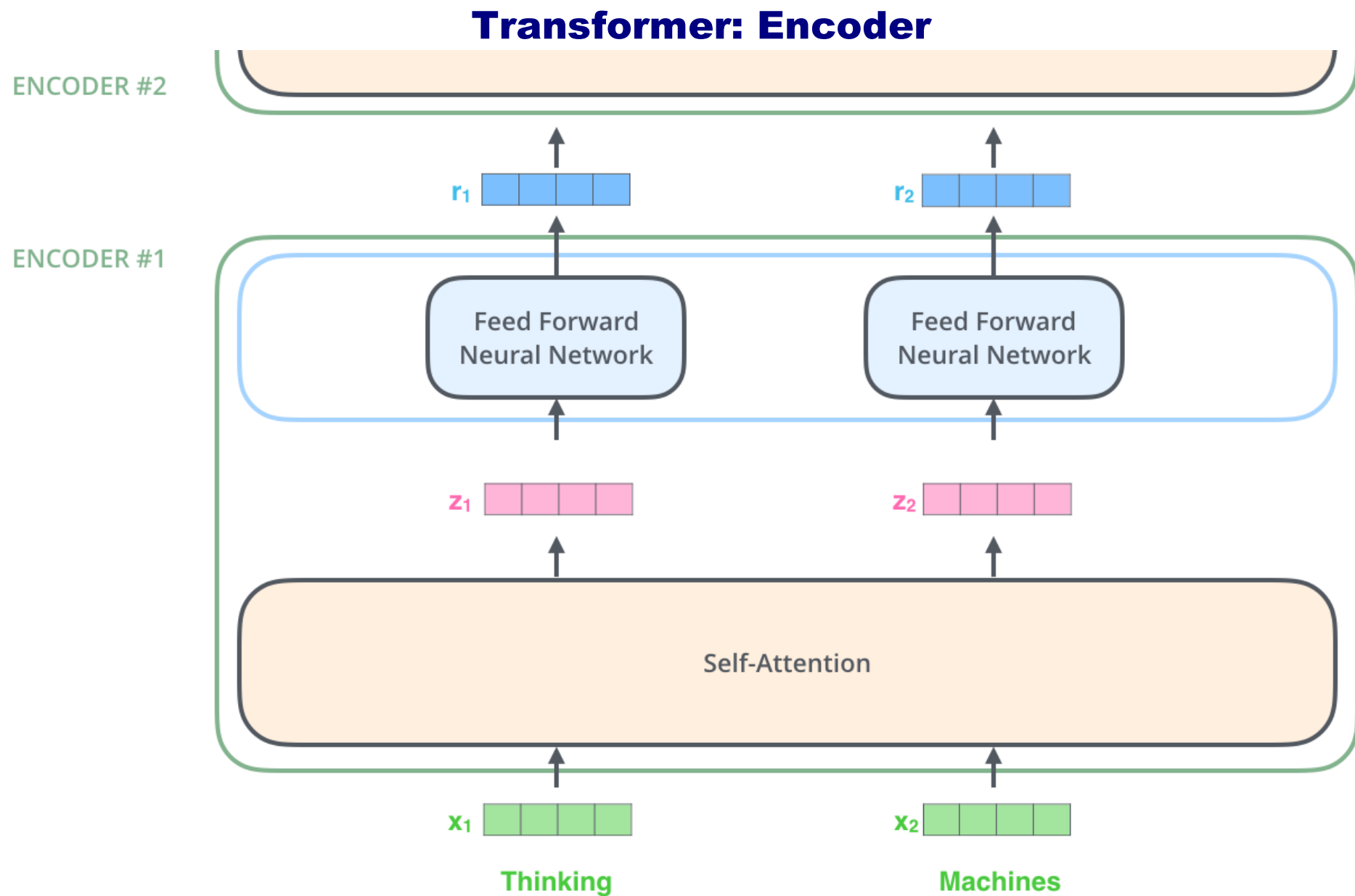
1й слой – self-attention

видеть семантику контекста – смотрим сразу на всё предложение

2й - сеть прямого распространения

улучшаем признаковое пространство

Encoder-Decoder Attention – смотрим на всё предложение на входе



слова подаём в виде представлений (embeddings) R^{512}

их преобразуем в векторы такой же размерности (представление следующего уровня)

Конценция «Self-Attention»

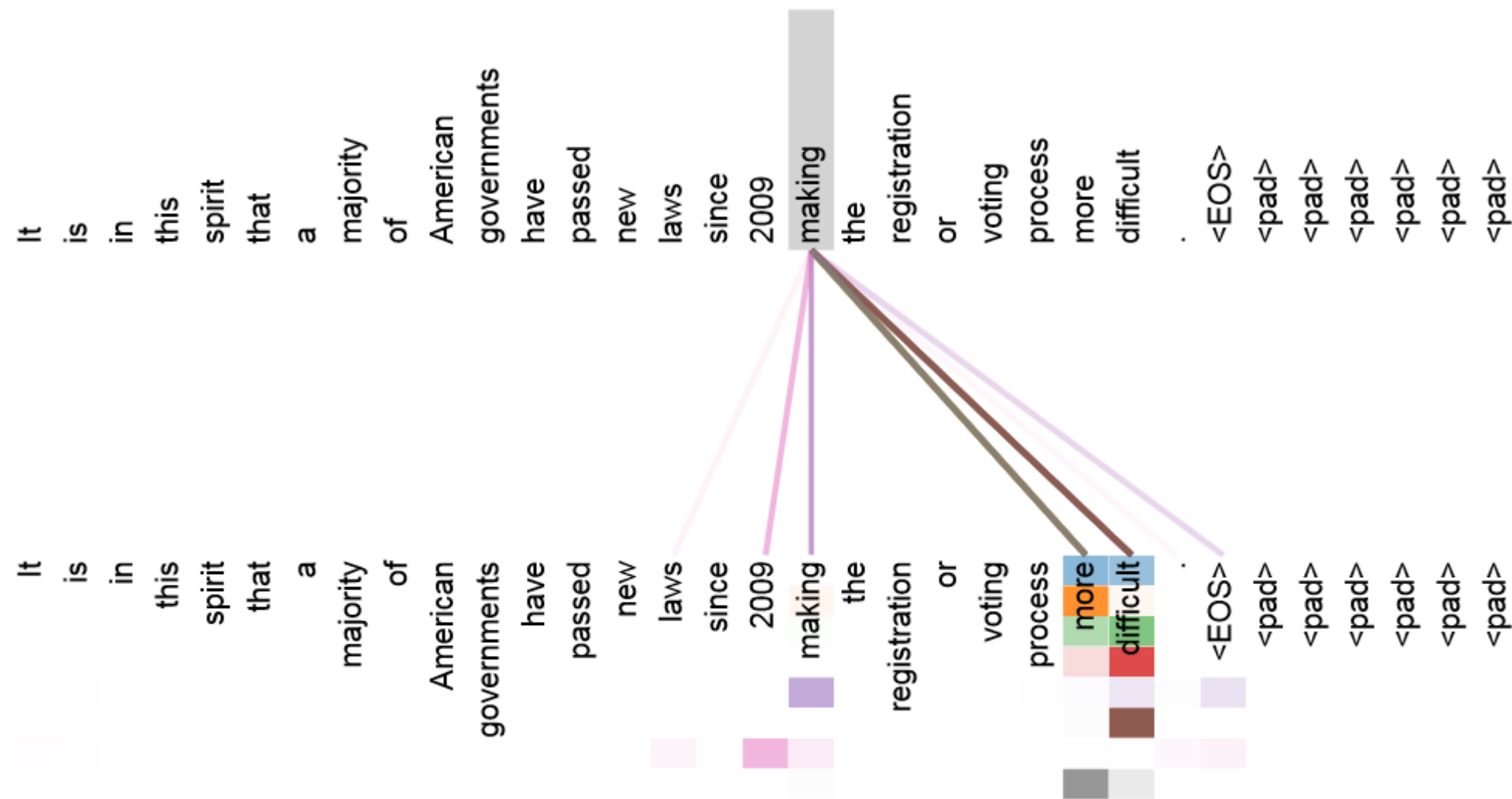
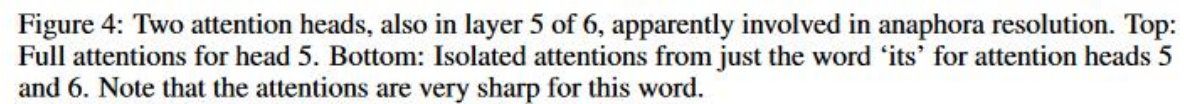
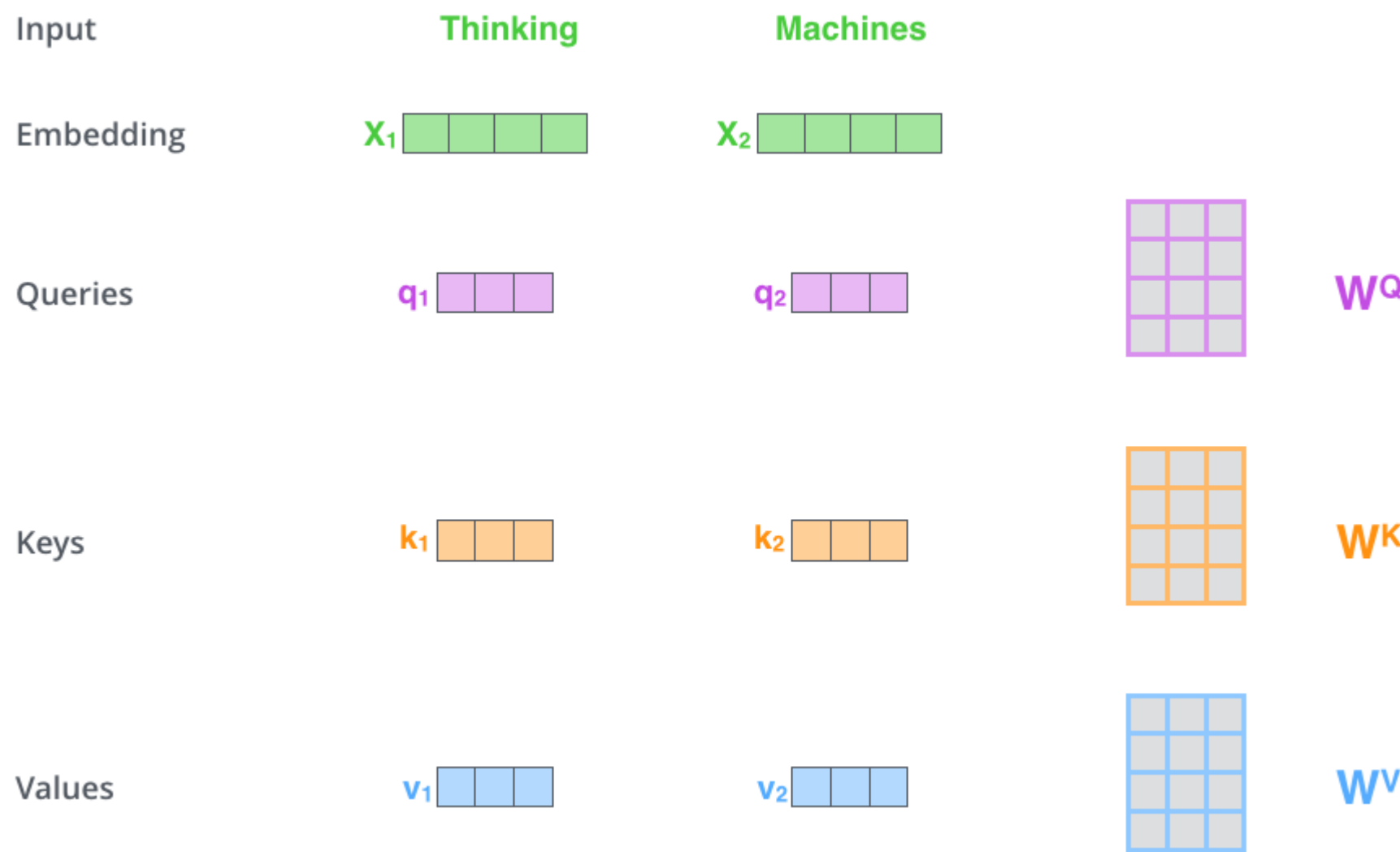


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

к чему относится «making»? Это должно влиять на представление следующего уровня

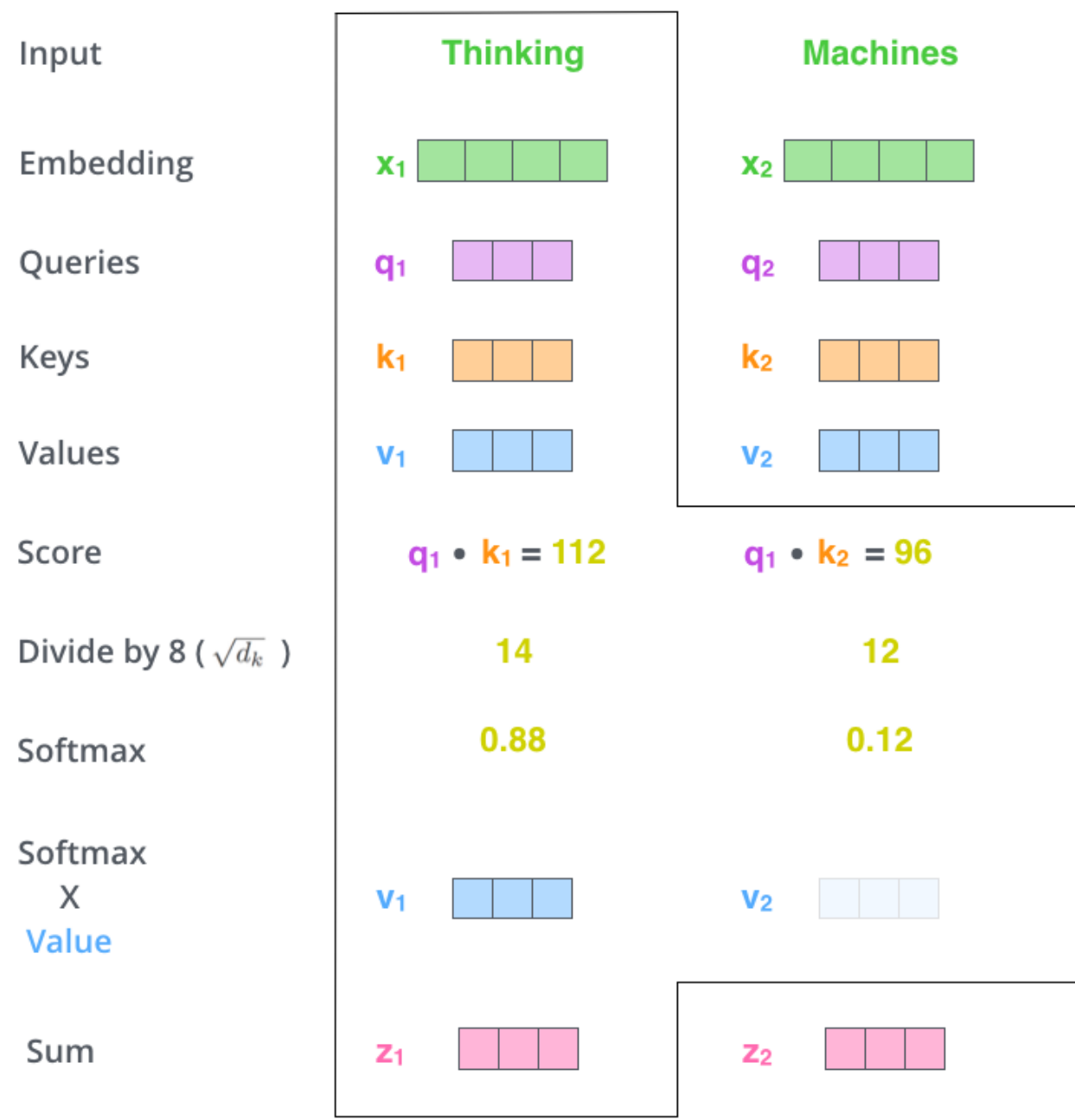


Реализация «Self-Attention»

