курс «Глубокое обучение»

Трансформер

Александр Дьяконов

31 октября 2022 года

План

attention / self- attention – матричная запись

Transformer

Особенности обучения трансформера

BERT

RoBERTa

SpanBERT

ALBERT

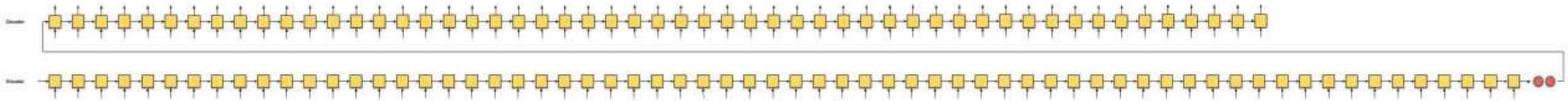
T5: Text-To-Text Transfer Transformer

ELECTRA

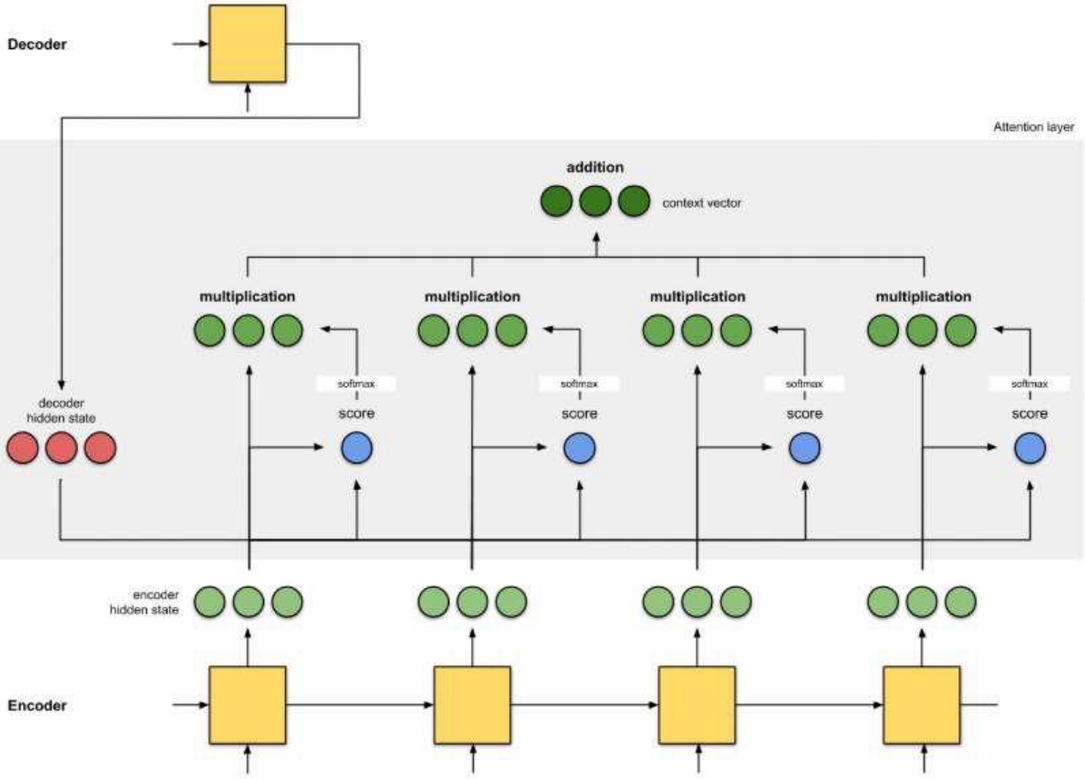
Рис. из https://towardsdatascience.com/@remykarem

Напоминание: внимание

seq2seq

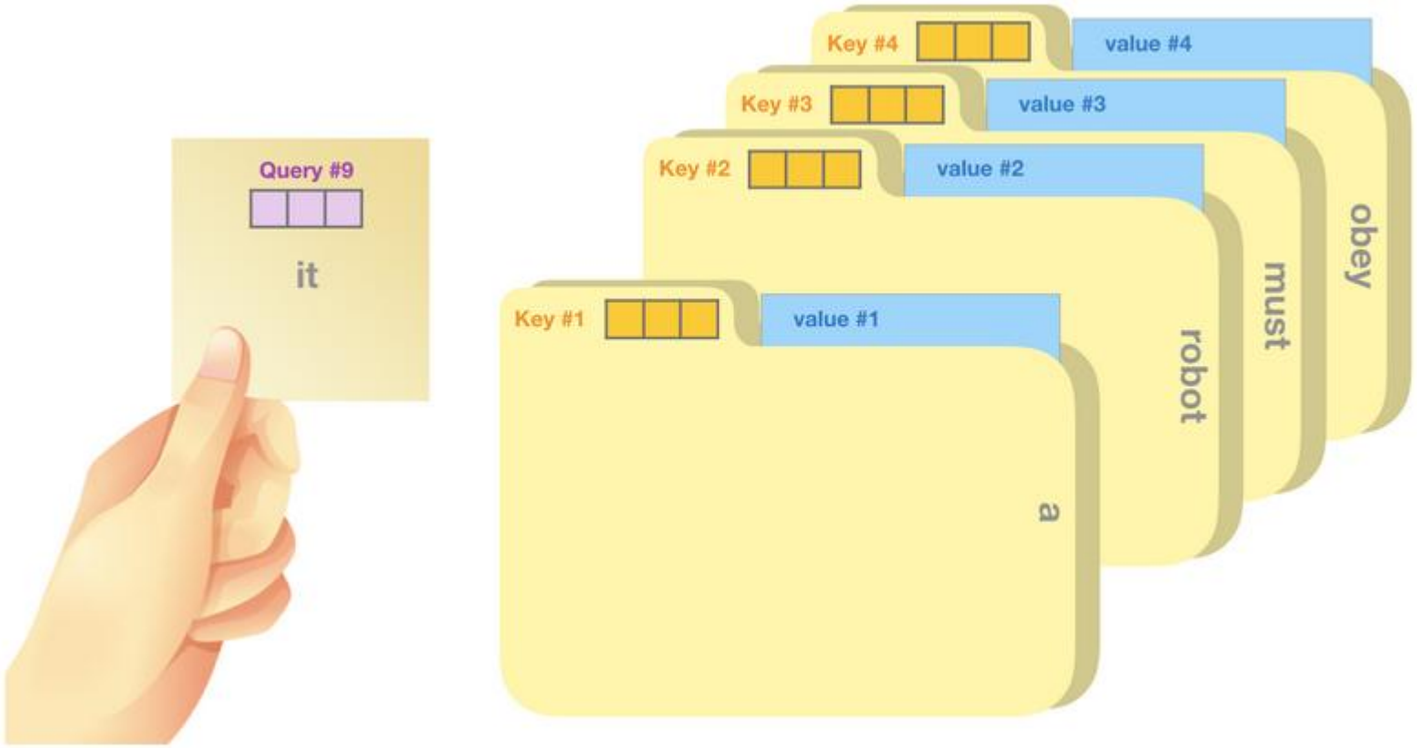


seq2seq + attention: Query, Key, Value



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**

$$V_{d\times s} = W^{\mathrm{v}}_{d\times D} X_{D\times s}$$
$$K_{d\times s} = W^{\mathrm{K}}_{d\times D} X_{D\times s}$$
$$Q_{d\times s} = W^{\mathrm{Q}}_{d\times D} X_{D\times s}$$

размерности K и V м.б. разные

$$v = (V_{d\times s}\,\mathrm{softmax}(K^{\mathrm{T}}_{s\times d} q_{d\times 1})_{s\times 1})_{d\times 1}$$

в матричной форме + нормировка

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

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

+ сделать несколько параллельных матриц

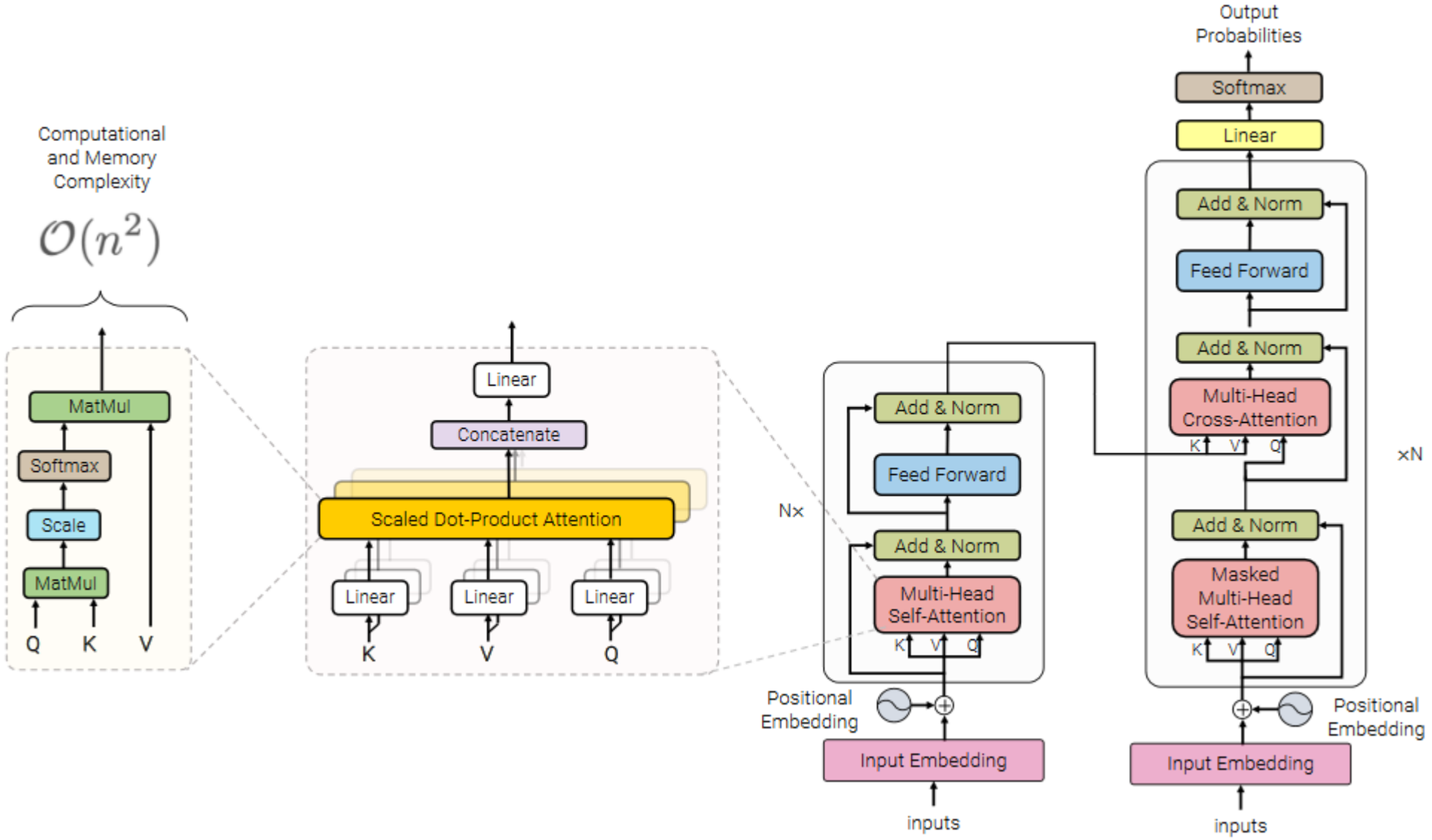
$$W_{D\times 8d}\cdot\mathrm{concat}[\mathrm{head}(W^{\mathrm{v}}_t,W^{\mathrm{K}}_t,W^{\mathrm{Q}}_t)_{d\times s},\ \mathrm{axis}=0]_{t=1}^{8}$$