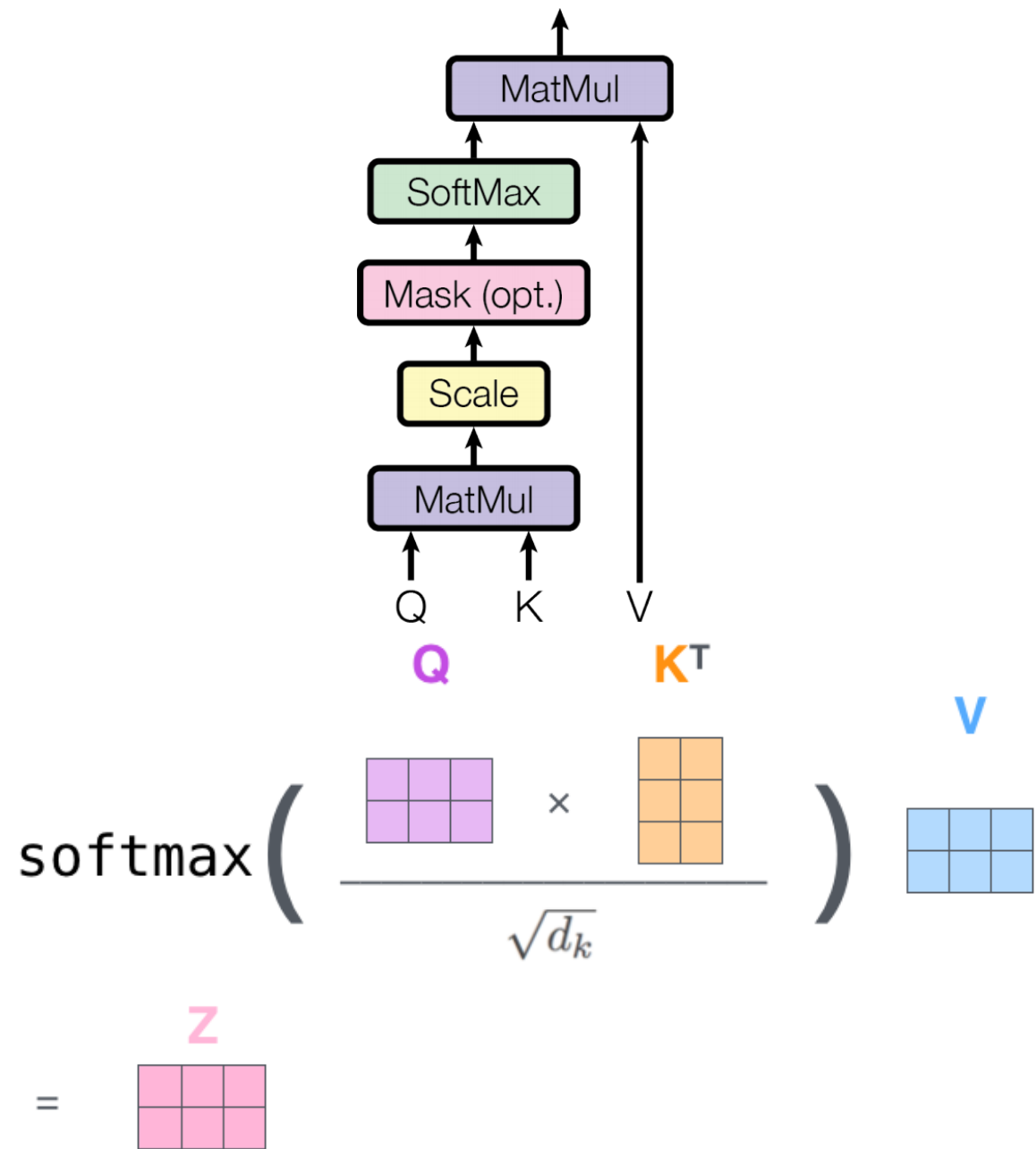
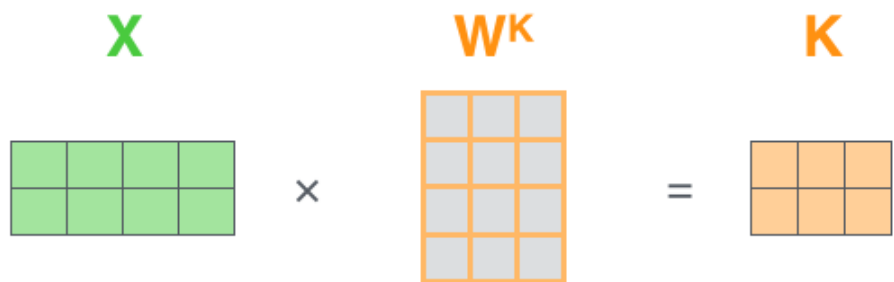


три 64-мерных вектора: Query, Key, Value – получаются из представлений (embeddings) умножением на матрицу (для каждого QKV – своя) – это обучаемый параметр

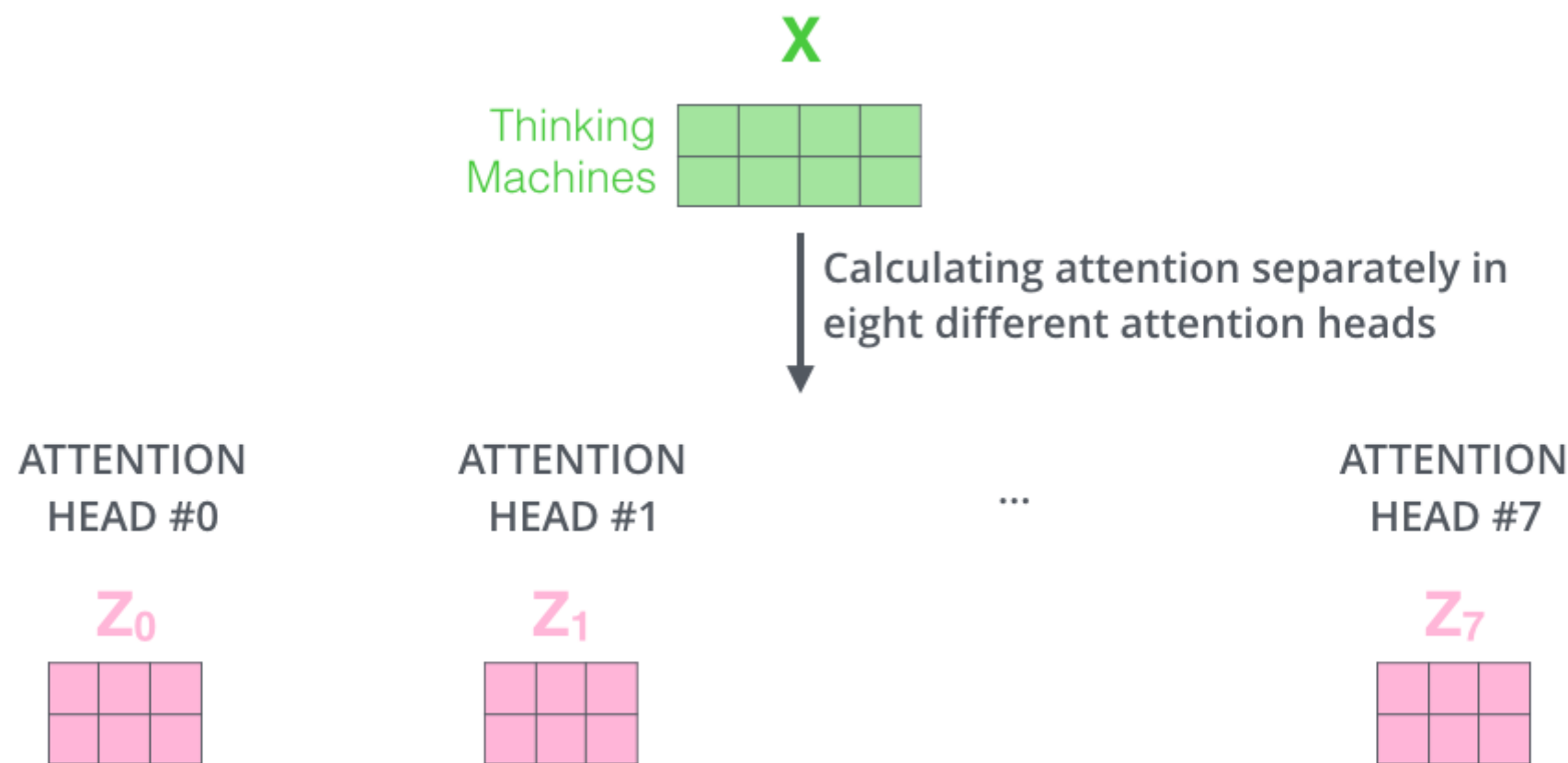
Реализация «Self-Attention»



Реализация «Self-Attention»: простая матричная



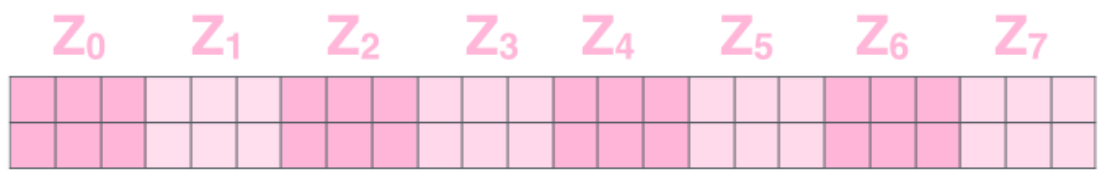
Multi-Headed Attention (несколько головок)



каждая головка обозревает свой аспект внимания (что это, что делает и т.п.)
как теперь превратить результаты разных головок в вектор нужной размерности?

Multi-Headed Attention (несколько головок)

1) Concatenate all the attention heads

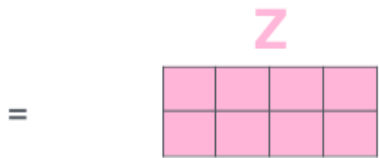


2) Multiply with a weight matrix W^O that was trained jointly with the model

\times



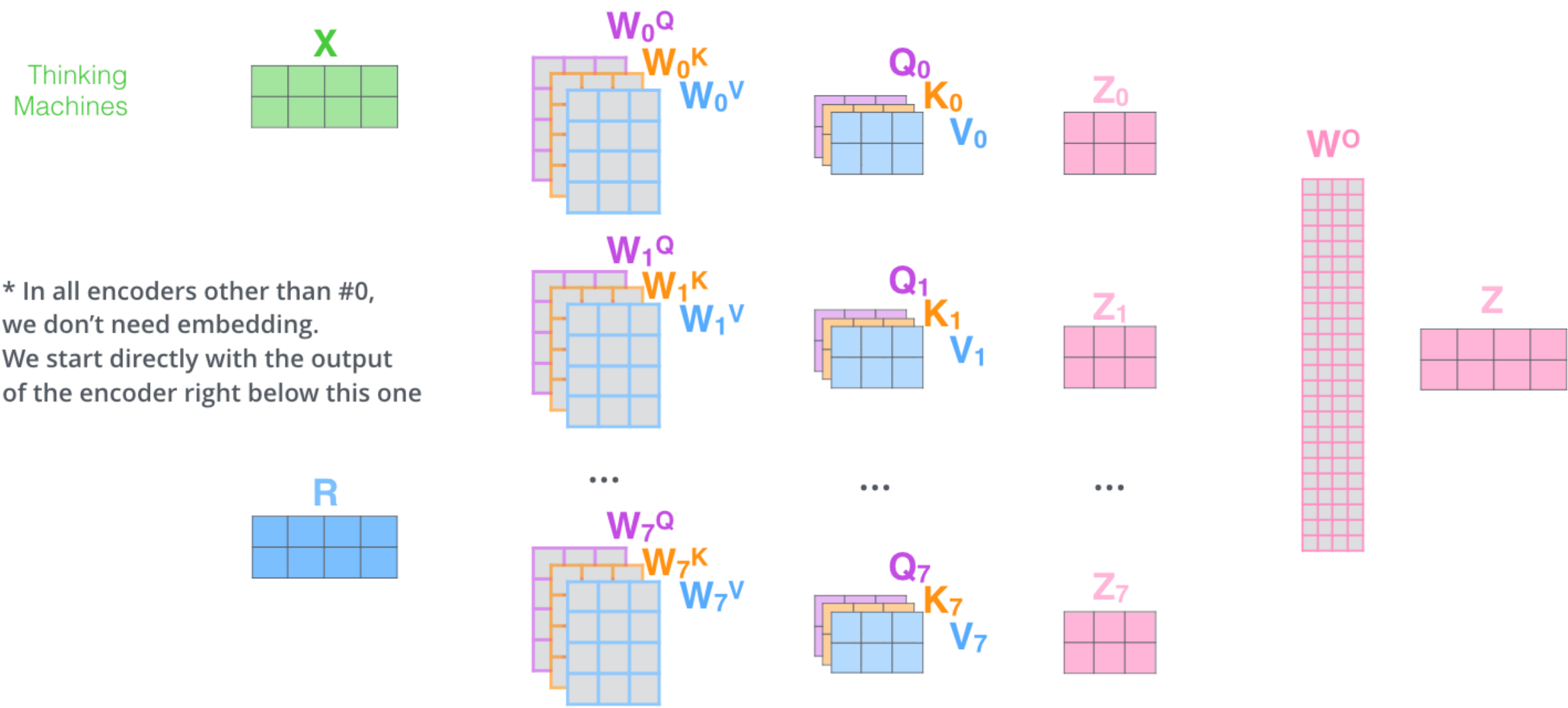
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN



конкатенация и умножение на ещё одну матрицу весов

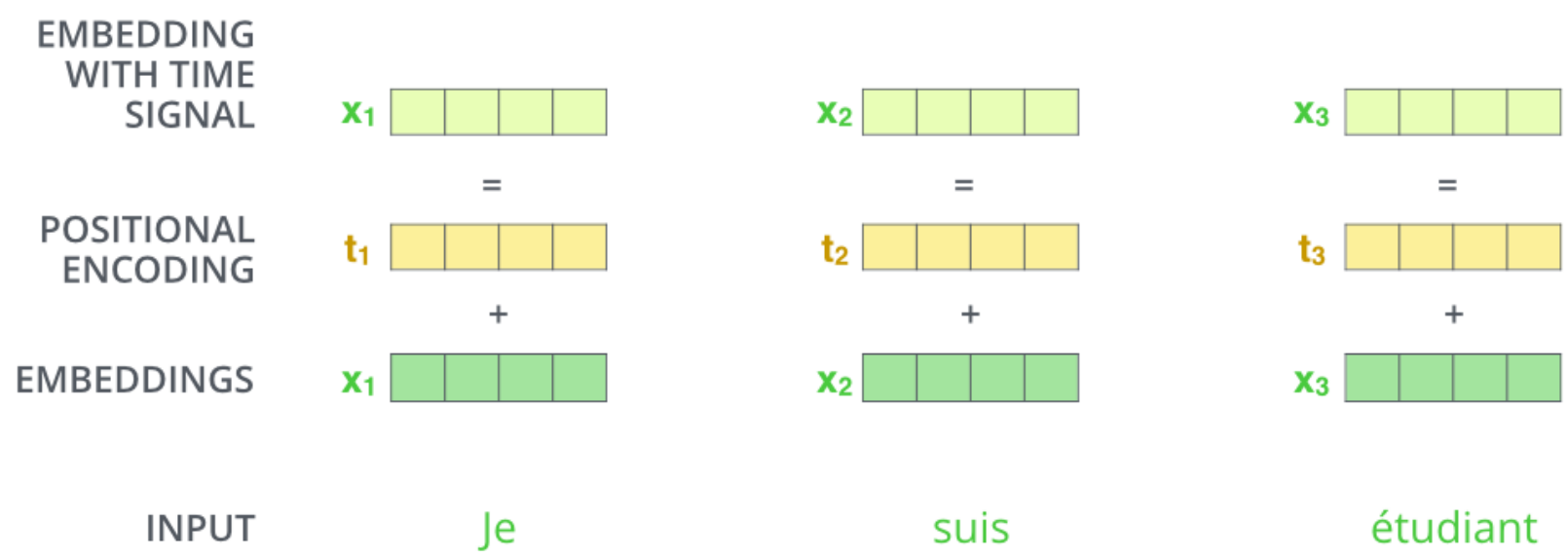
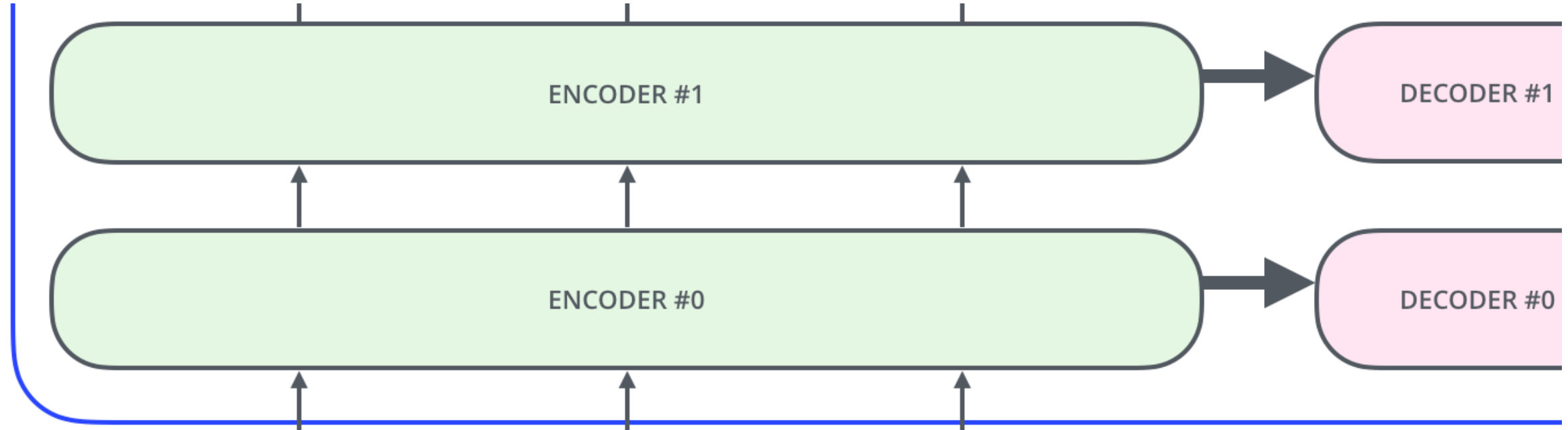
Multi-Headed Attention (несколько головок) – саммари