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://jalammar.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_{OHE} (на вход трансформеру)

self-attention можно рассмотреть как граф + пулинг по релевантности

relevance-based pooling

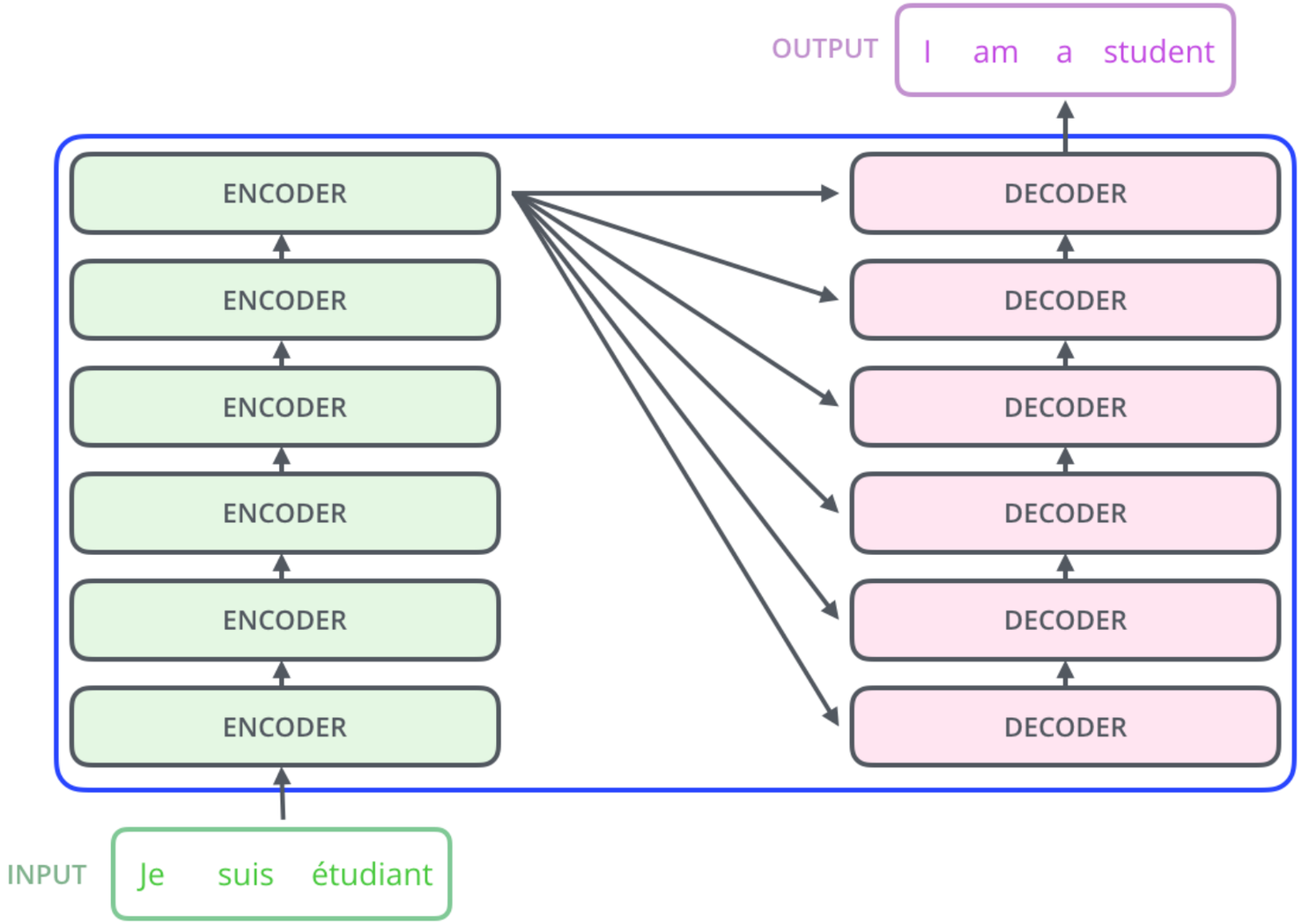
нерекуррентная модель для последовательностей

глубокая модель с блоками внимания

изначально для NMT

последний слой softmax + cross-entropy error

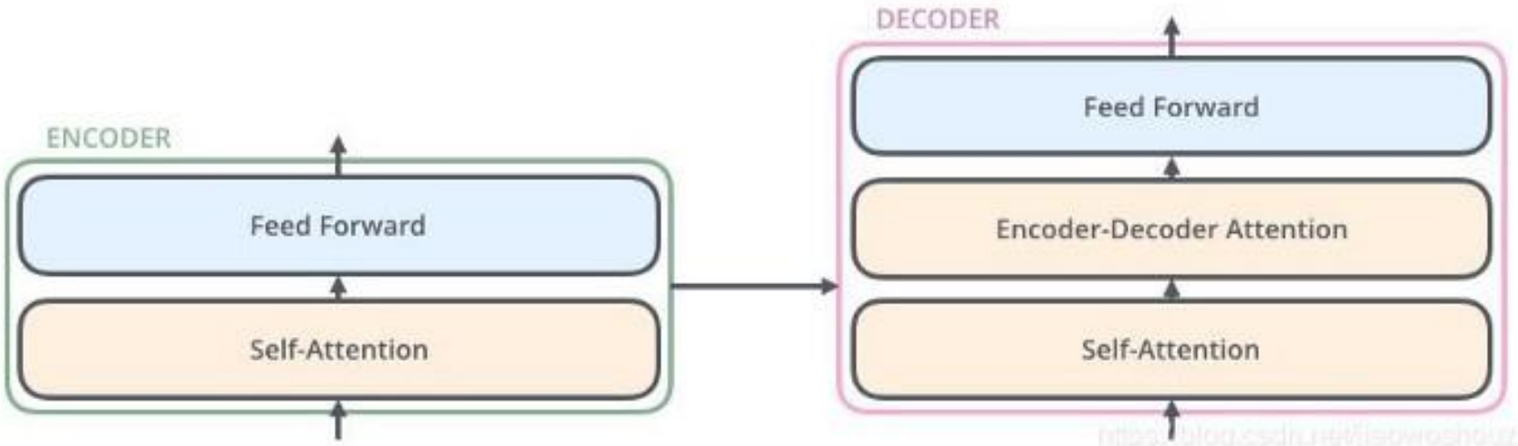
Transformer: общий вид



опять схема «кодировщик» – «декодировщик» (6 + 6 слоёв)

компоненты одинаковые по архитектуре, но веса разные

Transformer: Parallelized Attention



Encoder

1й слой – self-attention

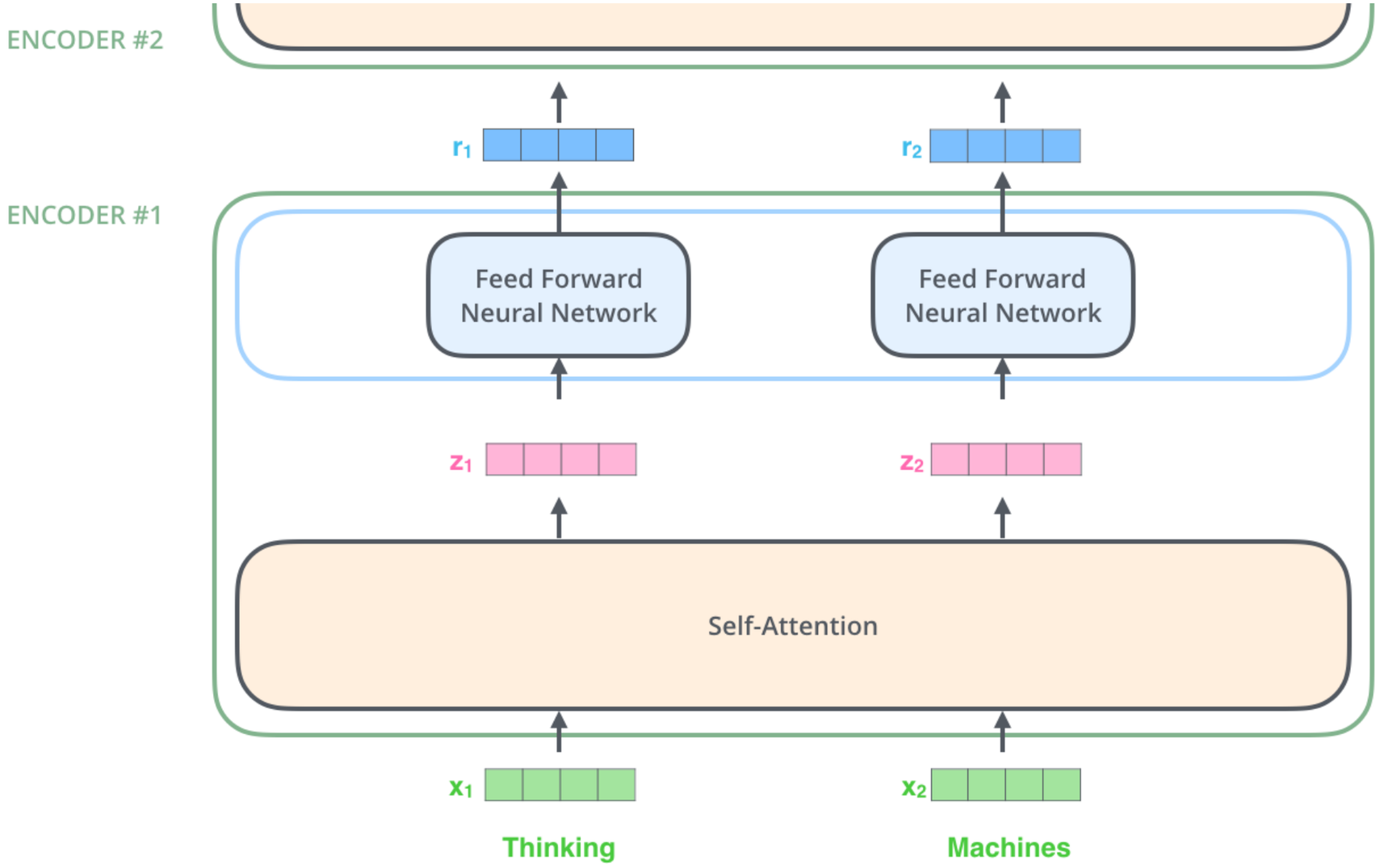
видеть семантику контекста – смотрим сразу на всё предложение

2й - сеть прямого распространения

улучшаем признаковое пространство

Encoder-Decoder Attention – смотрим на всё предложение на входе

Transformer: Encoder



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

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

Конценция «Self-Attention»

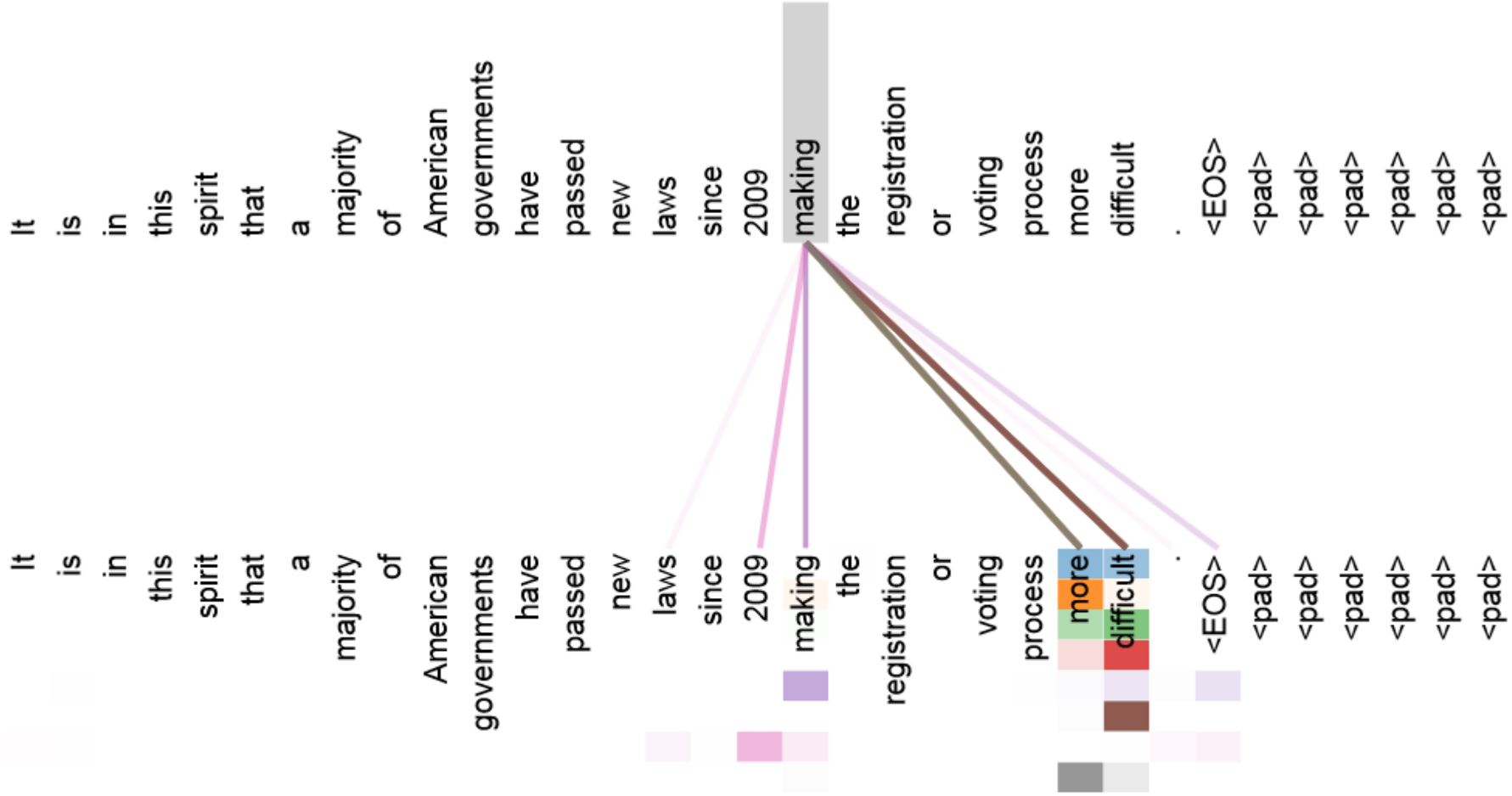


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word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