- 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



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

Кодирование позиции слова: Positional Encoding



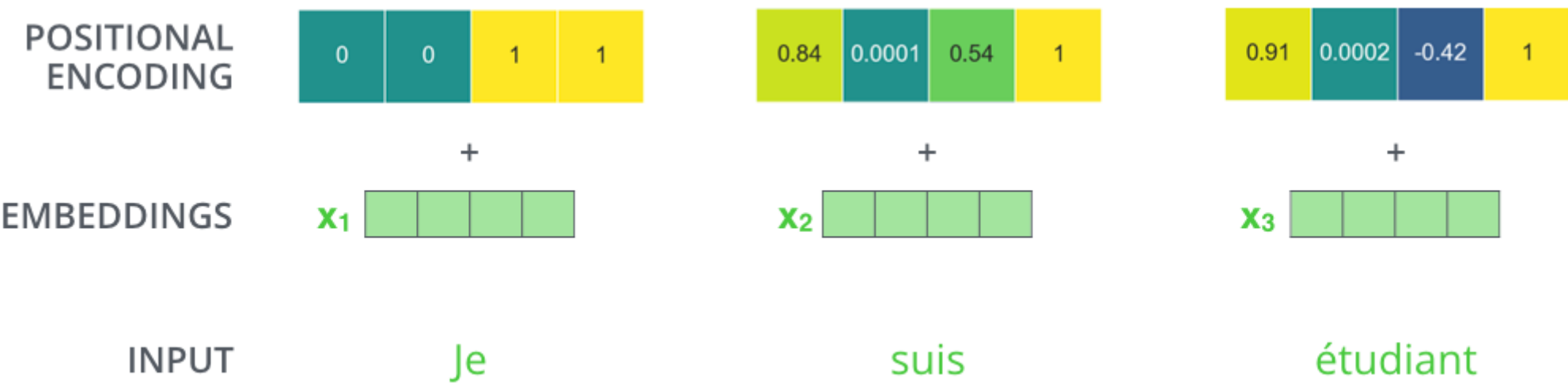
Кодирование позиции слова: Positional Encoding

Проблема: пока наш метод не зависит от перестановки слов,
а надо учитывать порядок \Rightarrow будем кодировать позицию

Но надо, чтобы метод не зависел от длины входного предложения

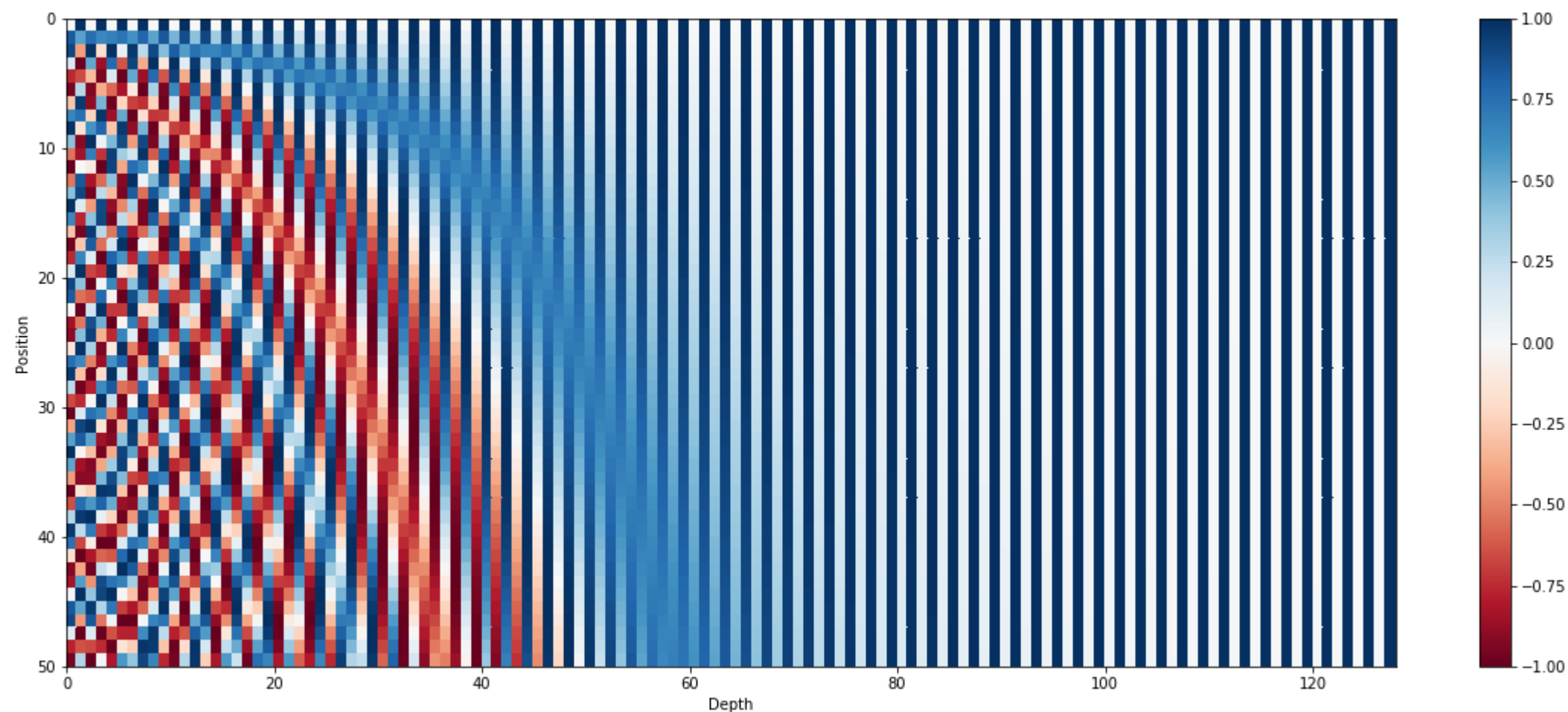
номеру позиции t соответствует d -мерный вектор, позиции которого

$$(2i, 2i + 1) \rightarrow \left(\sin\left(\frac{2t}{10000^{2i/d}}\right), \cos\left(\frac{2t}{10000^{2i/d}}\right) \right)$$



картинка для размерности представления = 4

Кодирование позиции слова: Positional Encoding



128-мерное кодирование 50 позиций

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

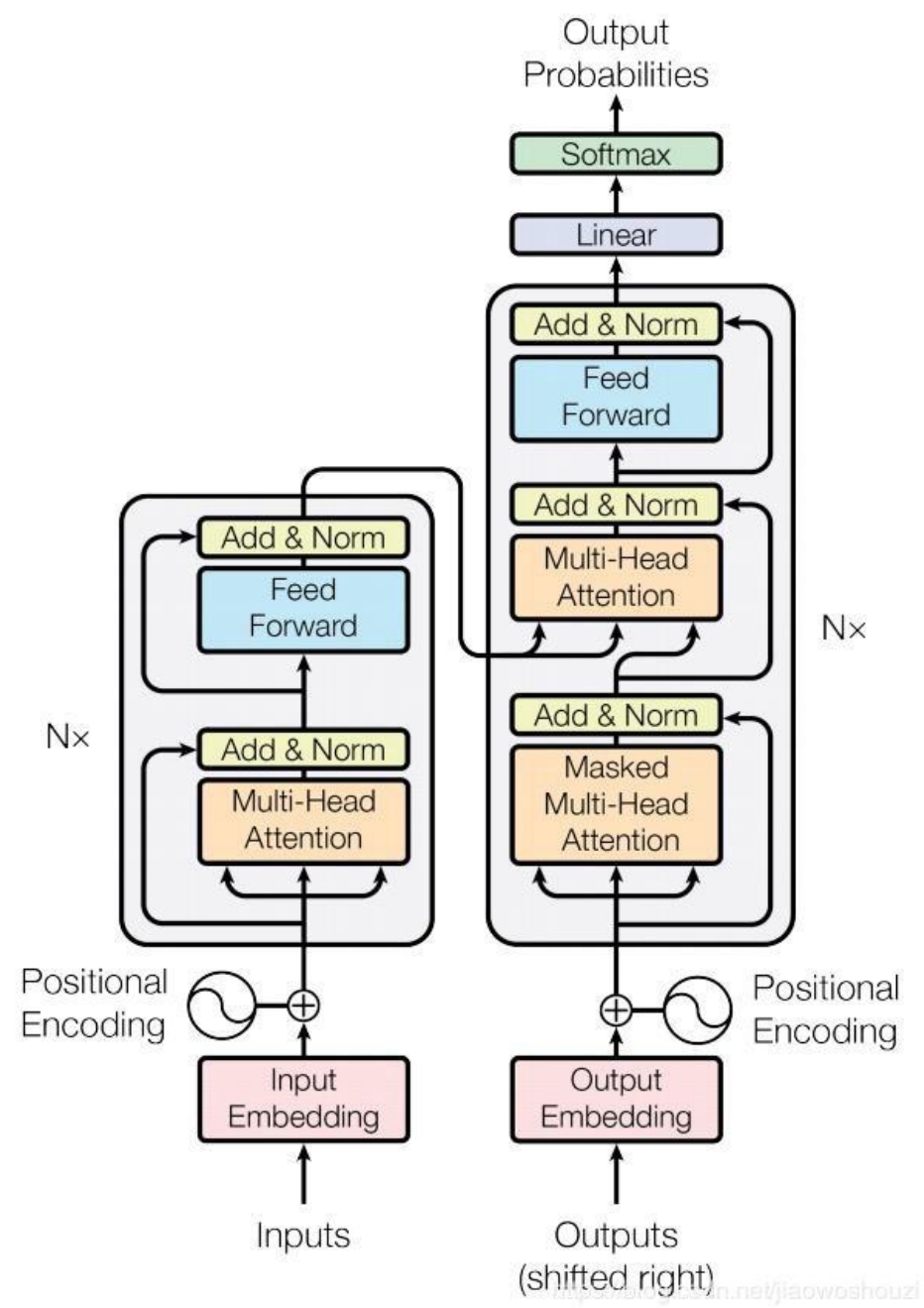
Кодирование позиции слова: Positional Encoding

```
class PositionalEncoding(nn.Module):
    "Implement the PE function."
    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

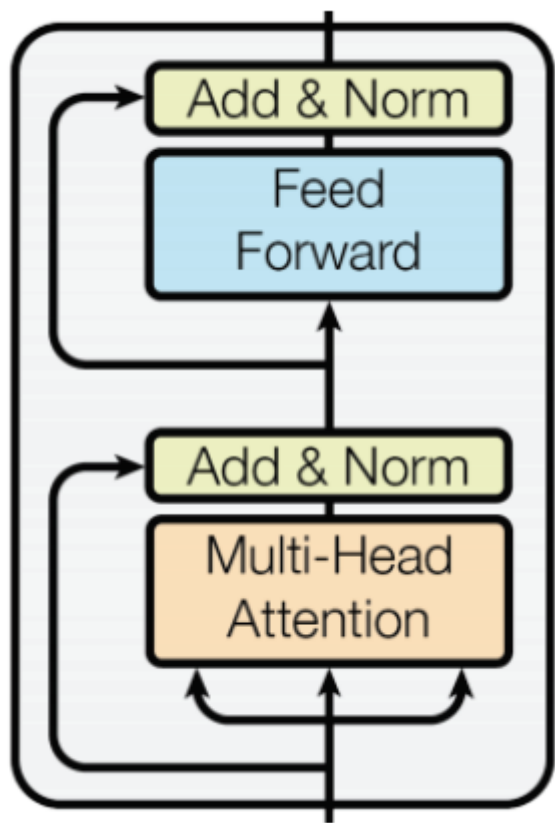
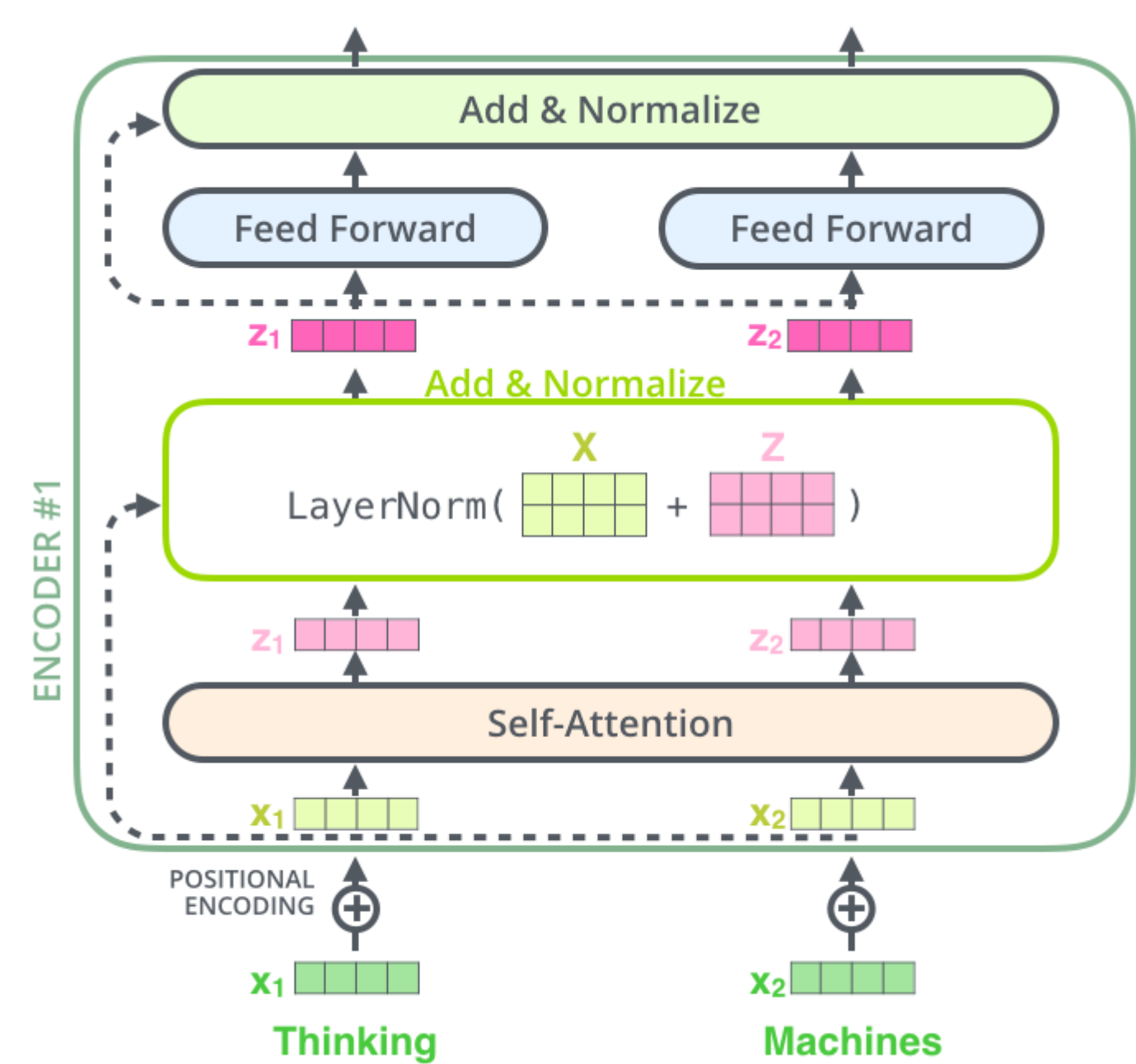
        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) *
                              - (math.log(10000.0) / d_model))
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        x = x + Variable(self.pe[:, :x.size(1)], requires_grad=False)
        return self.dropout(x)
```