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

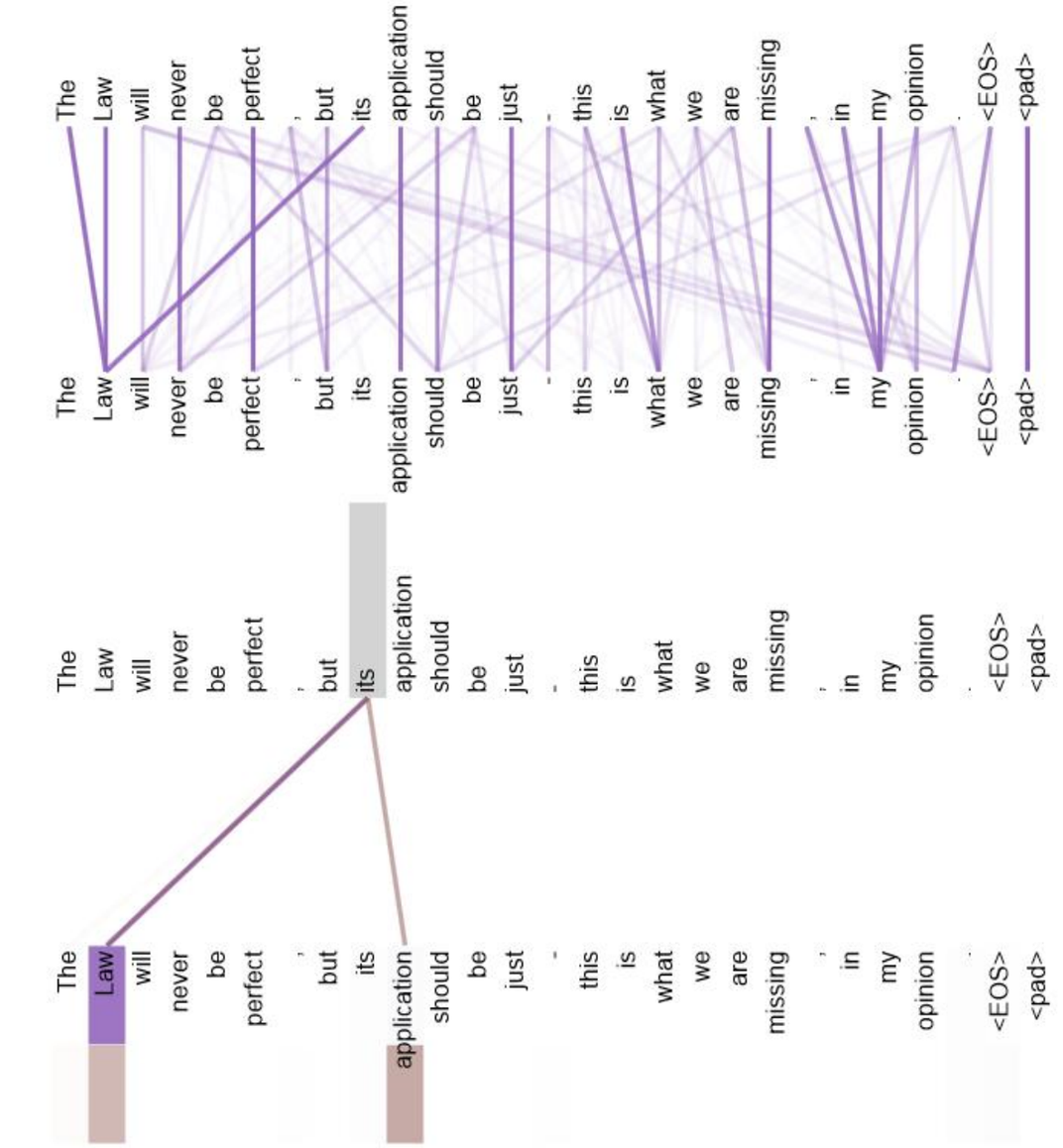
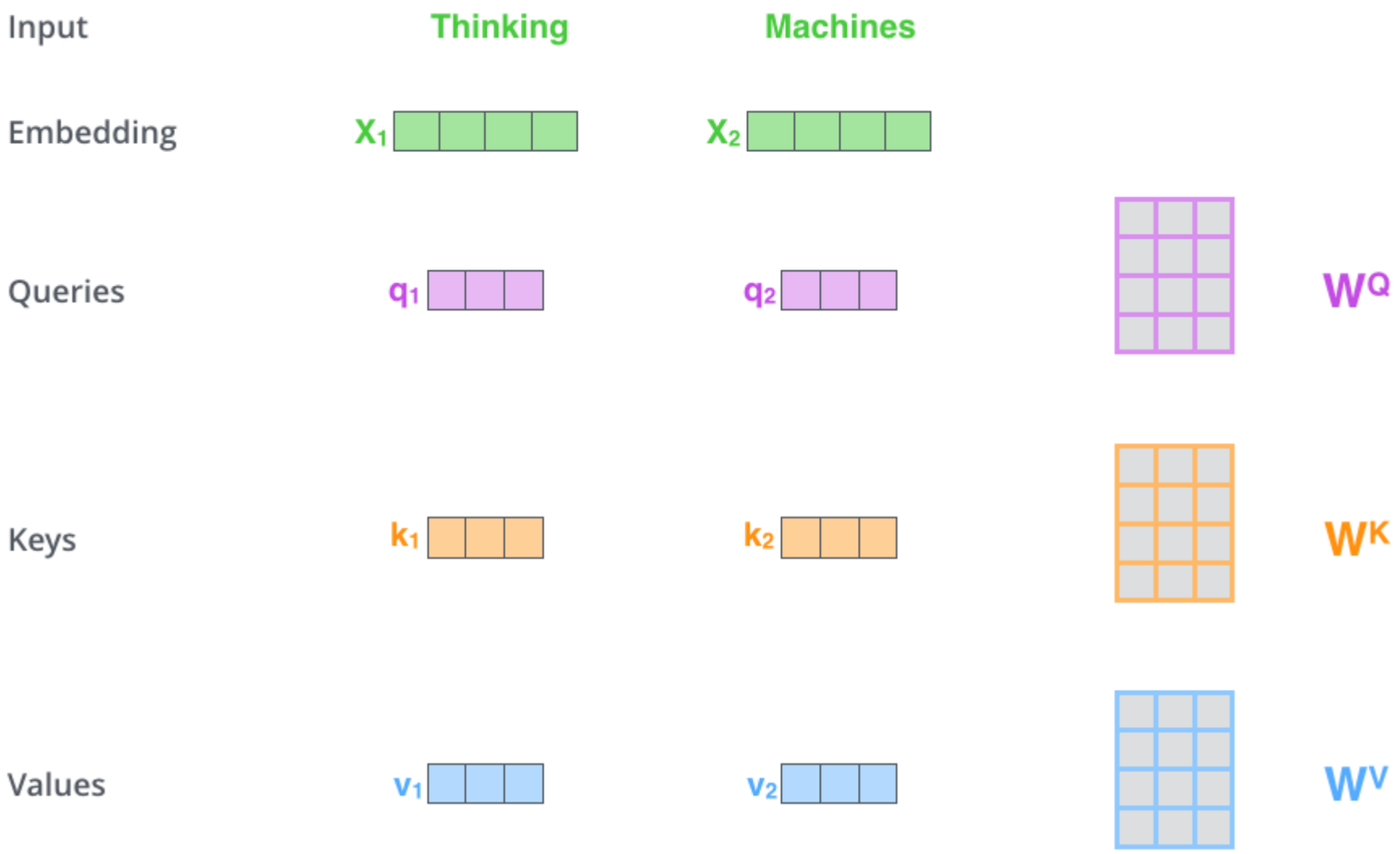


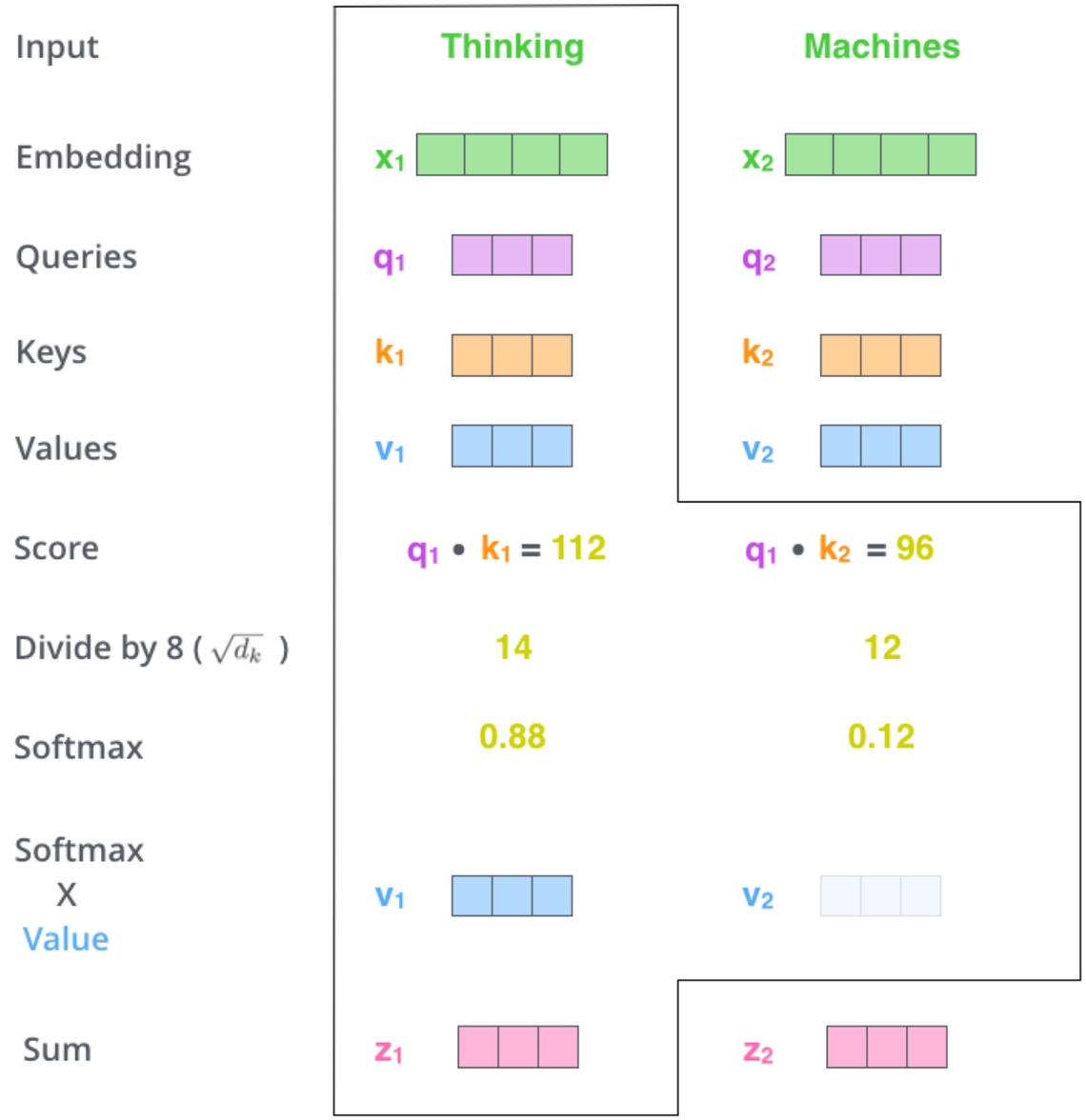
Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Реализация «Self-Attention»