Transformer: Parallelized Attention



Transformer: residual connection and layer normalization



Transformer block

Multihead attention
2-layer FFNN (ReLU)
Residual connections
LayerNorm

Transformer: residual connection and layer normalization

**Прокидывание связей – всё просто: размерности совпадают,
можно сложить**

$$X_A = \text{LayerNorm}(\text{MultiheadSelfAttention}(X)) + X$$

$$X_B = \text{LayerNorm}(\text{PositionFFN}(X_A)) + X_A$$

```
class SublayerConnection(nn.Module):  
    """  
    A residual connection + layer norm. For code simplicity the norm is first  
    """  
    def __init__(self, size, dropout):  
        super(SublayerConnection, self).__init__()  
        self.norm = LayerNorm(size)  
        self.dropout = nn.Dropout(dropout)  
  
    def forward(self, x, sublayer):  
        "Apply residual connection to any sublayer with the same size."  
        return x + self.dropout(sublayer(self.norm(x)))
```

Transformer: Layer Norm

$$x = (x_1, \dots, x_d), \mu = \frac{1}{d} \sum_{i=1}^d x_i, \sigma^2 = \frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2$$

$$\text{LN}(x) = \gamma \frac{x - \mu}{\sigma} + \beta$$

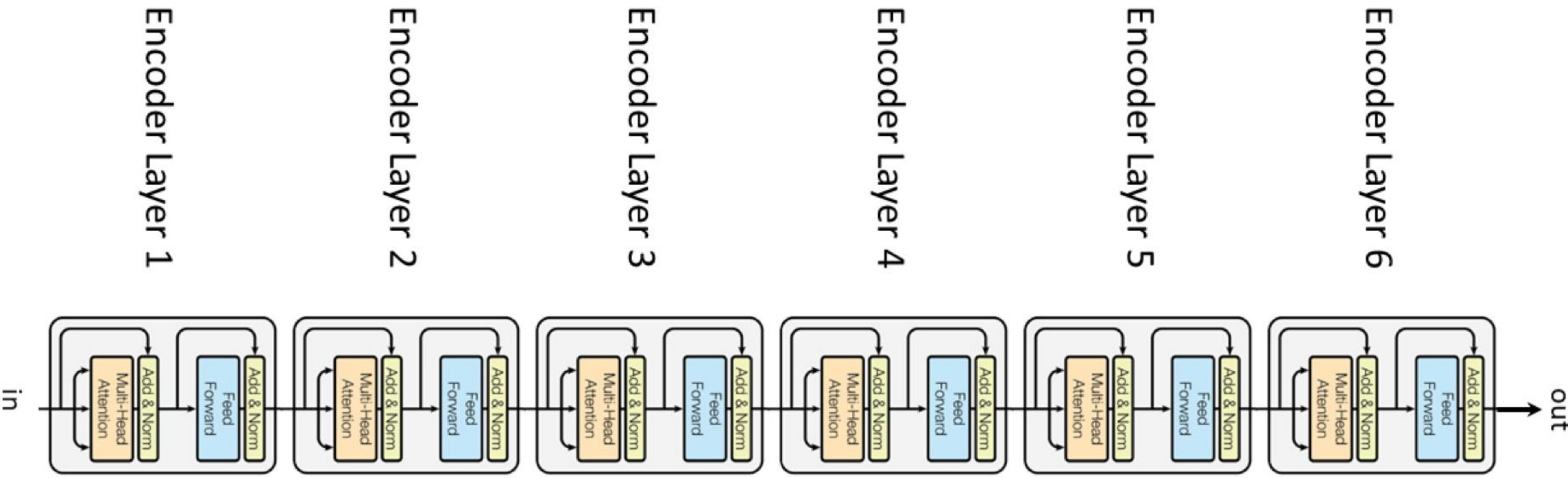
```
class LayerNorm(nn.Module):  
    "Construct a layernorm module (See citation for details)."  
    def __init__(self, features, eps=1e-6):  
        super(LayerNorm, self).__init__()  
        self.a_2 = nn.Parameter(torch.ones(features))  
        self.b_2 = nn.Parameter(torch.zeros(features))  
        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  
  
http://nlp.seas.harvard.edu/2018/04/03/attention.html
```

Transformer: FFNN

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

```
class PositionwiseFeedForward(nn.Module):  
    "Implements FFN equation."  
    def __init__(self, d_model, d_ff, dropout=0.1):  
        super(PositionwiseFeedForward, self).__init__()  
        self.w_1 = nn.Linear(d_model, d_ff)  
        self.w_2 = nn.Linear(d_ff, d_model)  
        self.dropout = nn.Dropout(dropout)  
  
    def forward(self, x):  
        return self.w_2(self.dropout(F.relu(self.w_1(x))))
```

Transformer: Encoder



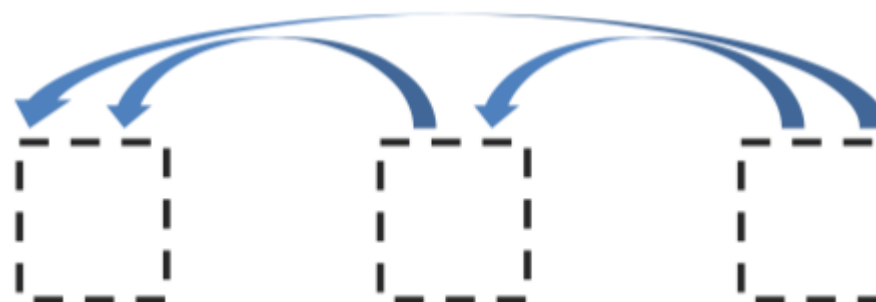
Transformer: виды внимания



Encoder Self-Attention



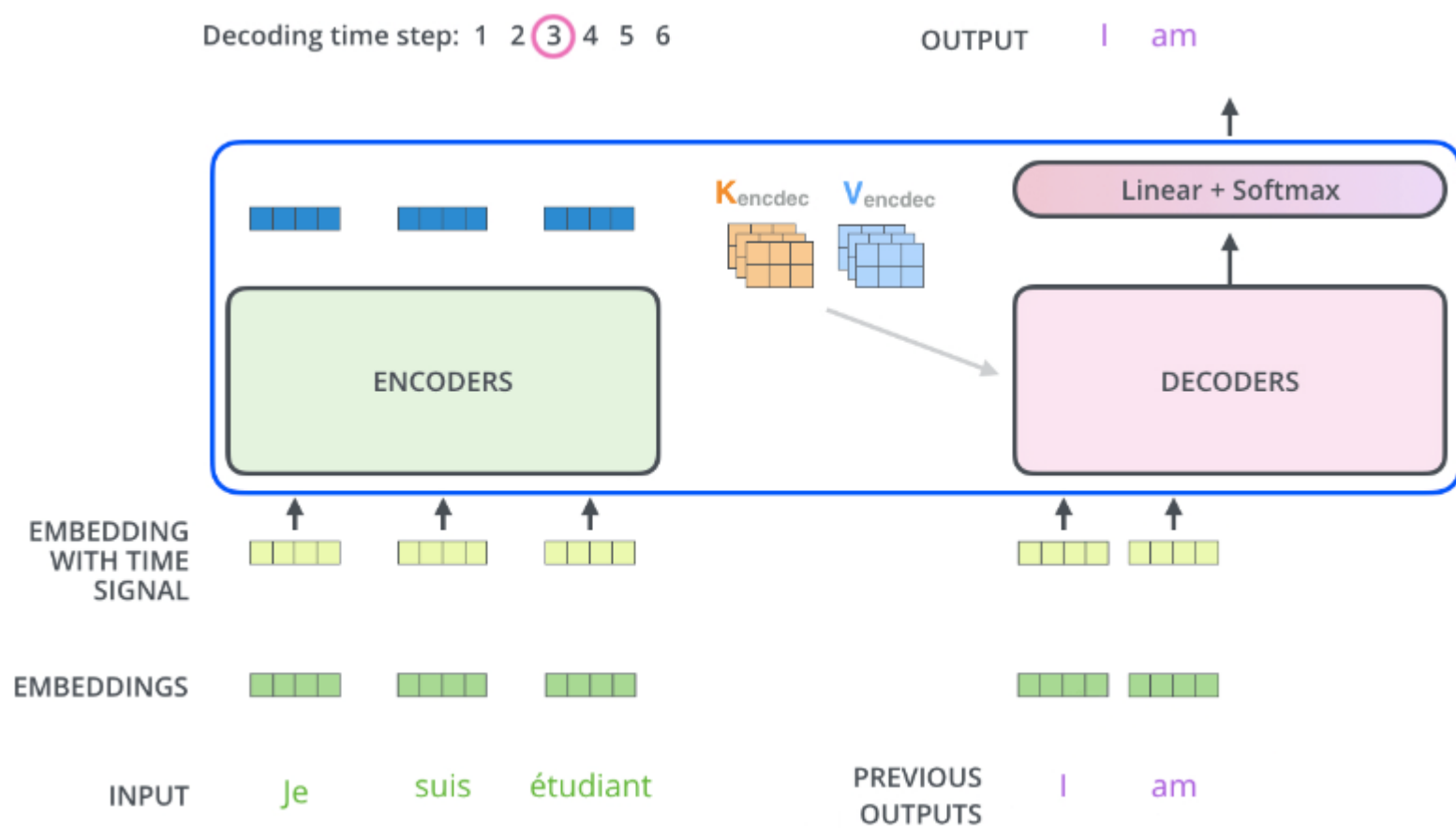
Encoder-Decoder Attention



Masked Decoder Self-Attention

Transformer: Decoder

Тонкость: не знаем длину ответа
⇒ **будем его получать последовательно**



Transformer: Decoder

Другая тонкость: при обучении нельзя видеть «будущее ответа» \Rightarrow маскирование
на схеме есть «masked multi-head attention»

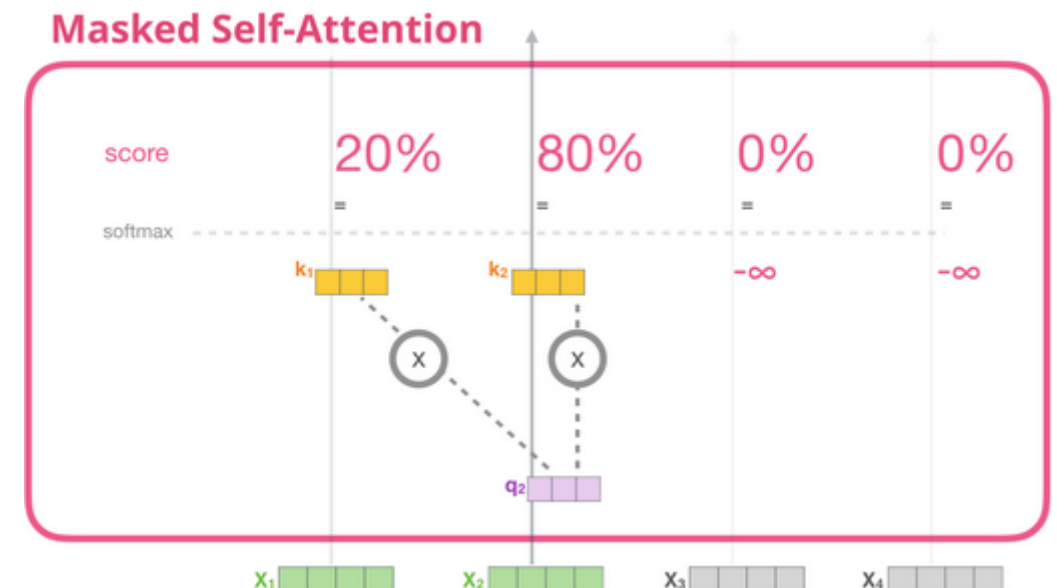
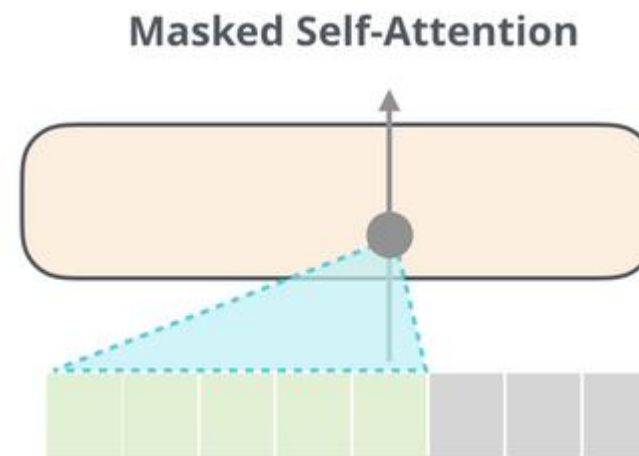
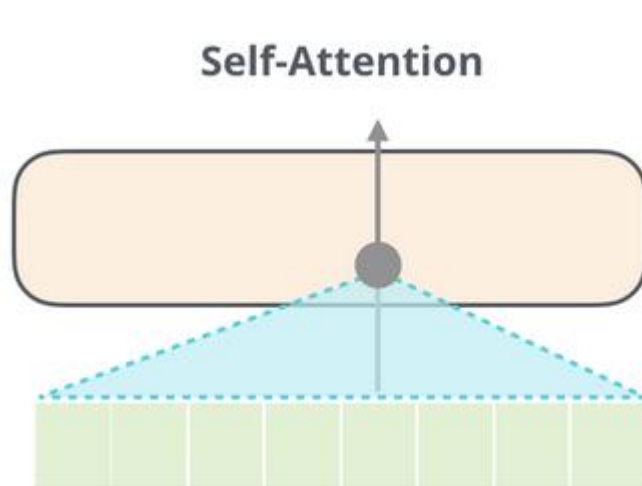
padding mask (in all the scaled dot-product attention)

~ добавляем 0 в конец коротких предложений

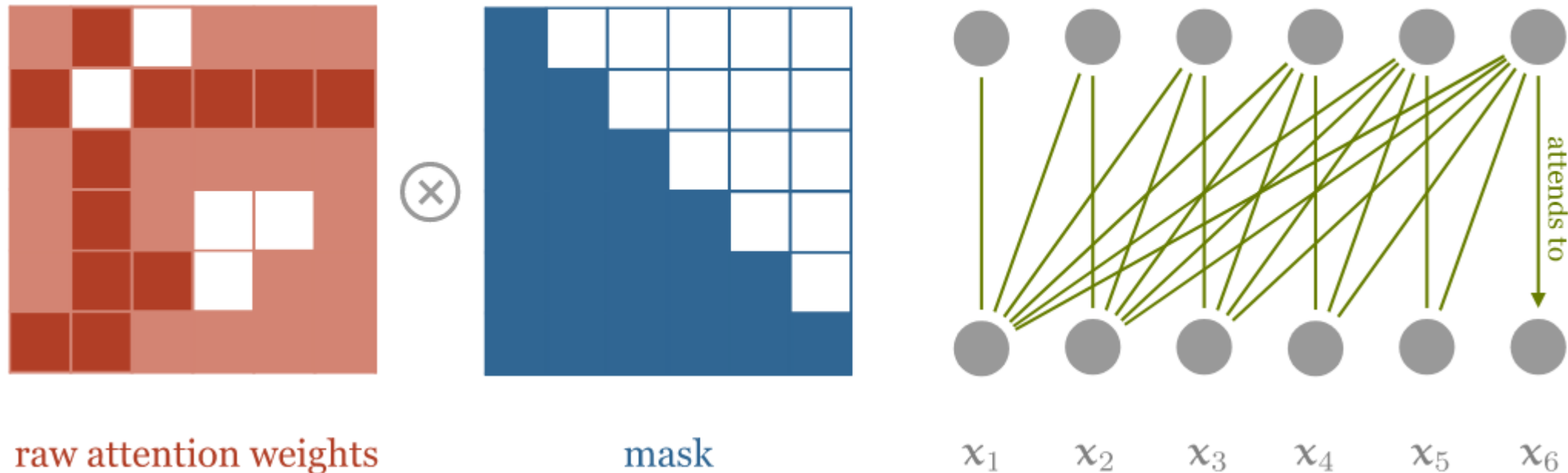
sequence mask (in the decoder's self-attention)

чтобы не видел информацию из будущего

ВЫВОДИТ ОДНО СЛОВО ЗА ТАКТ



Transformer: маскирование



raw attention weights

mask

```
dot = torch.bmm(queries, keys.transpose(1, 2))
indices = torch.triu_indices(t, t, offset=1)
dot[:, indices[0], indices[1]] = float('-inf')
dot = F.softmax(dot, dim=2)
```

<http://peterbloem.nl/blog/transformers>

Transformer: Output layer

как на выходе получить одно слово:

Which word in our vocabulary
is associated with this index?