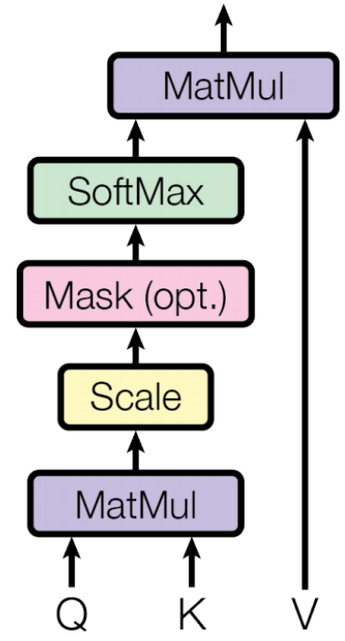


три 64-мерных вектора: Query, Key, Value – получаются из представлений (embeddings) умножением на матрицу (для каждого QKV – своя) – это обучаемый параметр

Реализация «Self-Attention»

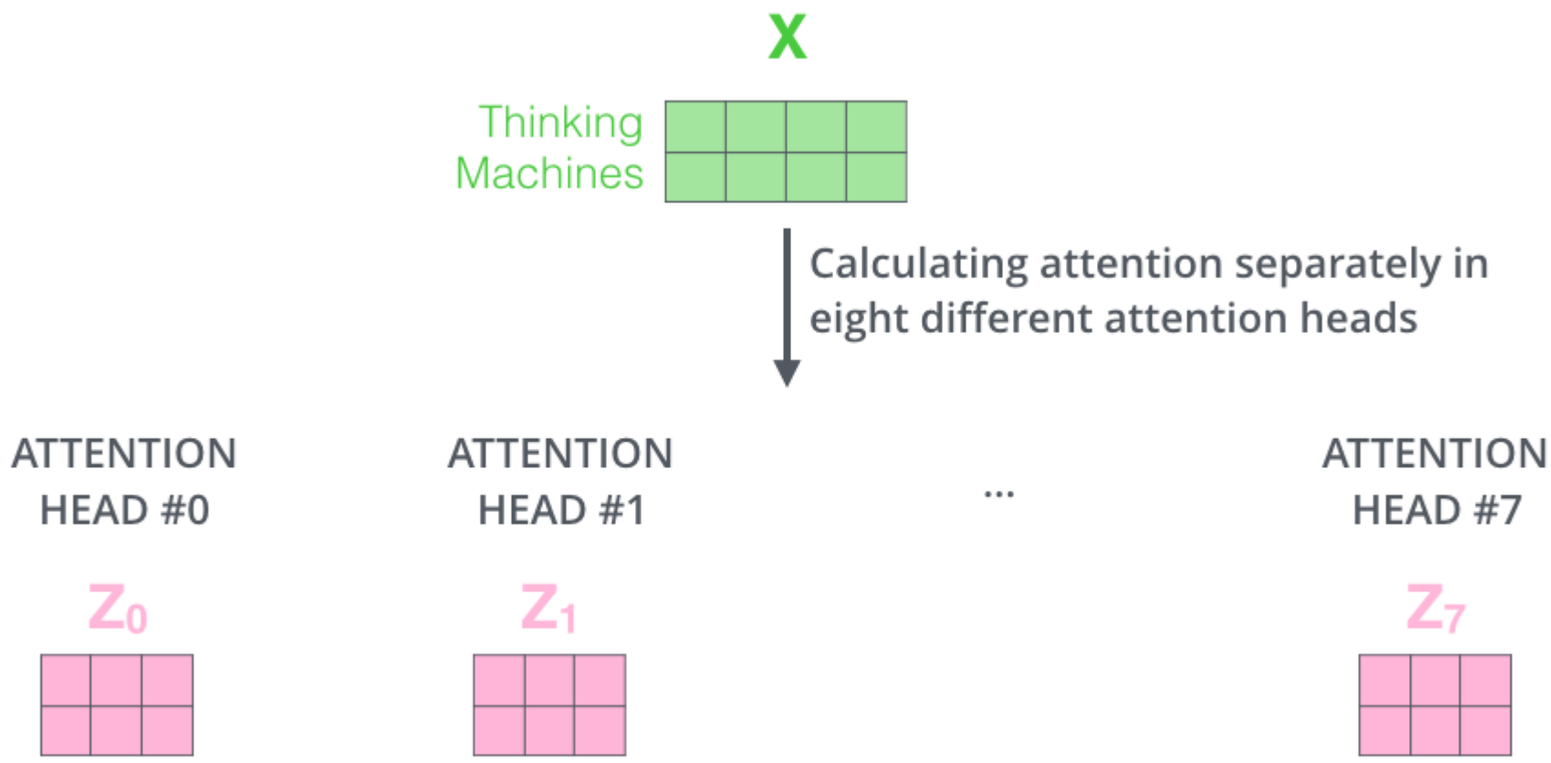


Реализация «Self-Attention»: простая матричная



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

Multi-Headed Attention (несколько головок)

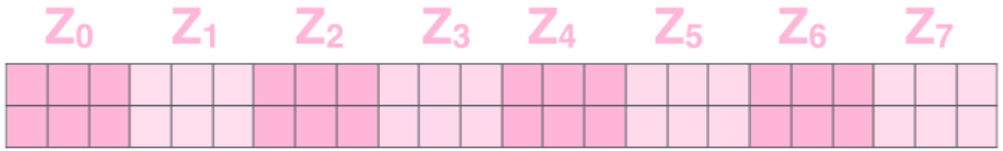


каждая головка обозревает свой аспект внимания (что это, что делает и т.п.)

как теперь превратить результаты разных головок в вектор нужной размерности?

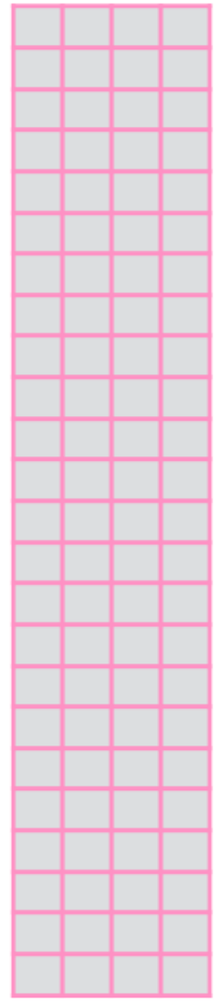
Multi-Headed Attention (несколько головок)

1) Concatenate all the attention heads



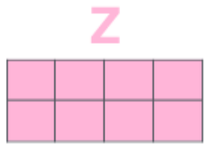
2) Multiply with a weight matrix W^O that was trained jointly with the model

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

=



конкатенация и умножение на ещё одну матрицу весов

Multi-Headed Attention (несколько головок) – саммари

1) This is our input sentence*

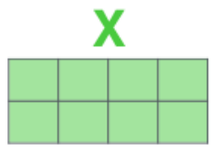
2) We embed each word*

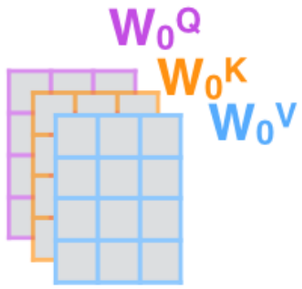
3) Split into 8 heads. We multiply X or R with weight matrices

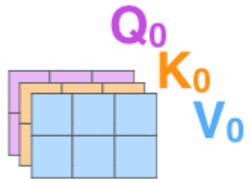
4) Calculate attention using the resulting Q/K/V matrices

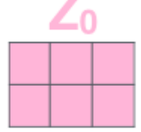
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

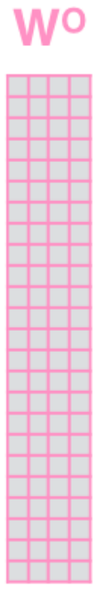
Thinking Machines

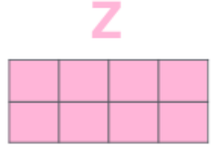




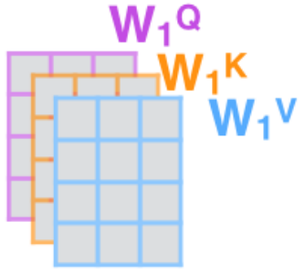


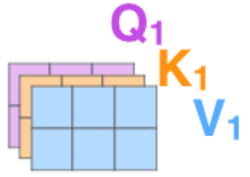


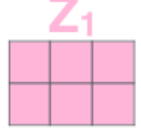


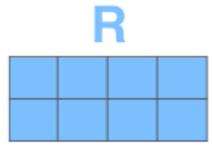


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

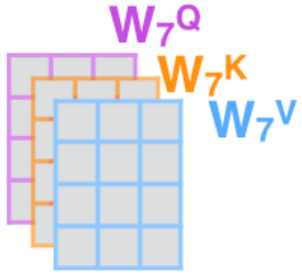




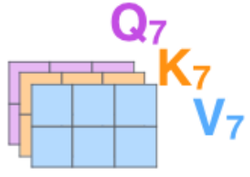




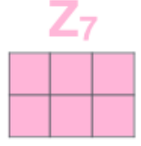
...



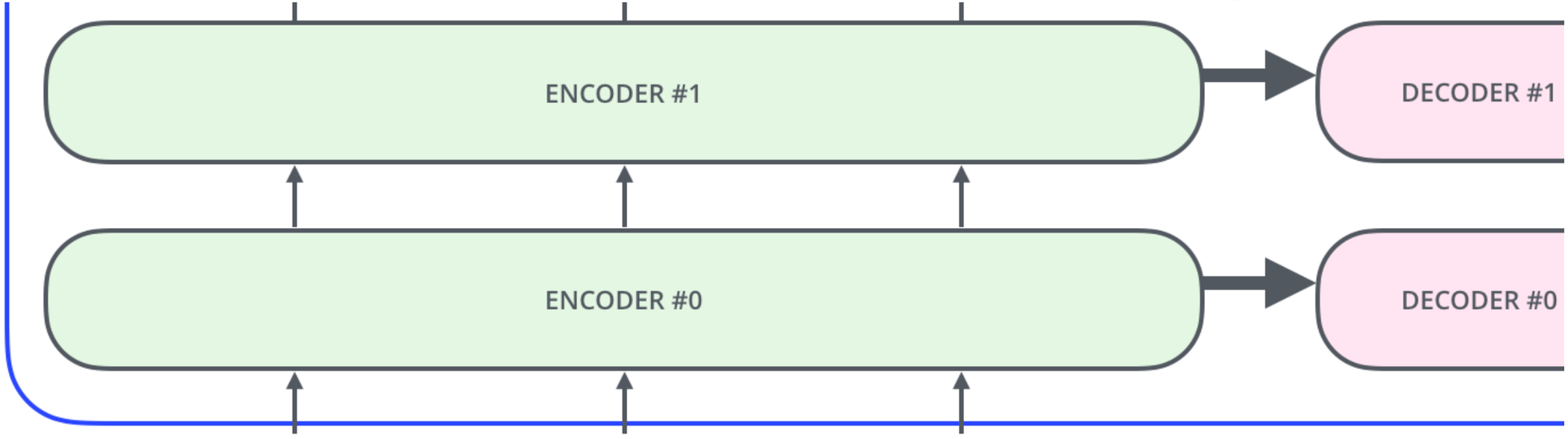
...

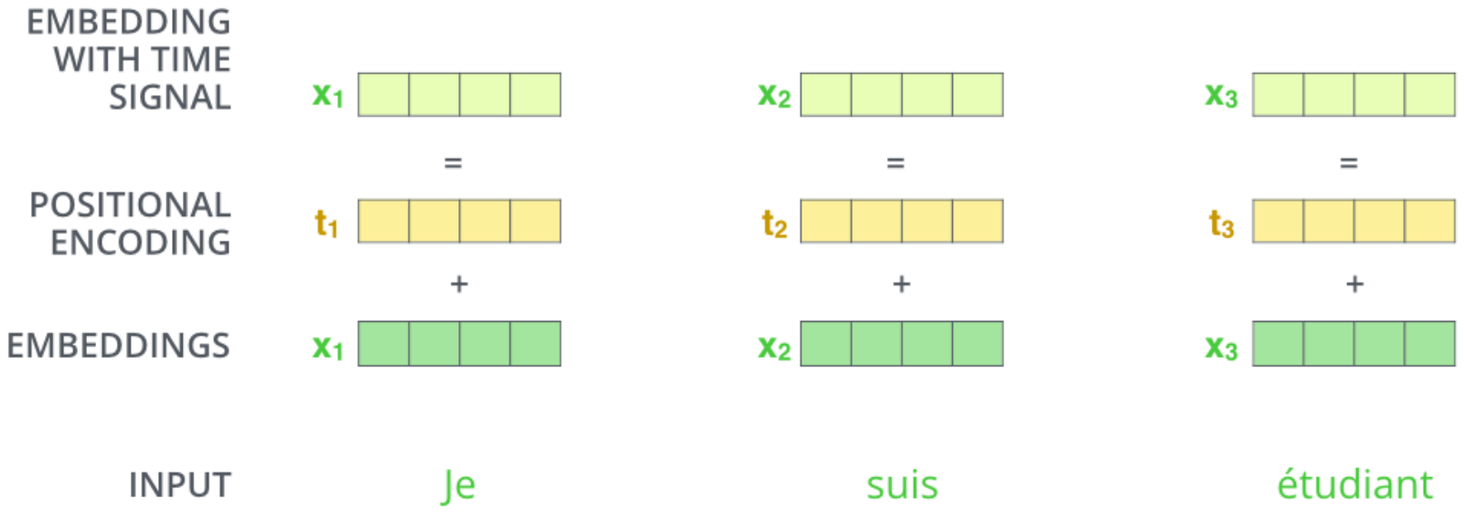


...



Кодирование позиции слова: Positional Encoding

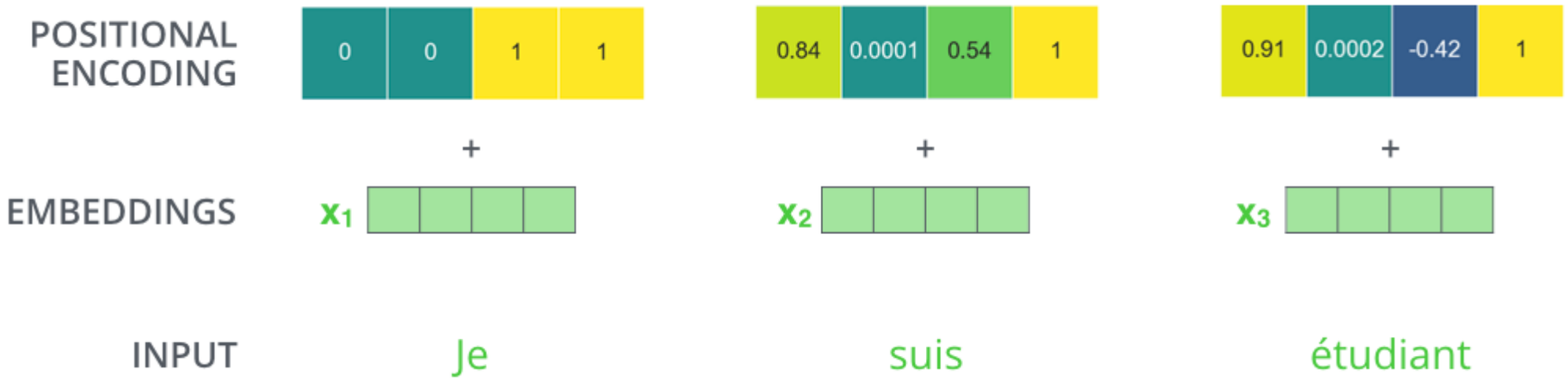




Кодирование позиции слова: Positional Encoding

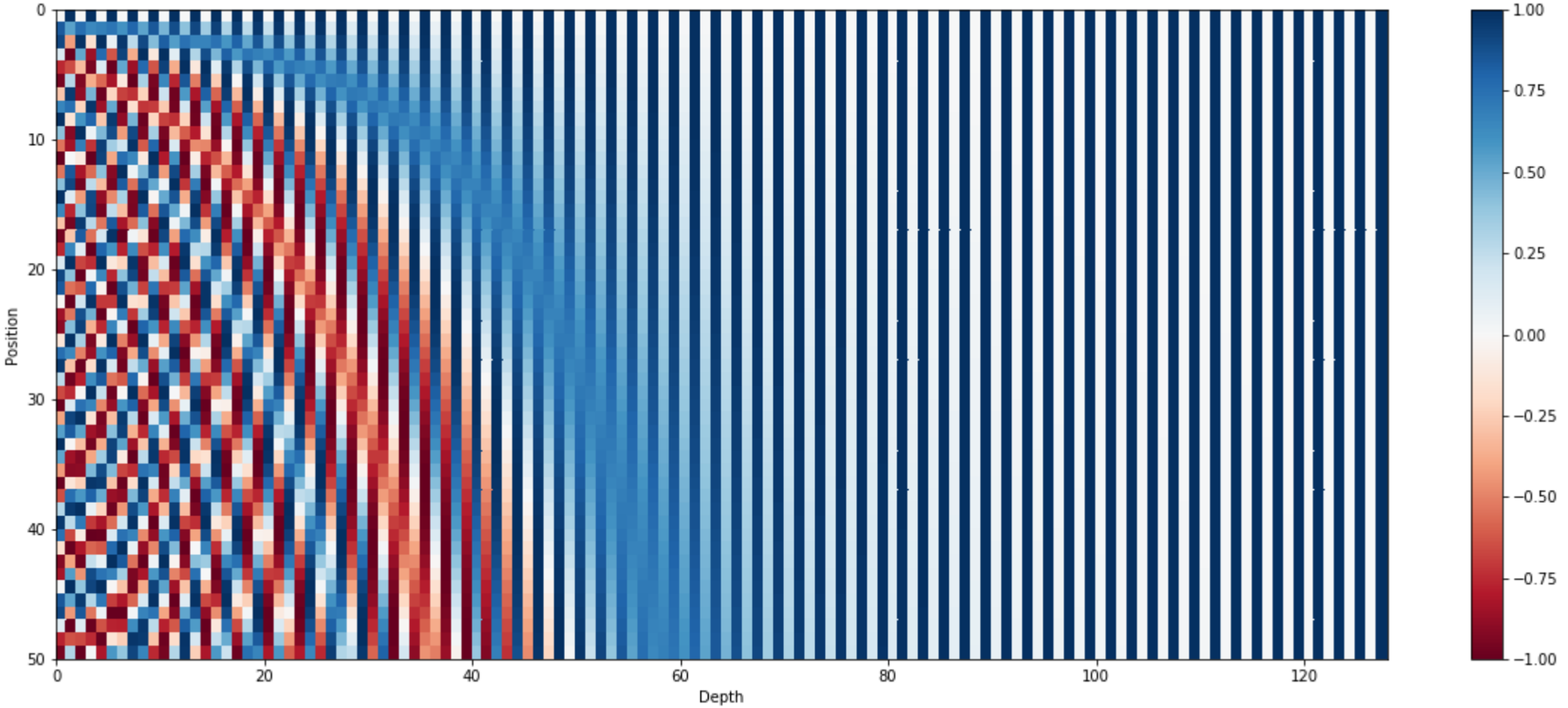
Проблема: пока наш метод не зависит от перестановки слов,
а надо учитывать порядок ⇒ будем кодировать позицию
Но надо, чтобы метод не зависел от длины входного предложения
номеру позиции t соответствует d-мерный вектор, позиции которого