Get the index of the cell
with the highest value
(argmax)

am

5

log_probs



Softmax

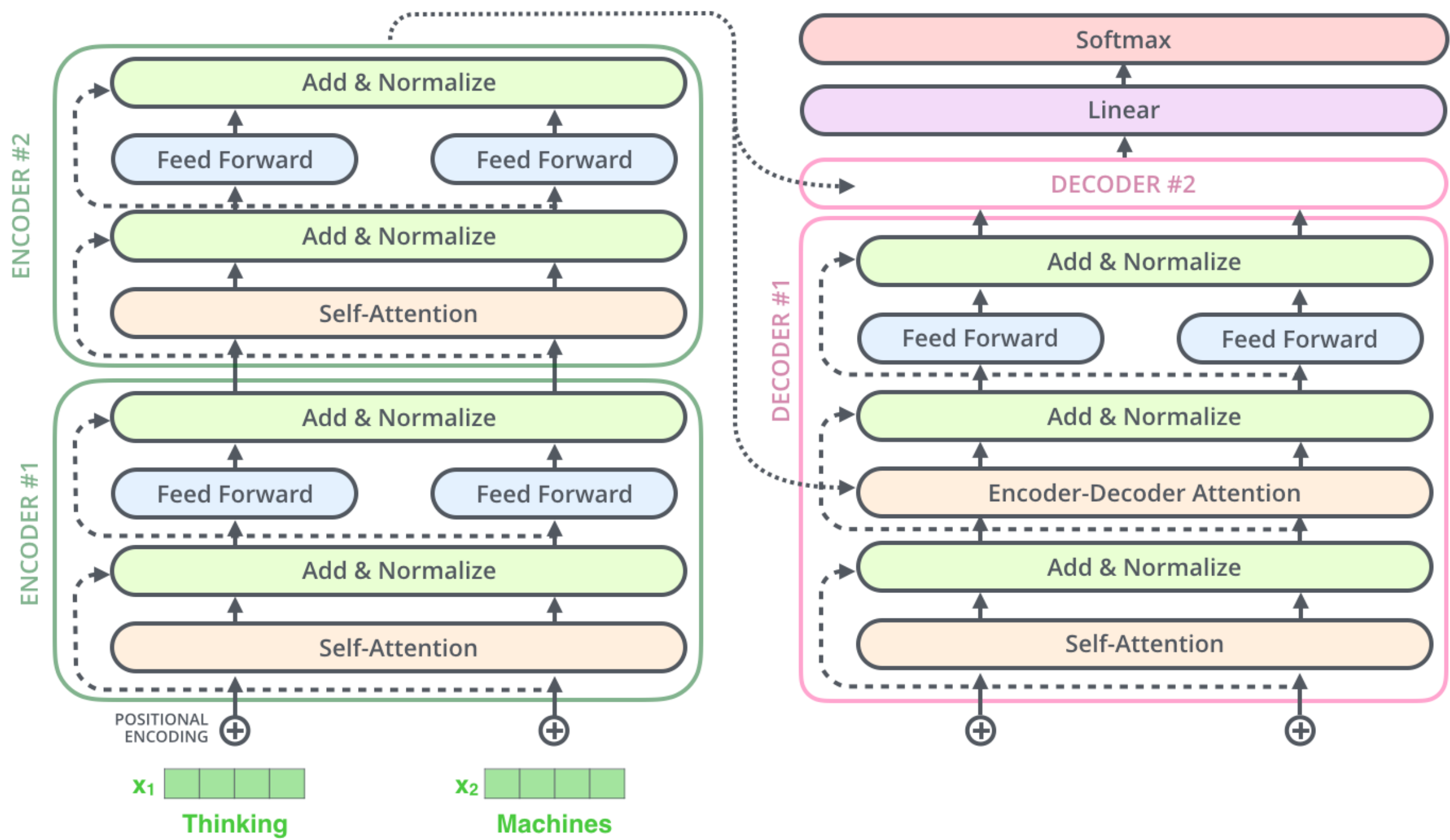
logits



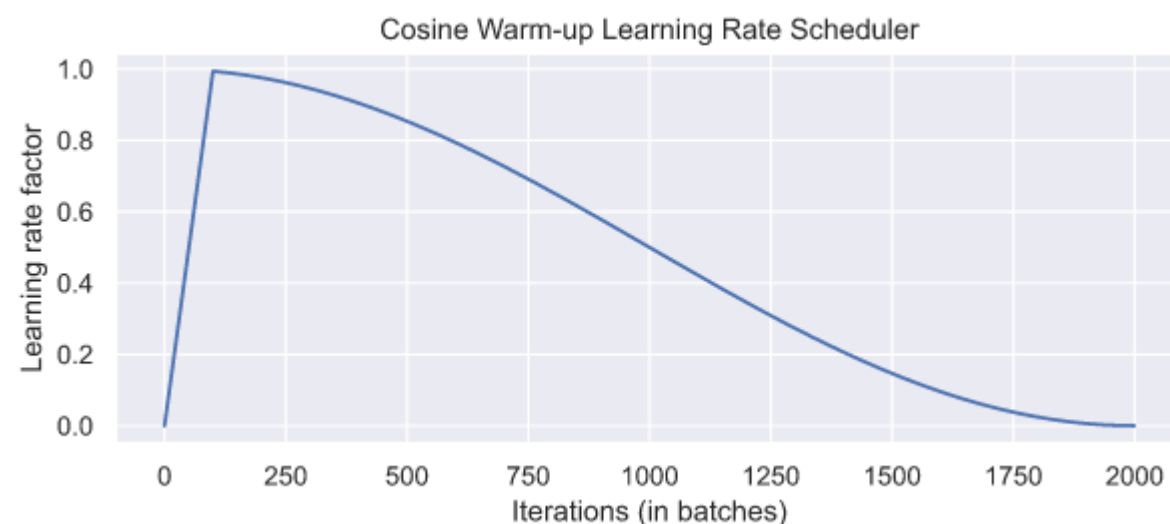
Linear

Decoder stack output





Обучение трансформера: Learning rate warm-up



**трансформеры обучают
с такой программой изменения LR**

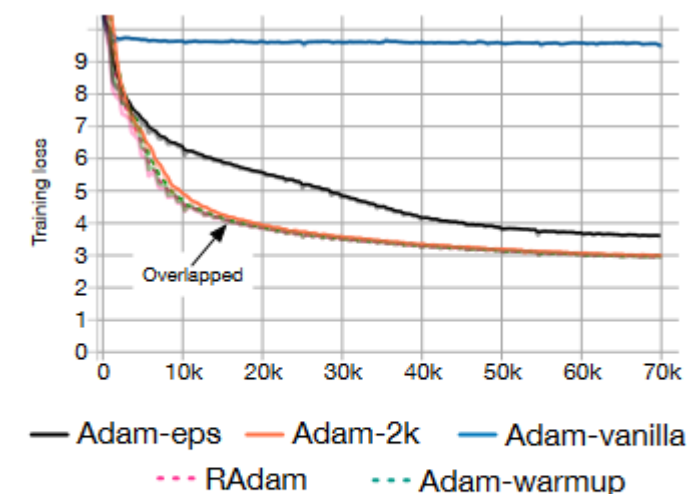


Figure 1: Training loss v.s. # of iterations of Transformers on the De-En IWSLT'14 dataset.

Рис. из <https://arxiv.org/pdf/1908.03265.pdf>

**В Adam используется bias correction factors –
он увеличивает дисперсию на первых итерациях**

Layer Normalization – может увеличивать норму градиента

https://huggingface.co/transformers/main_classes/optimizer_schedules.html?highlight=cosine#transformers.get_cosine_schedule_with_warmup

Особенности обучения трансформера

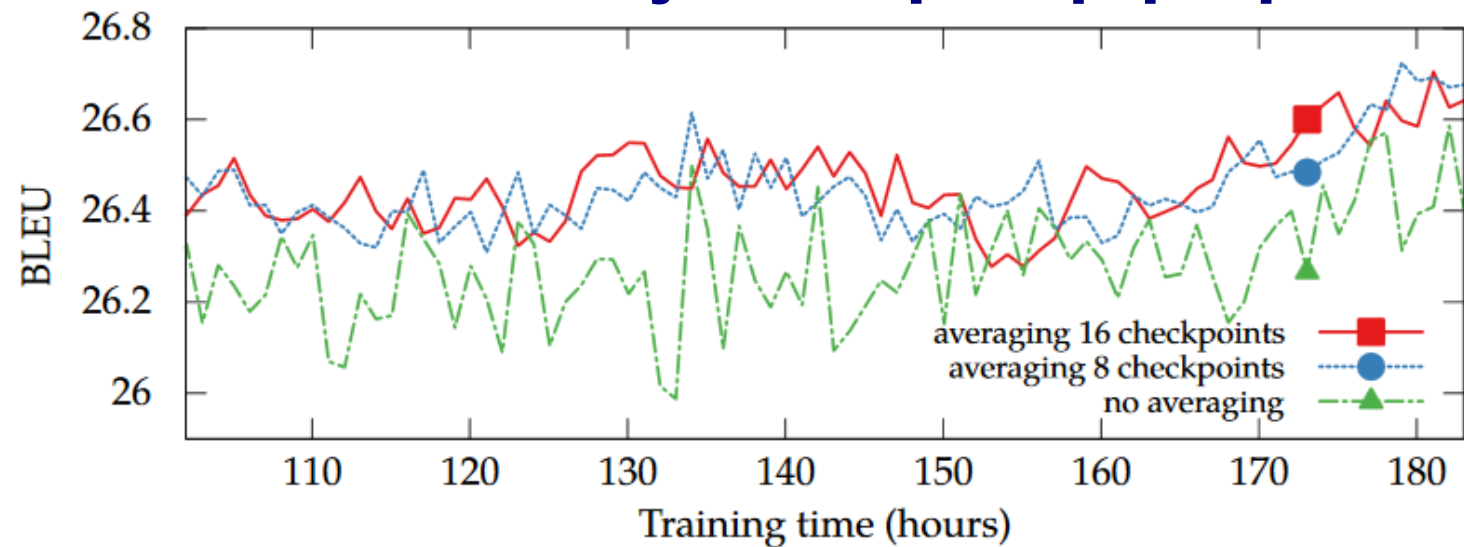


Figure 10: Effect of checkpoint averaging. All trained on 6 GPUs.

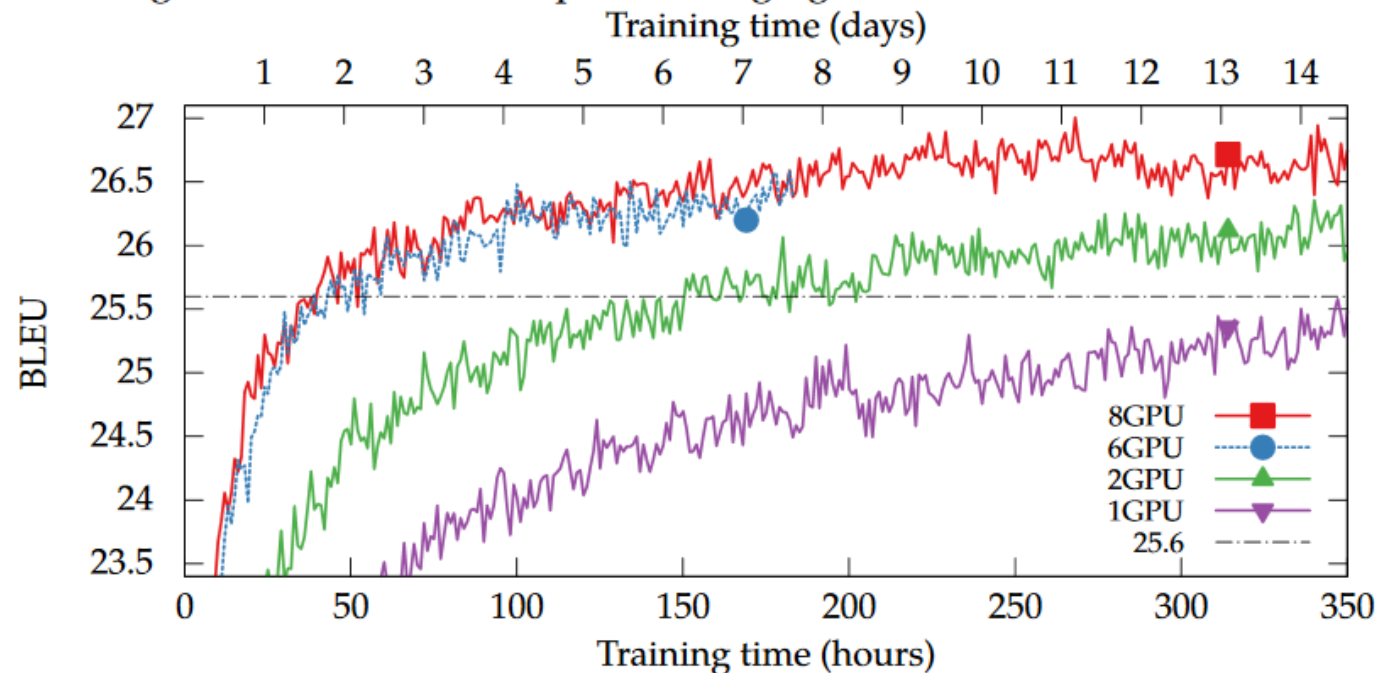


Figure 9: Effect of the number of GPUs. BLEU=25.6 is marked with a black line.