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



картинка для размерности представления = 4

Кодирование позиции слова: Positional Encoding



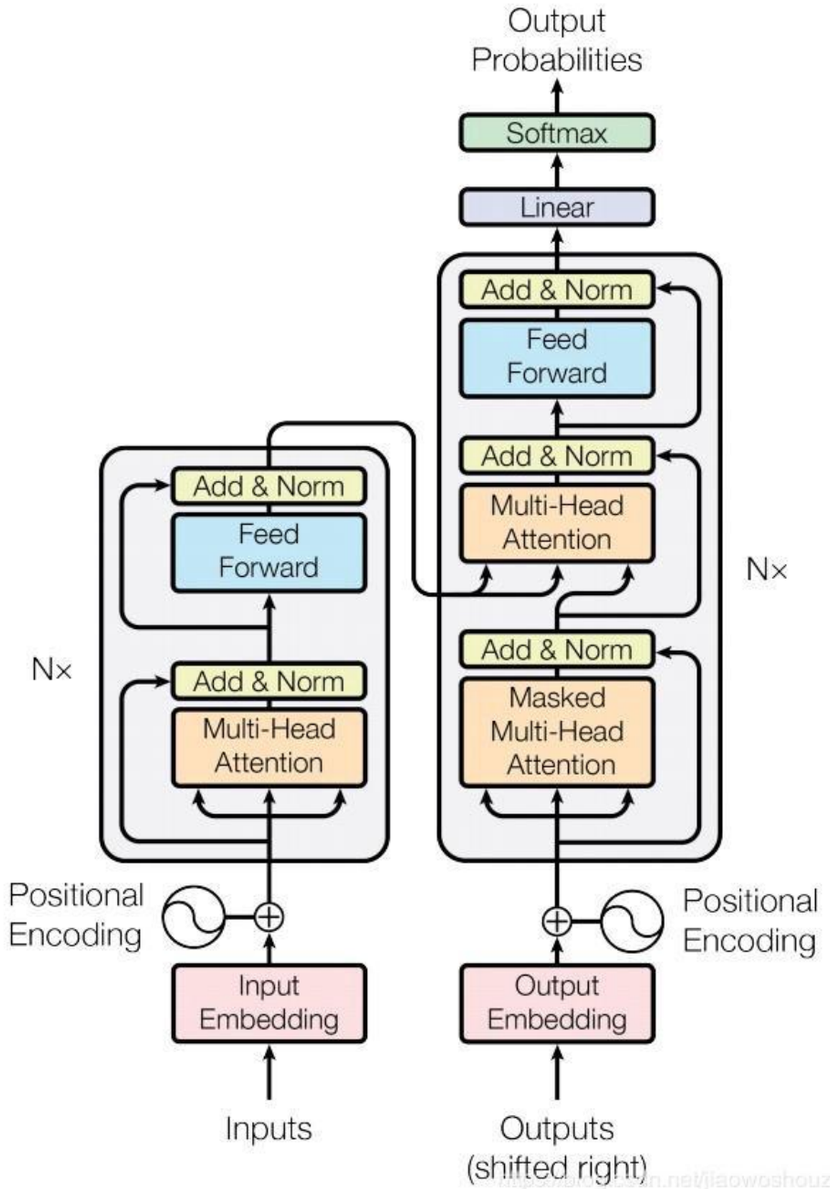
128-мерное кодирование 50 позиций

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

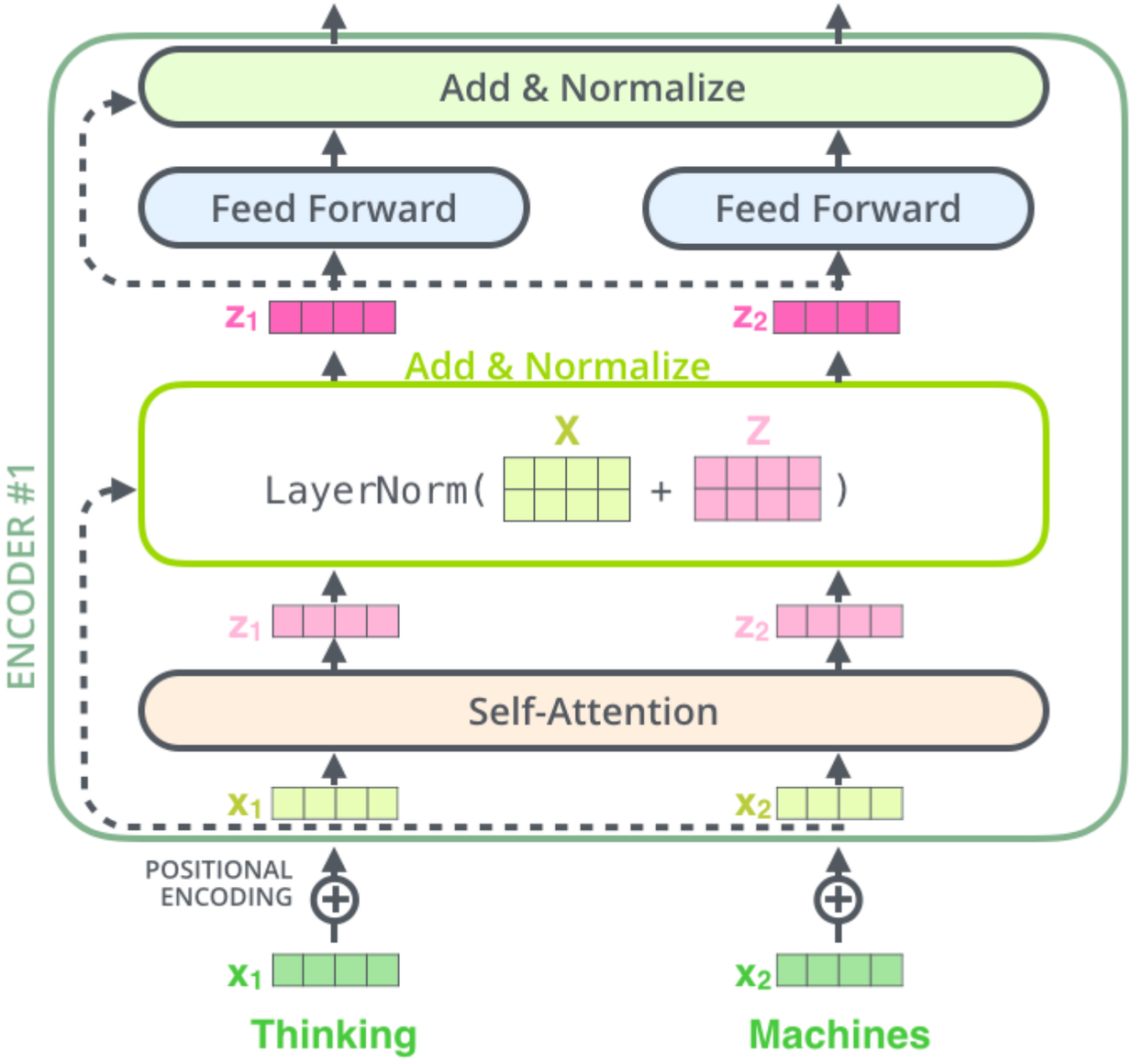
Кодирование позиции слова: Positional Encoding

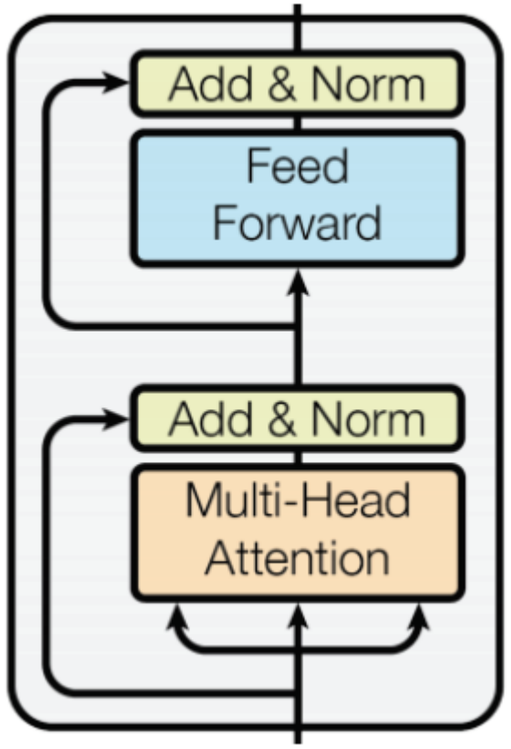
```
class PositionalEncoding(nn.Module):
    "Implement the PE function."
    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # 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) # не считается параметром модели

    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

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

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

```
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
```

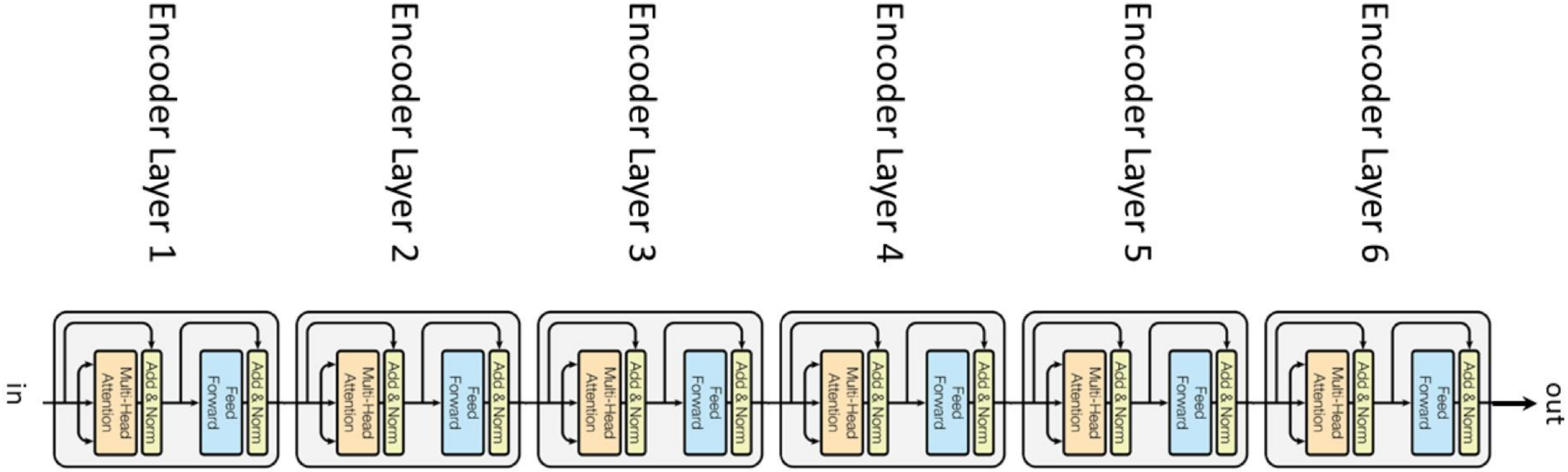
http://nlp.seas.harvard.edu/2018/04/03/attention.html

Transformer: FFNN

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

```
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))))
```

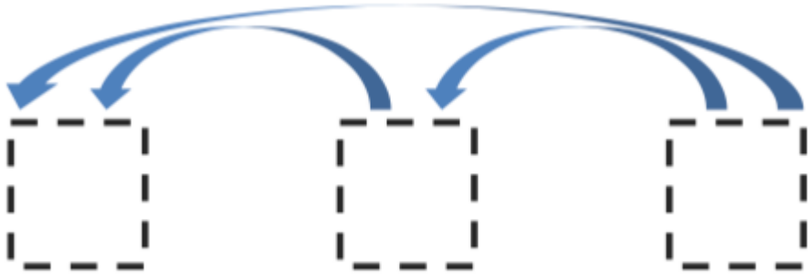
Transformer: Encoder



Transformer: виды внимания

Encoder Self-Attention

Encoder-Decoder Attention

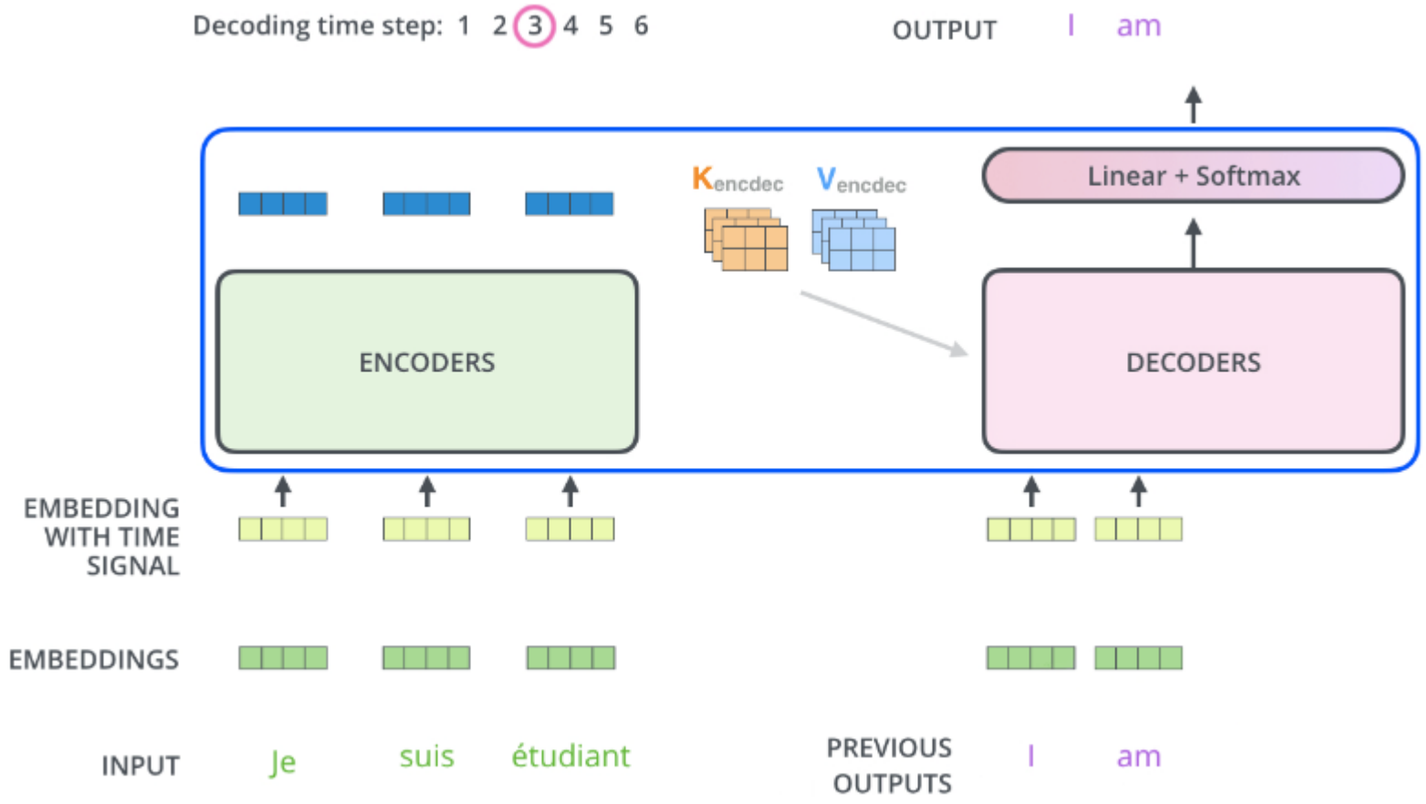


MaskedDecoder Self-Attention

Transformer: Decoder

Тонкость: не знаем длину ответа

⇒ будем его получать последовательно



Transformer: Decoder

Другая тонкость: при обучении нельзя видеть «будущее ответа» ⇒ маскирование

на схеме есть «masked multi-head attention»

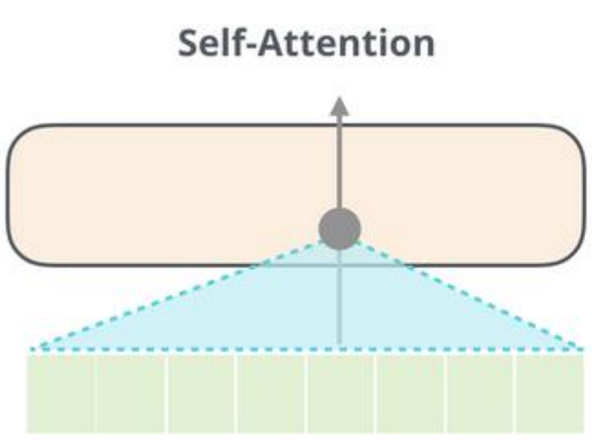
padding mask (in all the scaled dot-product attention)

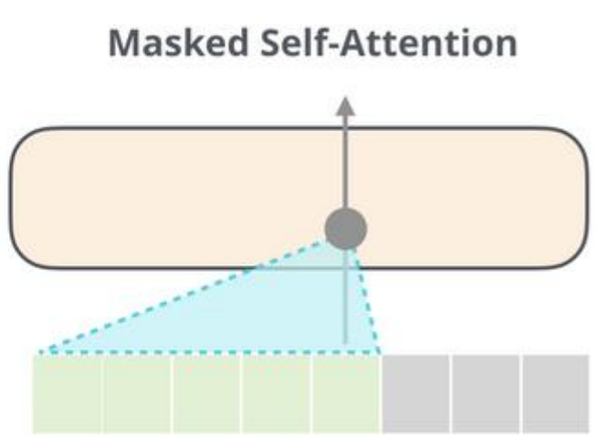
~ добавляем 0 в конец коротких предложений

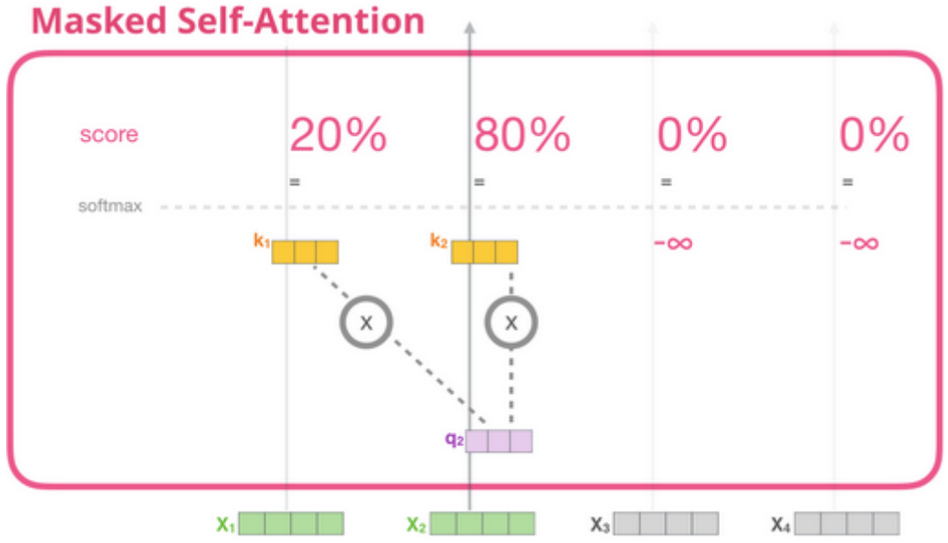
sequence mask (in the decoder's self-attention)

чтобы не видел информацию из будущего

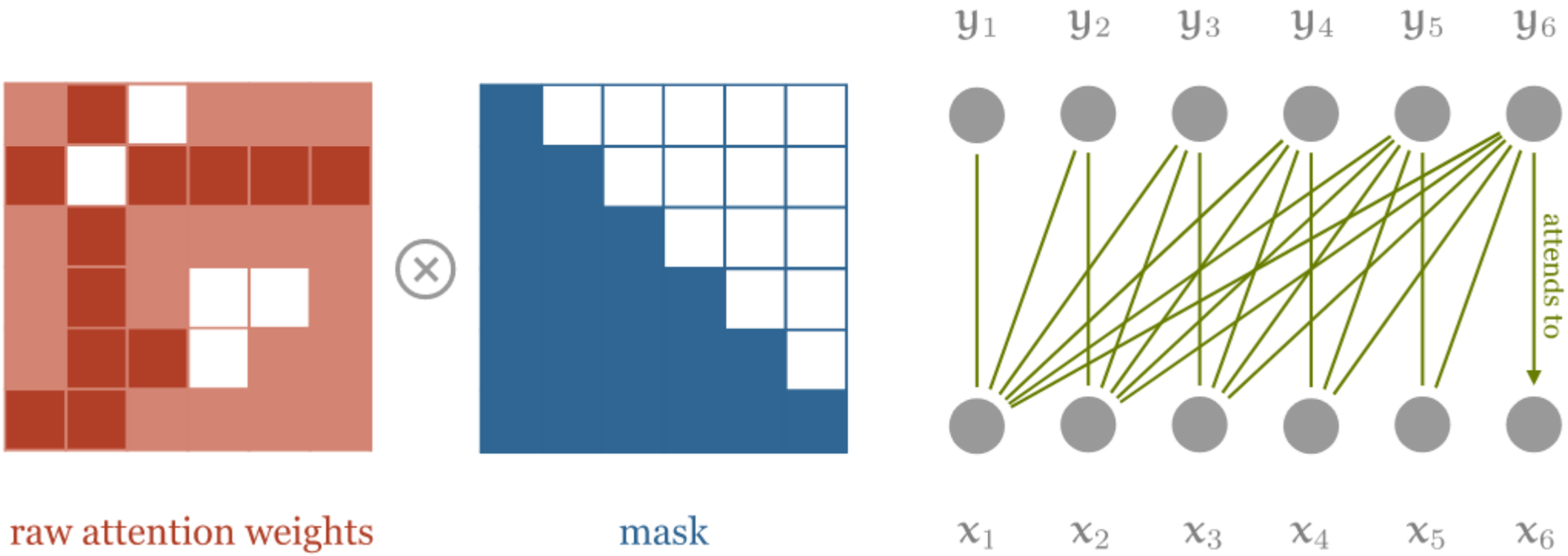
выводит одно слово за такт







Transformer: маскирование




```
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

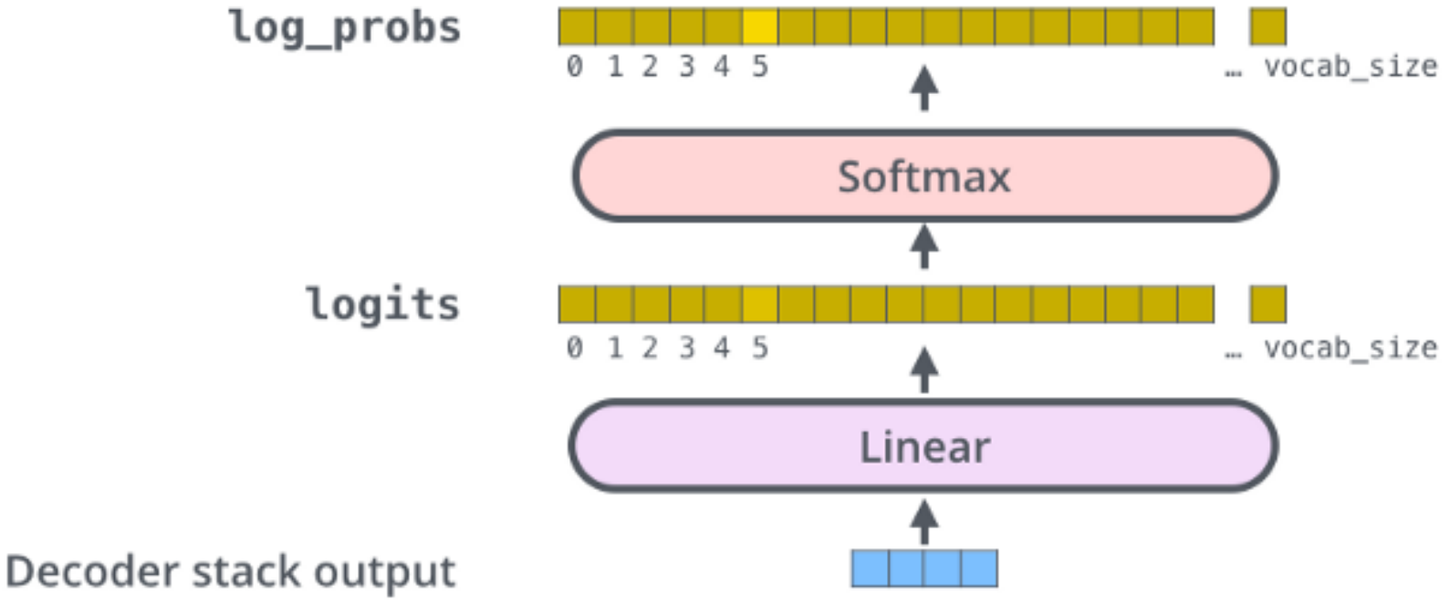
как на выходе получить одно слово:

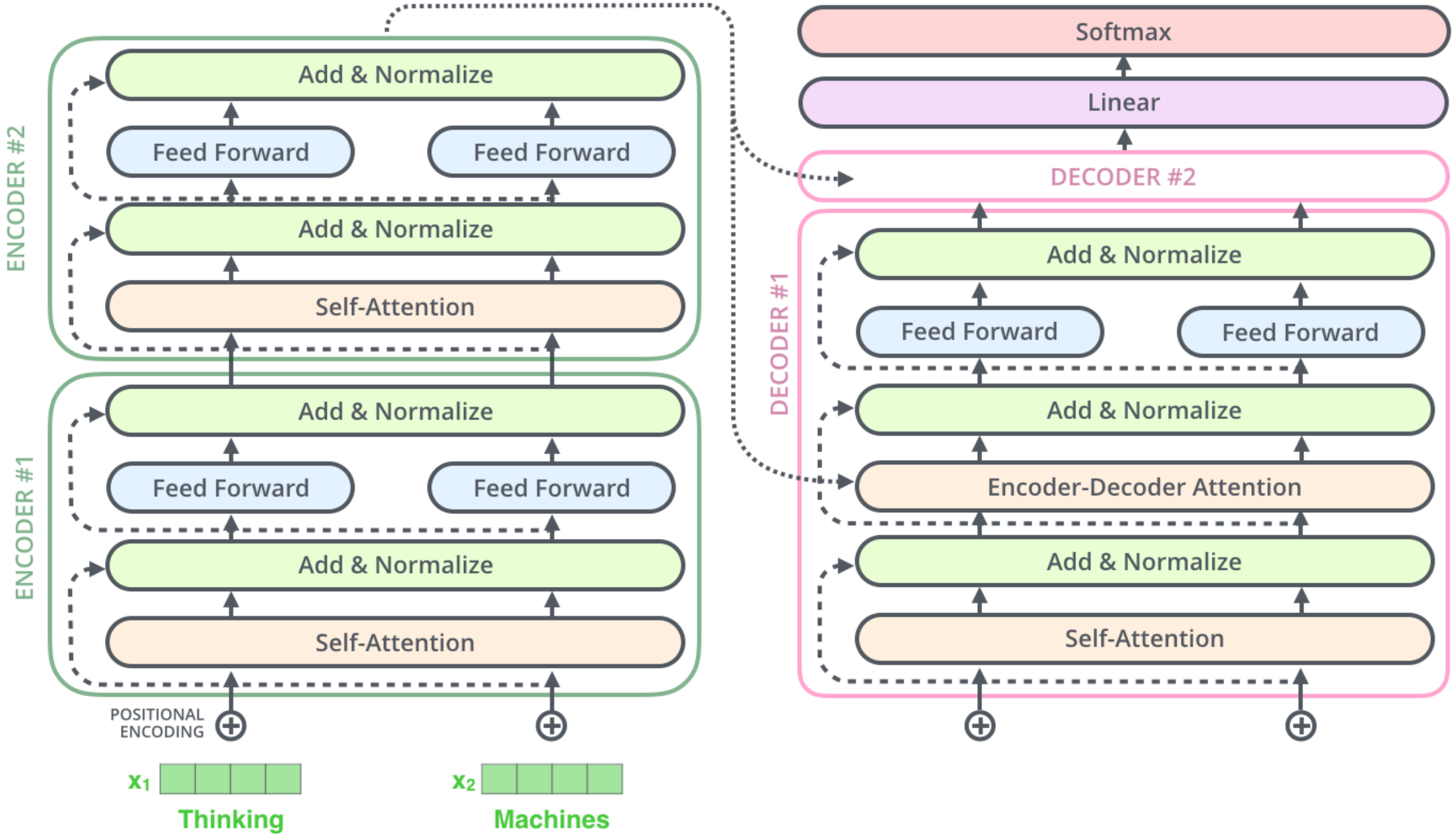
Which word in our vocabulary is associated with this index?