Особенности обучения трансформера

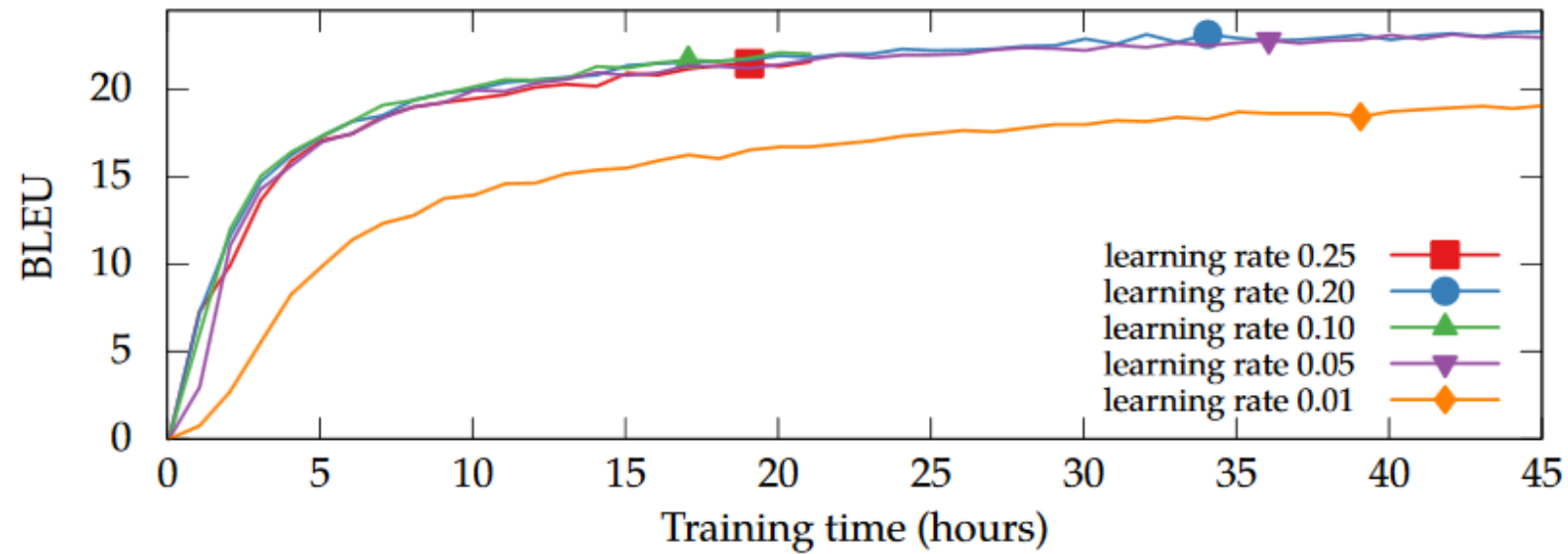


Figure 7: Effect of the learning rate on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and warmup steps (16k).

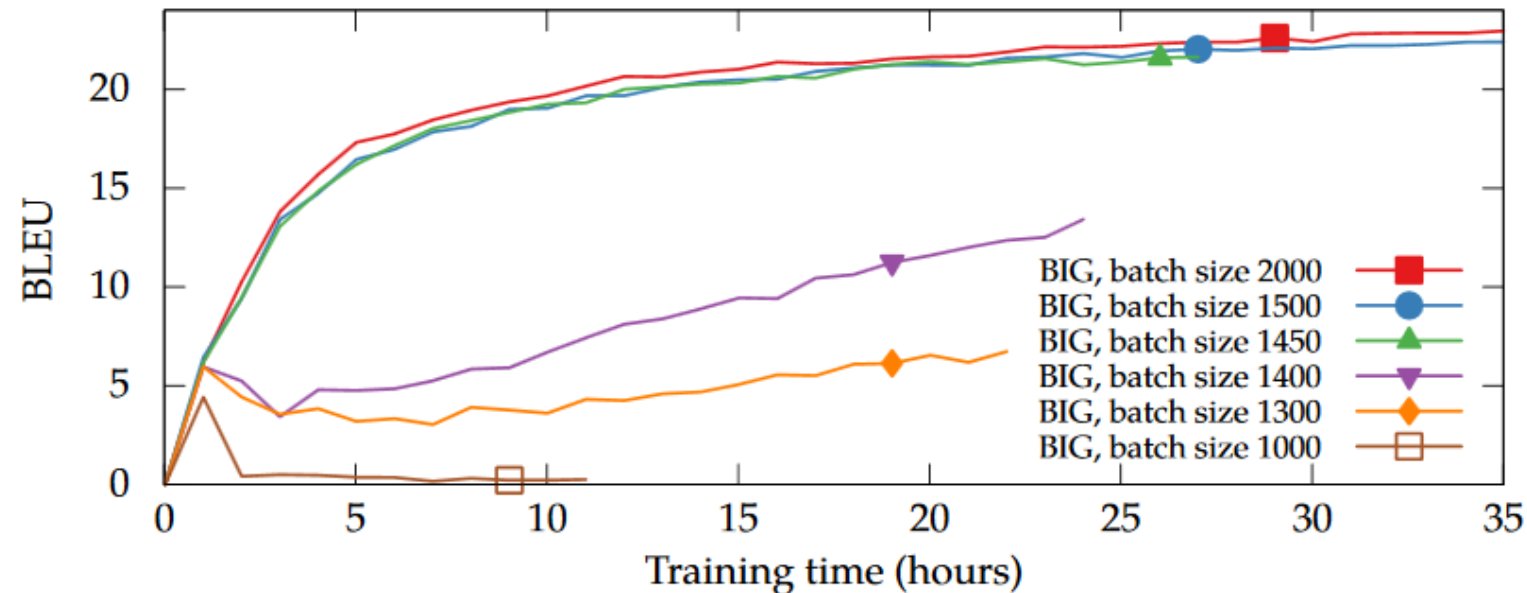


Figure 6: Effect of the batch size with the BIG model. All trained on a single GPU.

Особенности обучения трансформера

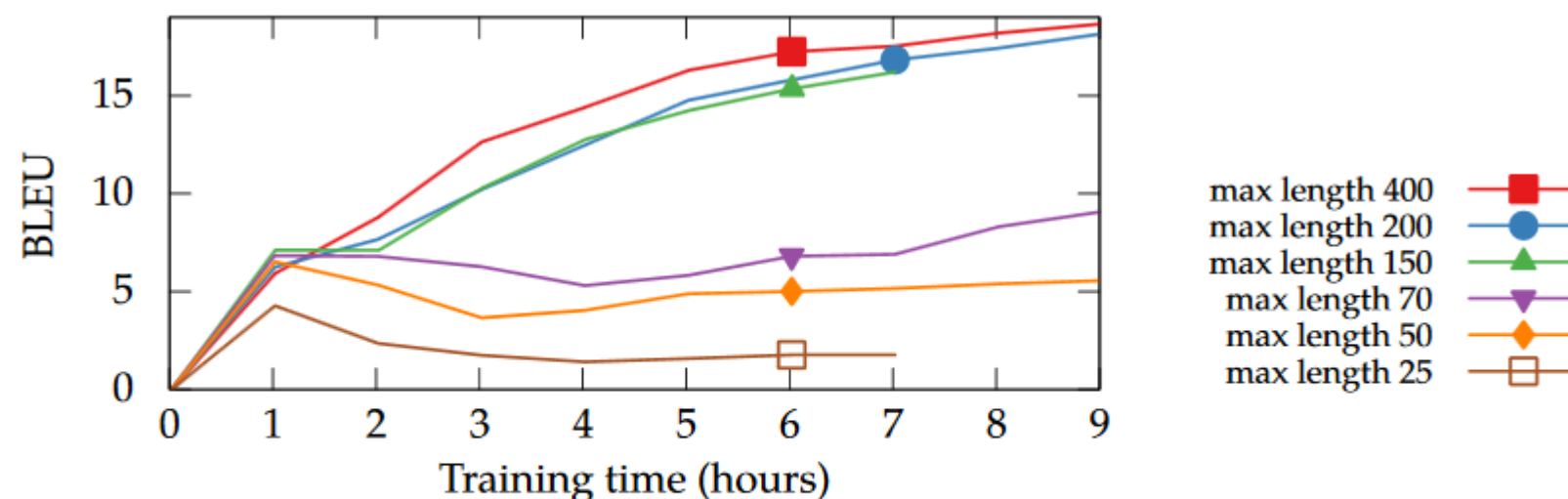


Figure 4: Effect of restricting the training data to various max_length values. All trained on a single GPU with the BIG model and batch_size=1500. An experiment without any max_length is not shown, but it has the same curve as max_length=400.

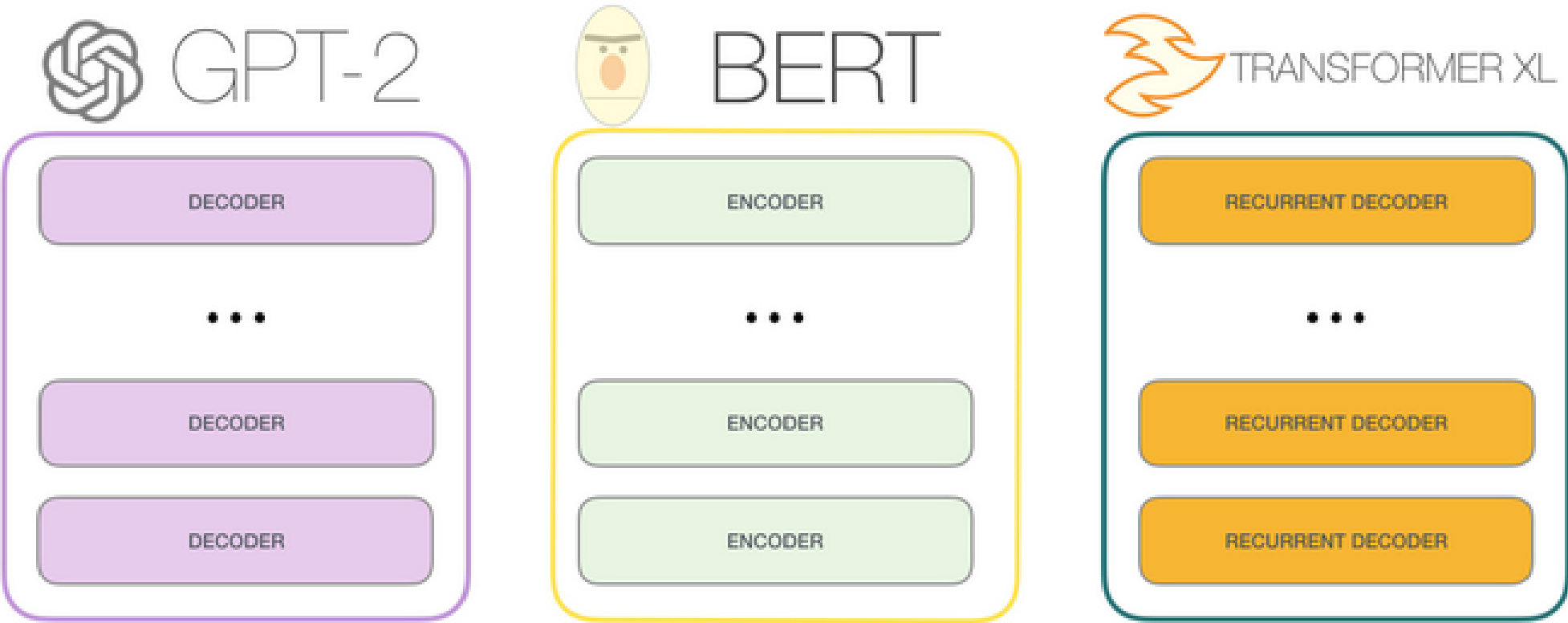
Martin Popel, Ondřej Bojar «Training Tips for the Transformer Model» // <https://arxiv.org/pdf/1804.00247.pdf>

визуализация обучения:

<https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>

Использование трансформера

encoder-only (классификация)
decoder-only (LM)
encoder-decoder (перевод)



BERT и ELMo

(Devlin et al, 2018)



(Peters et al, 2018)

BERT = Bidirectional Encoder Representations from Transformers

transformer network для предобучения модели языка и получения представления слов
– идея трансферного обучения

Идея из GPT:

**давайте возьмём LM – обучим на большом корпусе,
будем «поднастраивать» на конкретные задачи**

Фишки:

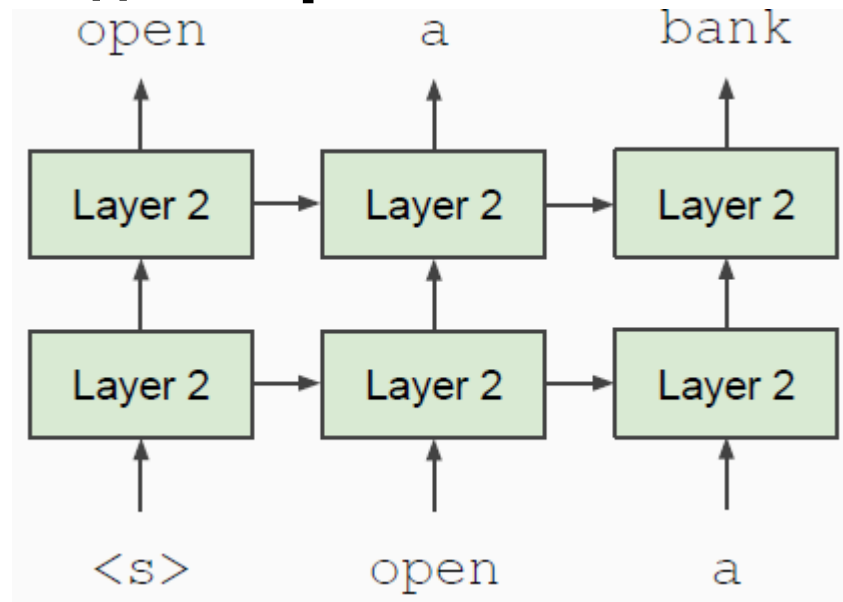
- **двунаправленность (точнее, полный обзор) при обучении**
- **разные задачи для преднастройки (в отличие от ELMo и OpenAI-GPT)**
маскирование, предсказание следующего предложения

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova «Bert: Pre-training of deep bidirectional transformers for language understanding». 2018. <https://arxiv.org/abs/1810.04805>

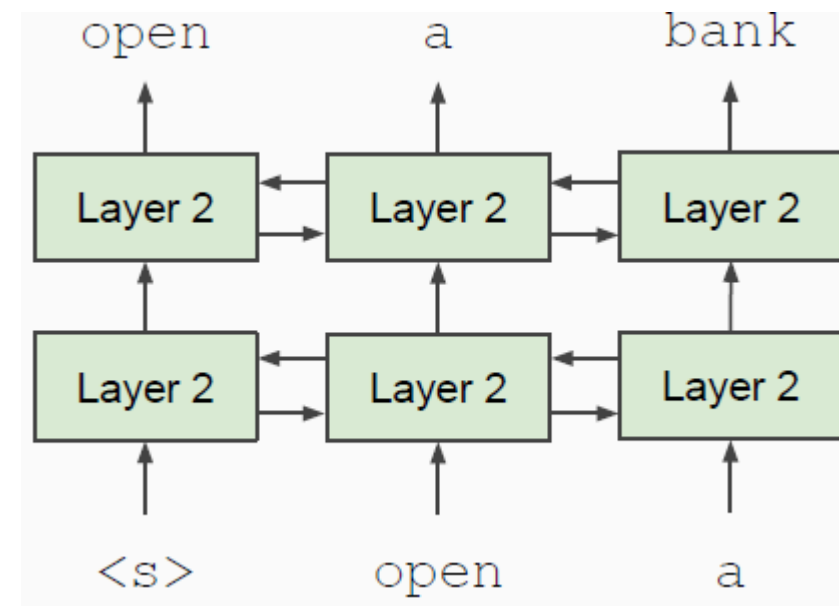
BERT: учёт контекста

обычно левый или правый контекст, а надо «двусторонний»
главный подводный камень – слова увидят себя

односторонний контекст



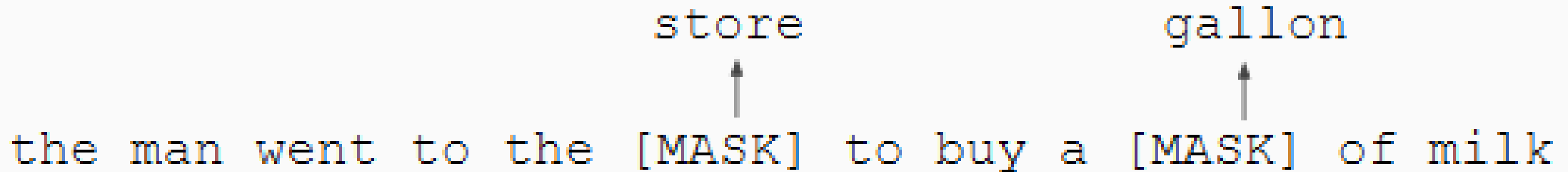
двусторонний контекст



слова видят друг друга

BERT: Mask language model (MLM)

решение: маскировать $k\%$ слов и предсказывать их



store gallon

↑ ↑

the man went to the [MASK] to buy a [MASK] of milk

15% случайно выбранных токенов маскируем – их предсказываем

$p=0.8$ – заменять на [MASK]

went to the store → went to the [MASK]

$p=0.1$ – заменять на случайное слово

went to the store → went to the running

$p=0.1$ – оставлять

went to the store → went to the store

BERT: Связь между предложениями (Next sentence prediction)

Решаем такую задачу: предложение B после A или нет
т.е. это не совсем, как в дословном переводе «предсказание»

Sentence A = The man went to the store.
Sentence B = He bought a gallon of milk.
Label = IsNextSentence

Sentence A = The man went to the store.
Sentence B = Penguins are flightless.
Label = NotNextSentence

генерация выборки: соотношение классов 1/0 = 50/50

BERT: Представление входа (Input Representation)

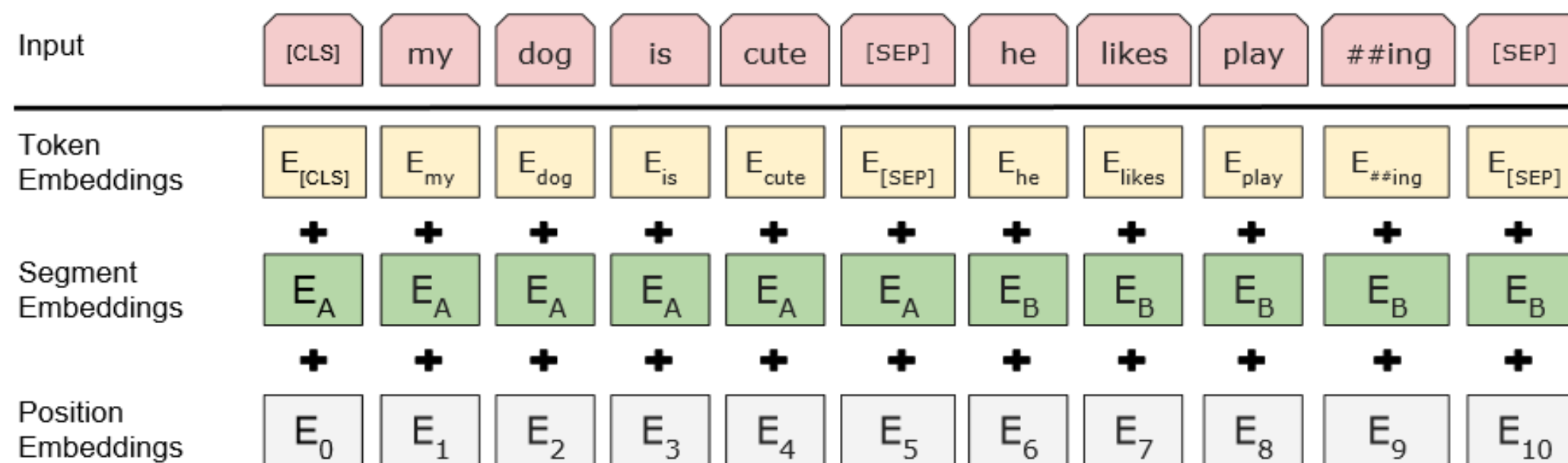


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

каждый токен = сумма 3х вложений
2 предложения на входе разделяются спецсимволом
Представление позиции – обучаемо

BERT: Детали

- **Data: Wikipedia (2.5B words) + BookCorpus (800M words)**
- **Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)**
- **Training Time: 1M steps (~40 epochs)**
- **Optimizer: AdamW, 1e-4 learning rate, linear decay (после 10000 шагов к исходному состоянию)**
- **dropout = 0.1 на всех слоях**
- **BERT-Base: 12-layer, 768-hidden, 12-head (базовая архитектура)**
- **BERT-Large: 24-layer, 1024-hidden, 16-head (большая архитектура)**
- **Trained on 4x4 or 8x8 TPU slice for 4 days**
- **WordPiece-токенизация (30 000 токенов)**

BERT vs GPT vs ELMo

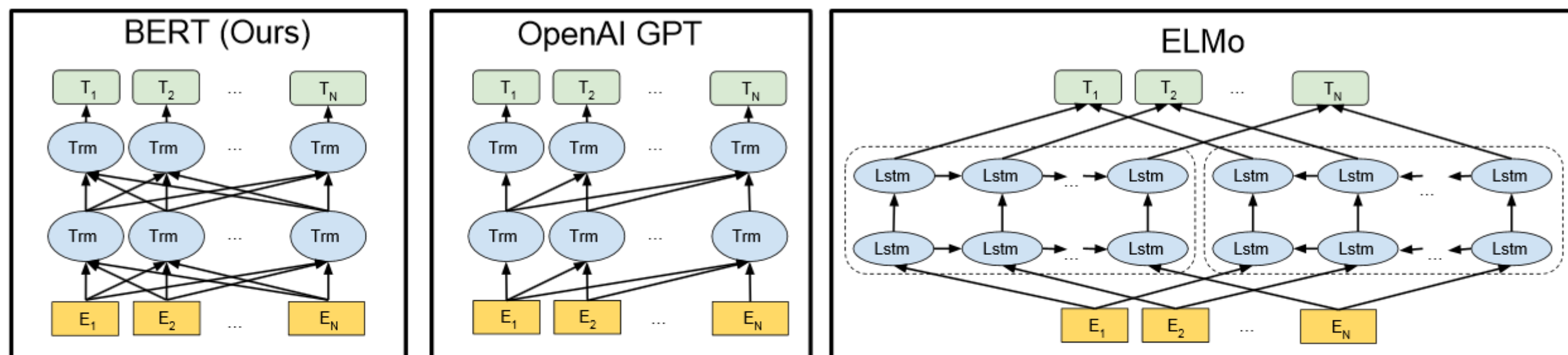


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.

BERT: схема обучения и дообучения на конкретную задачу

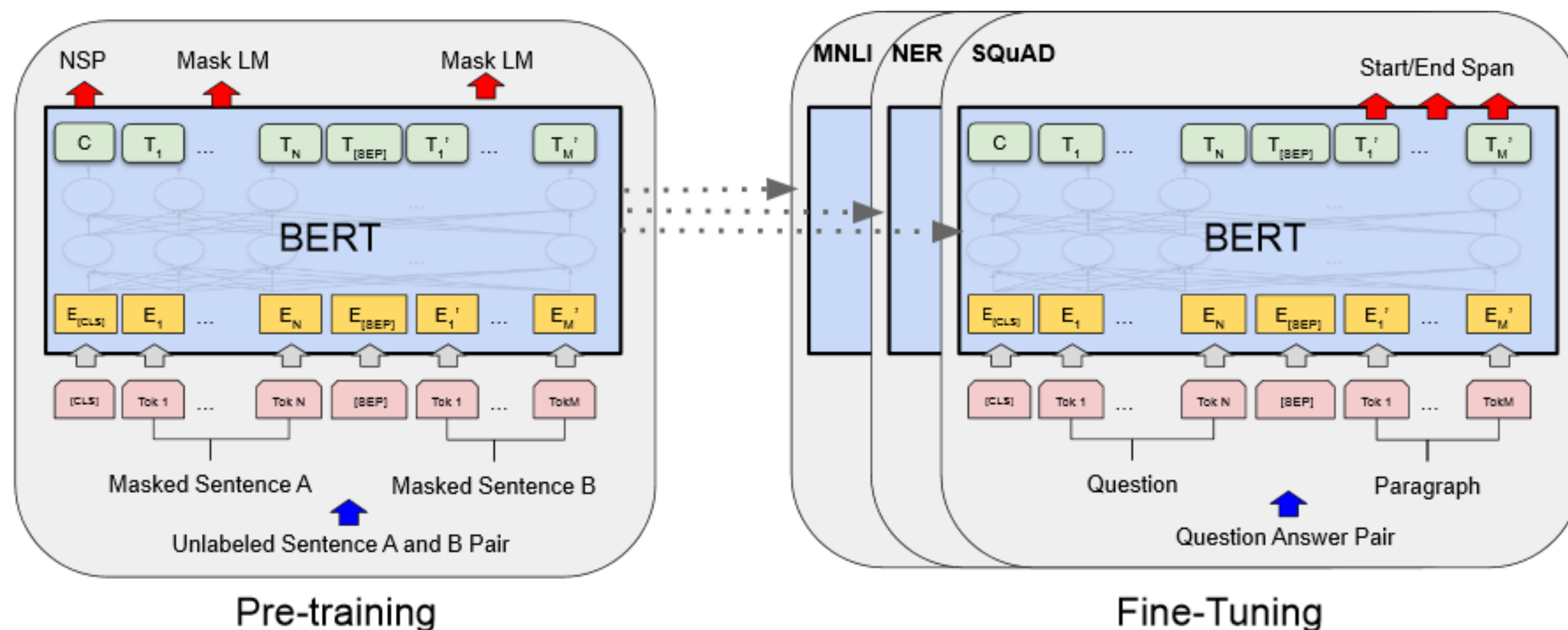
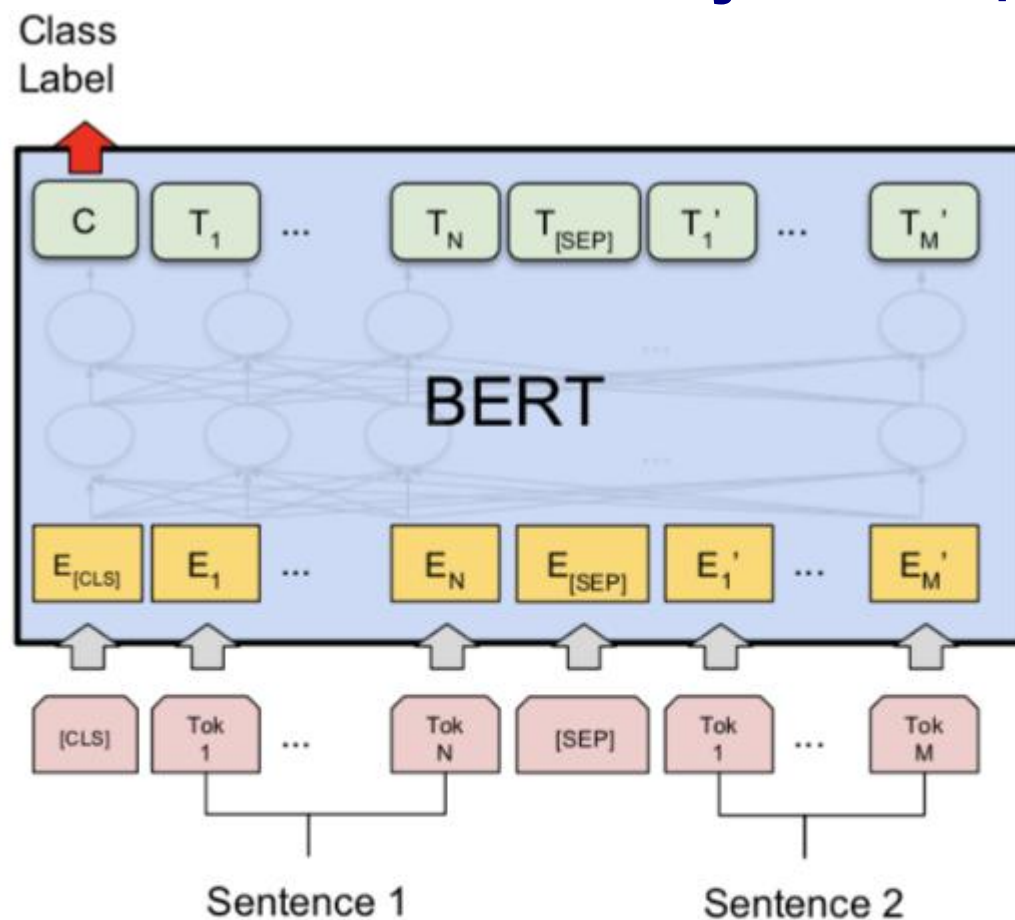
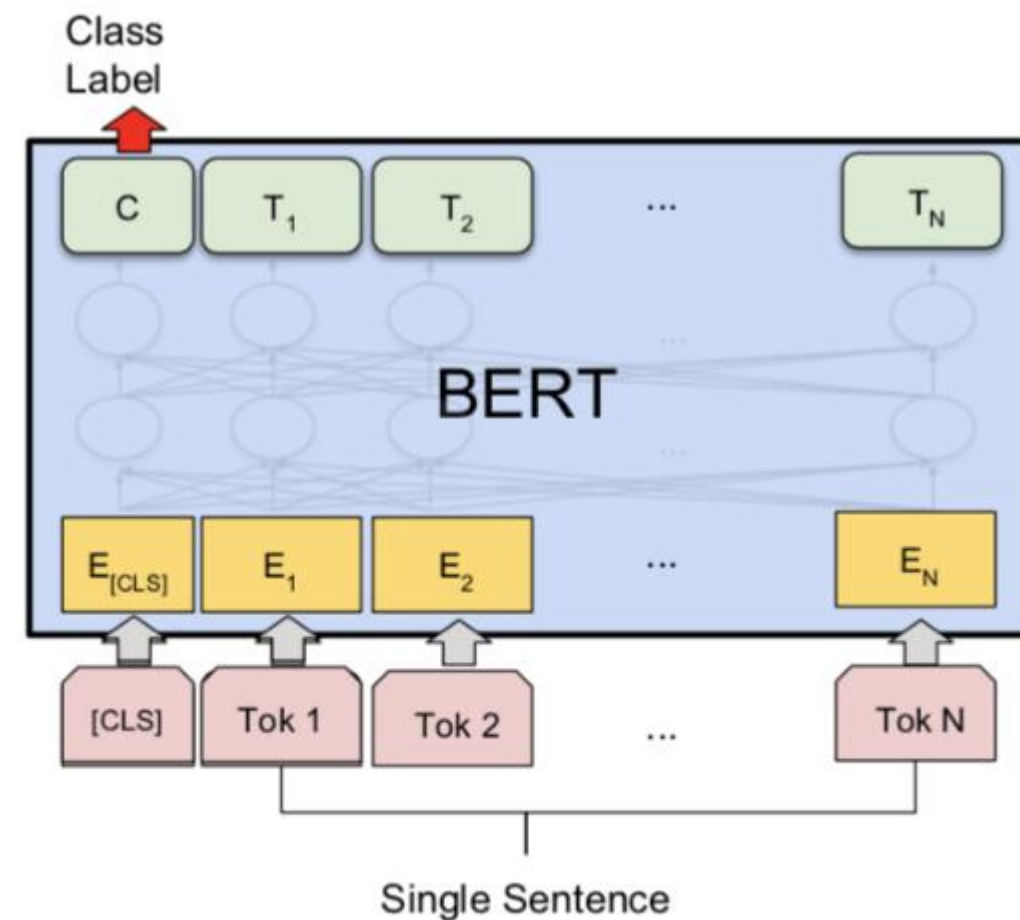


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

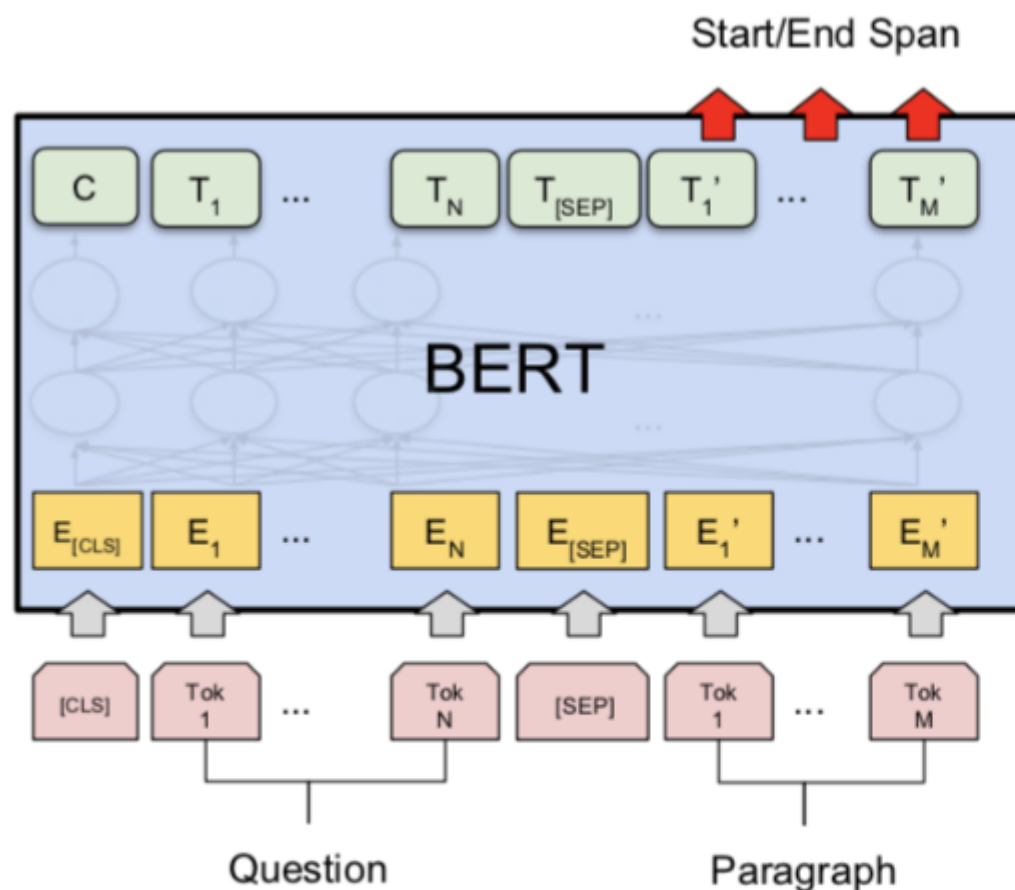
BERT: схема обучения и дообучения на конкретную задачу

(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG

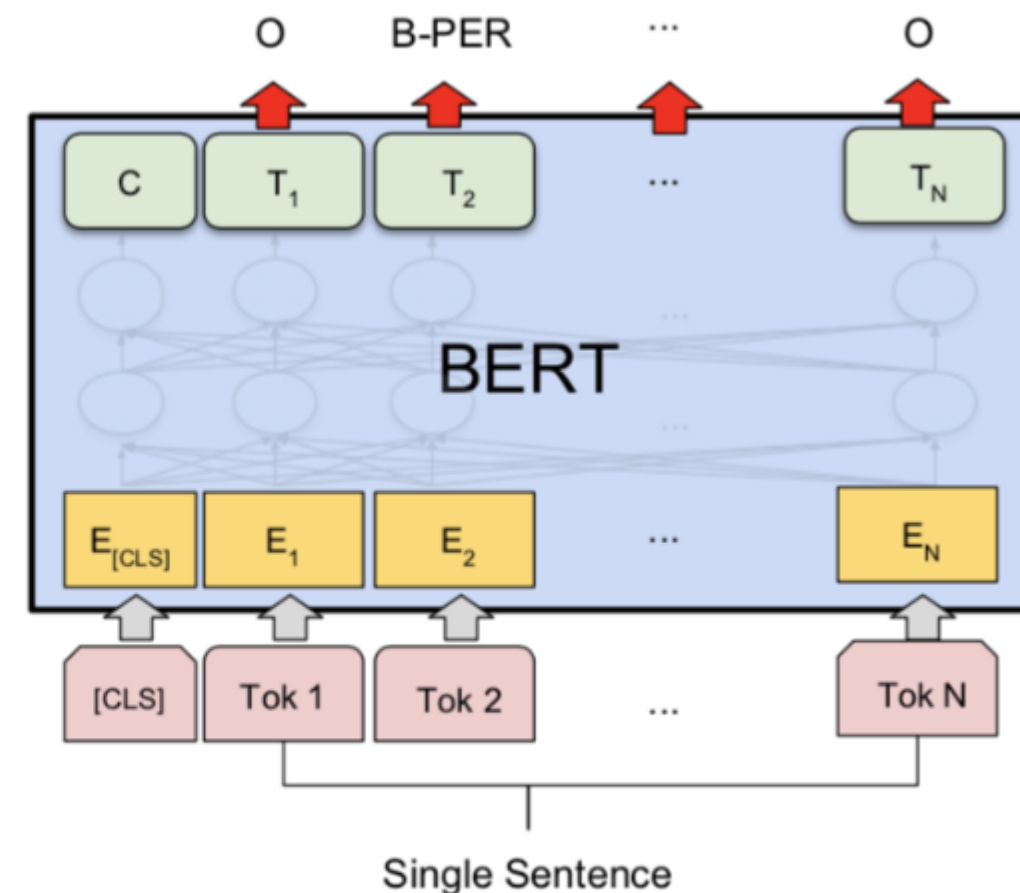


(b) Single Sentence Classification Tasks:
SST-2, CoLA

**для классификации берём финальное состояние специального первого токена [CLS]
умножаем его на обучаемую матрицу и берём softmax**

BERT: схема обучения и дообучения на конкретную задачу

(c) Question Answering Tasks:
SQuAD v1.1

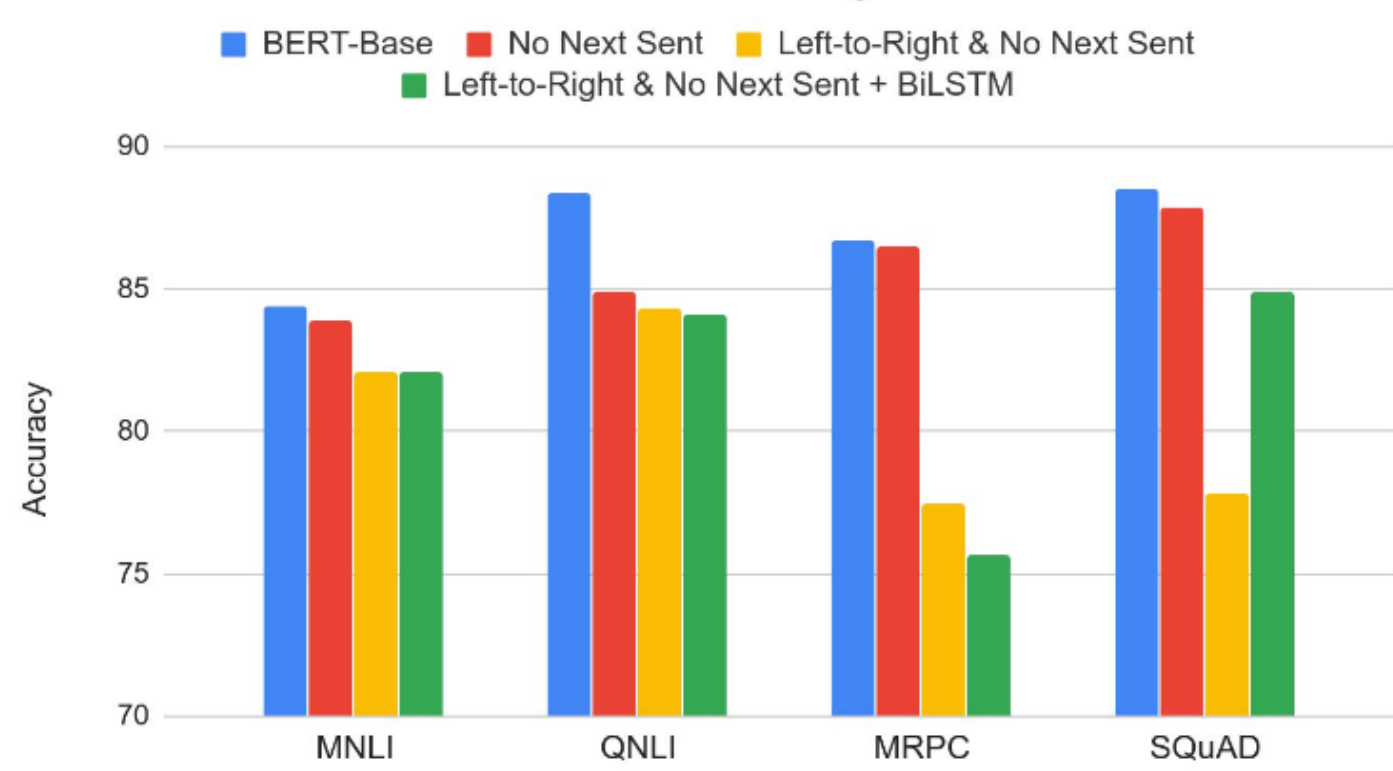


(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

QA: ответ «кусоч текста» – для каждого токена предсказываем вероятность быть началом и концом, обучаемые параметры – две матрицы, которые умножаются на состояния с последующим softmax, чтобы получить вероятности

BERT: Эффект предобучения

Effect of Pre-training Task



Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
BERT _{BASE}	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

Table 5: Ablation over the pre-training tasks using the BERT_{BASE} architecture. “No NSP” is trained without the next sentence prediction task. “LTR & No NSP” is trained as a left-to-right LM without the next sentence prediction, like OpenAI GPT. “+ BiLSTM” adds a randomly initialized BiLSTM on top of the “LTR + No NSP” model during fine-tuning.

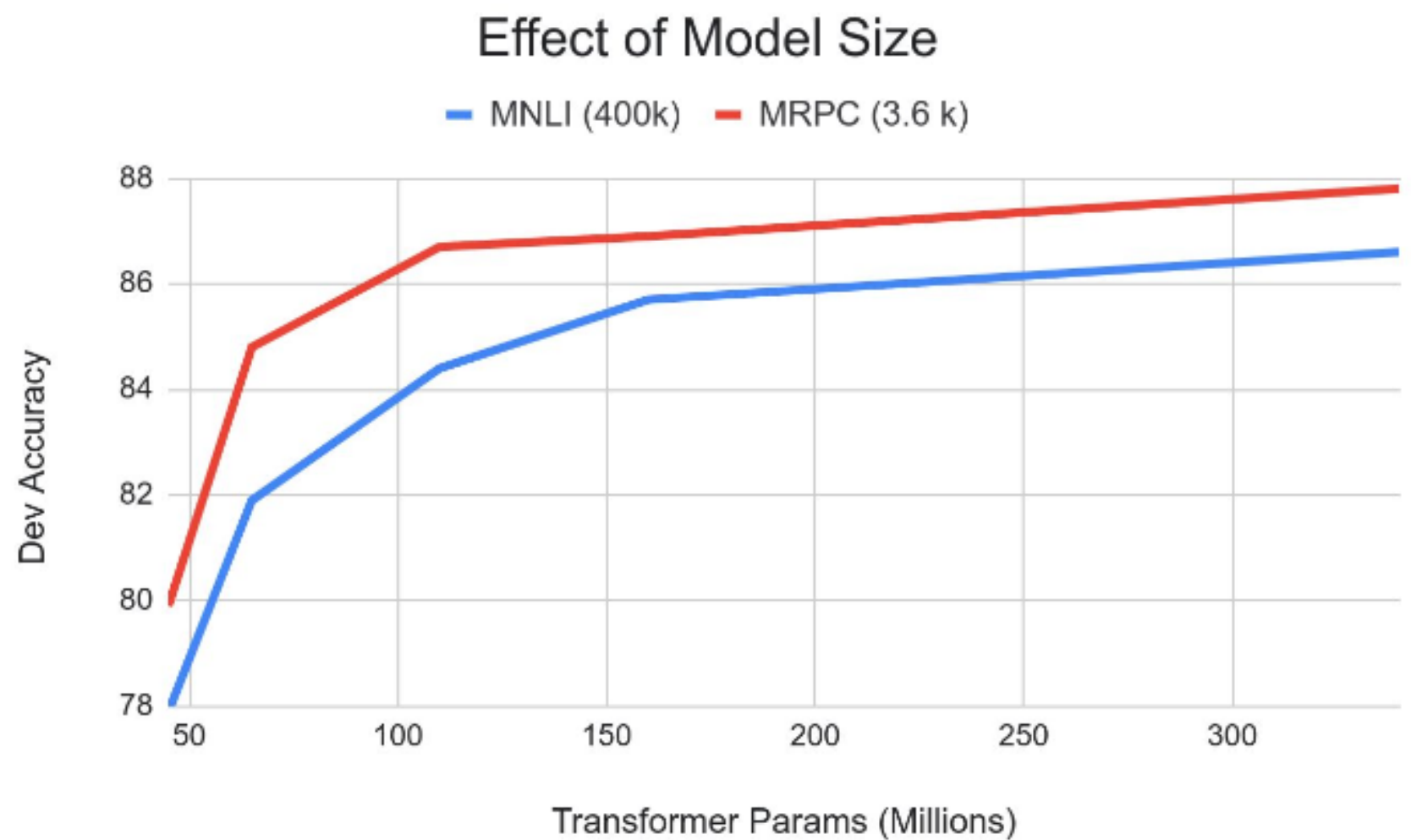
- Для каких-то задач важнее Masked LM, для каких-то NSP
 - на SQuAD плохи модели «слева-направо»

BERT: результаты

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

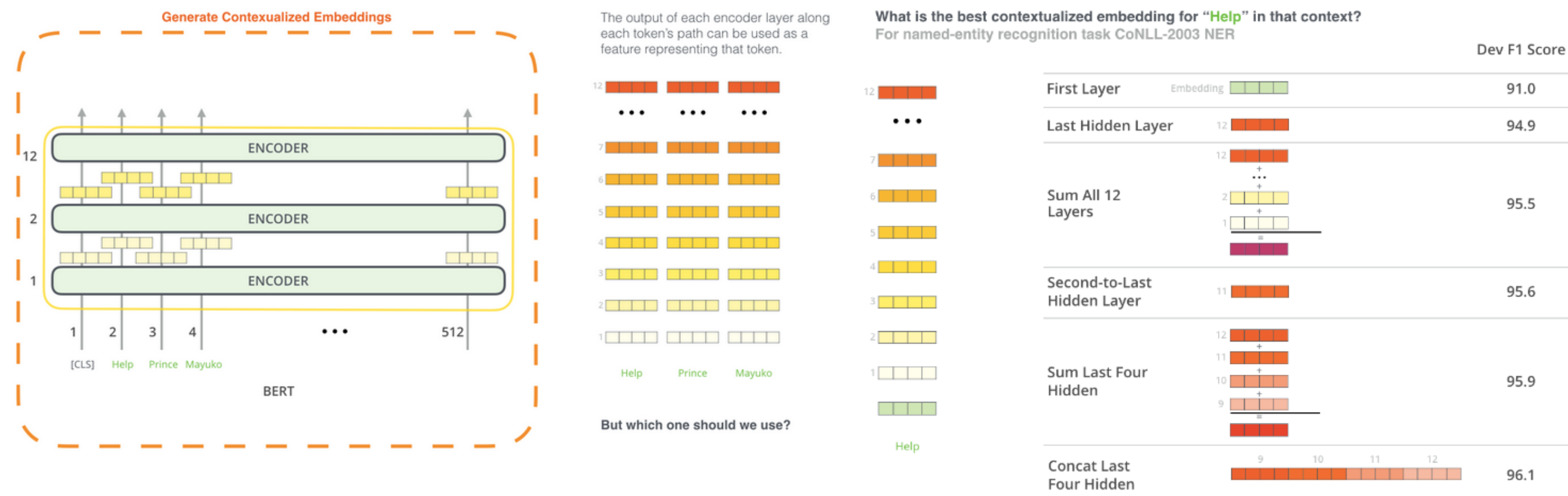
Table 1: GLUE Test results, scored by the evaluation server (<https://gluebenchmark.com/leaderboard>). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.⁸ BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

BERT: эффект размера модели



**глубина имеет значение (даже для относительно небольших датасетов)
не вышли на асимптоту;)**

Представления слов с помощью BERT



как в EIMo можно комбинировать состояния разных уровней