am

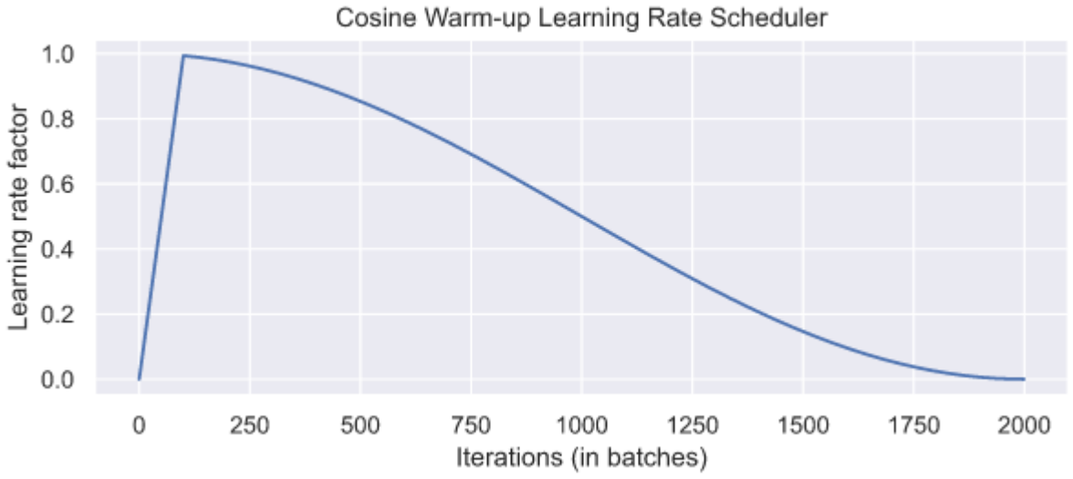
Get the index of the cell with the highest value (argmax)

5




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

Обучение трансформера: Learning rate warm-up



трансформеры обучают
с такой программой изменения LR

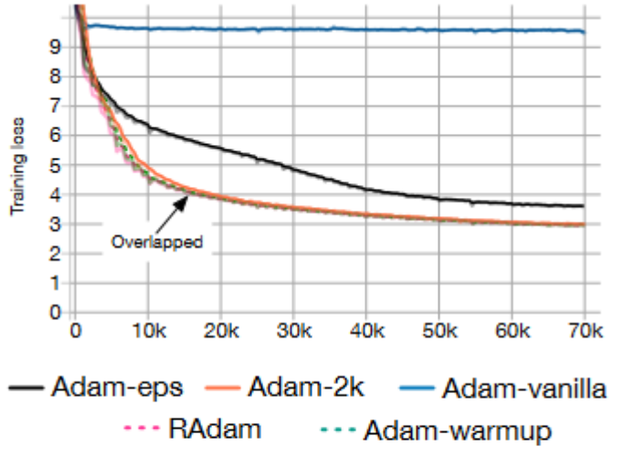


Figure 1: Training loss v.s. # of iterations of Transformers on the De-En IWSLT'14 dataset.

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

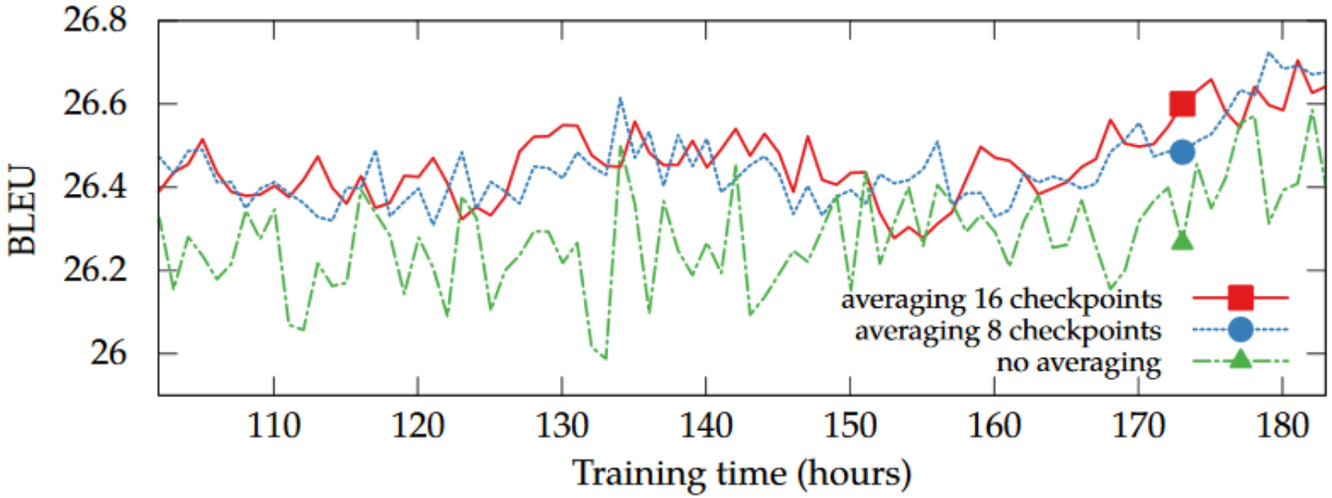


Figure 10: Effect of checkpoint averaging. All trained on 6 GPUs.

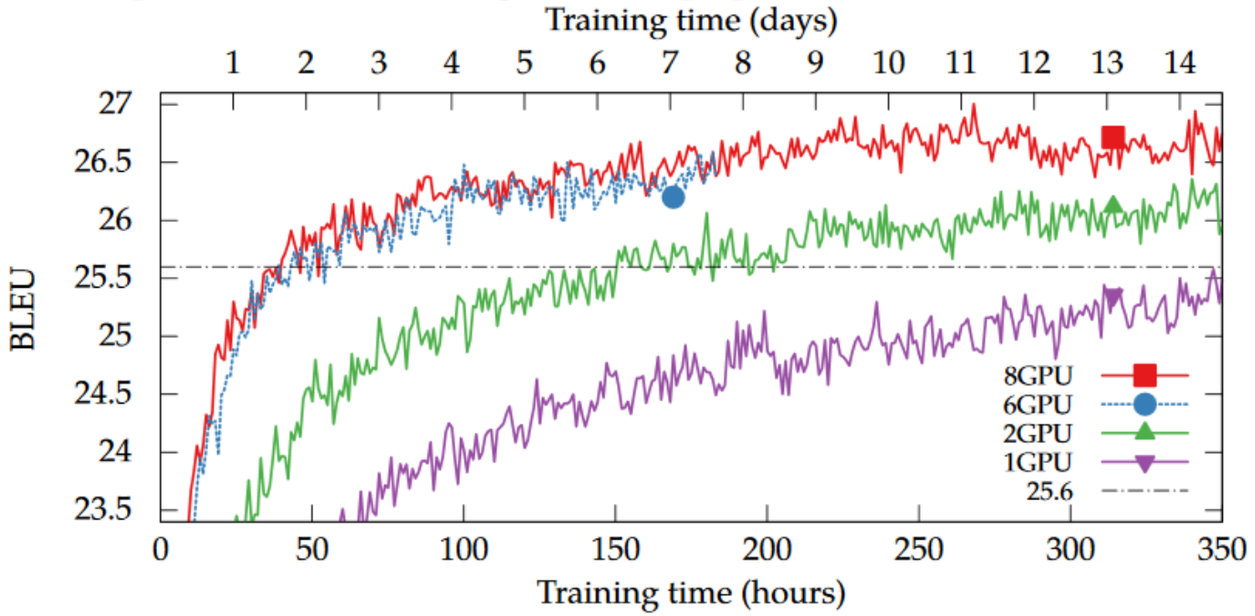


Figure 9: Effect of the number of GPUs. BLEU=25.6 is marked with a black line.

Особенности обучения трансформера

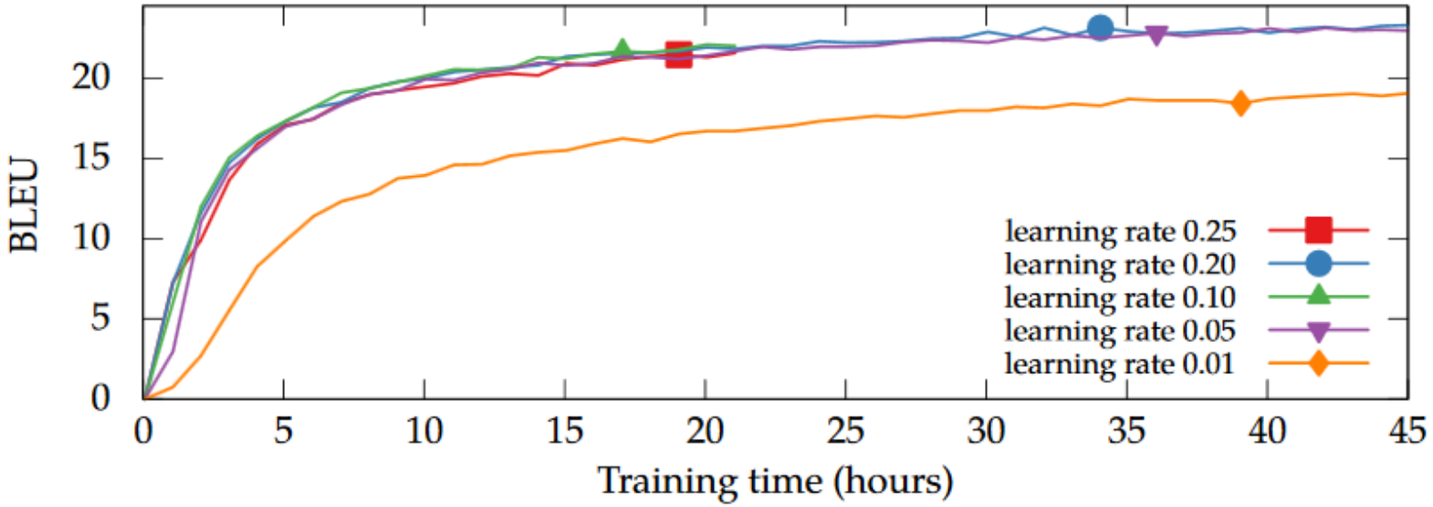


Figure 7: Effect of the learning rate on a single GPU. All trained on CzEng 1.0 with the default batch size (1500) and warmup steps (16k).

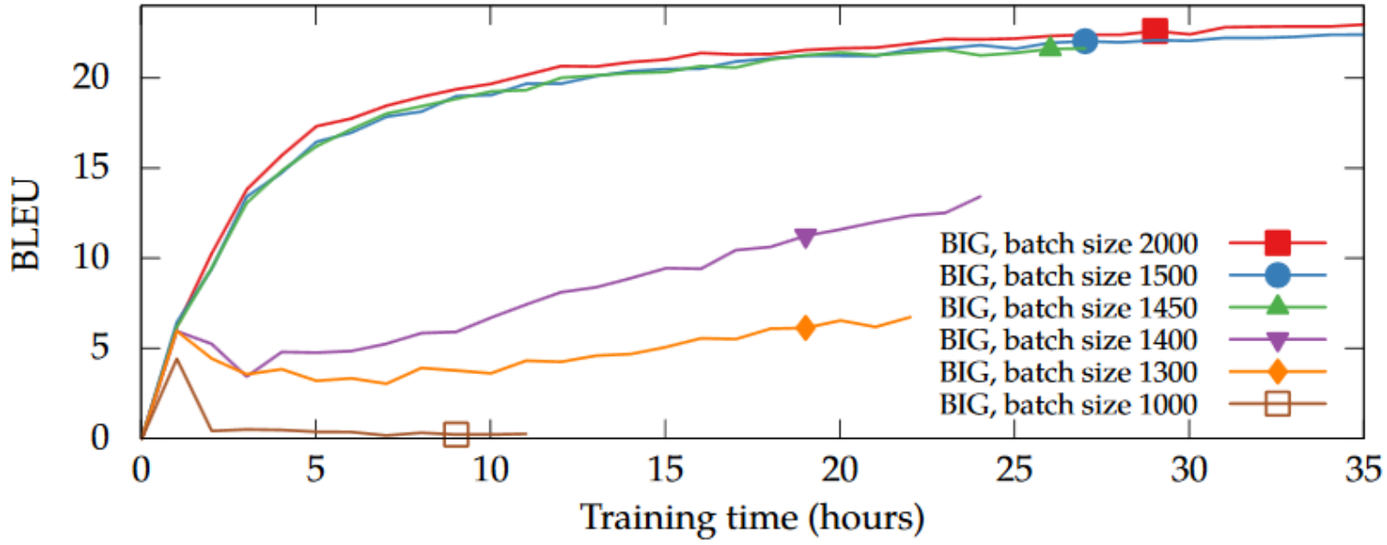


Figure 6: Effect of the batch size with the BIG model. All trained on a single GPU.

Особенности обучения трансформера

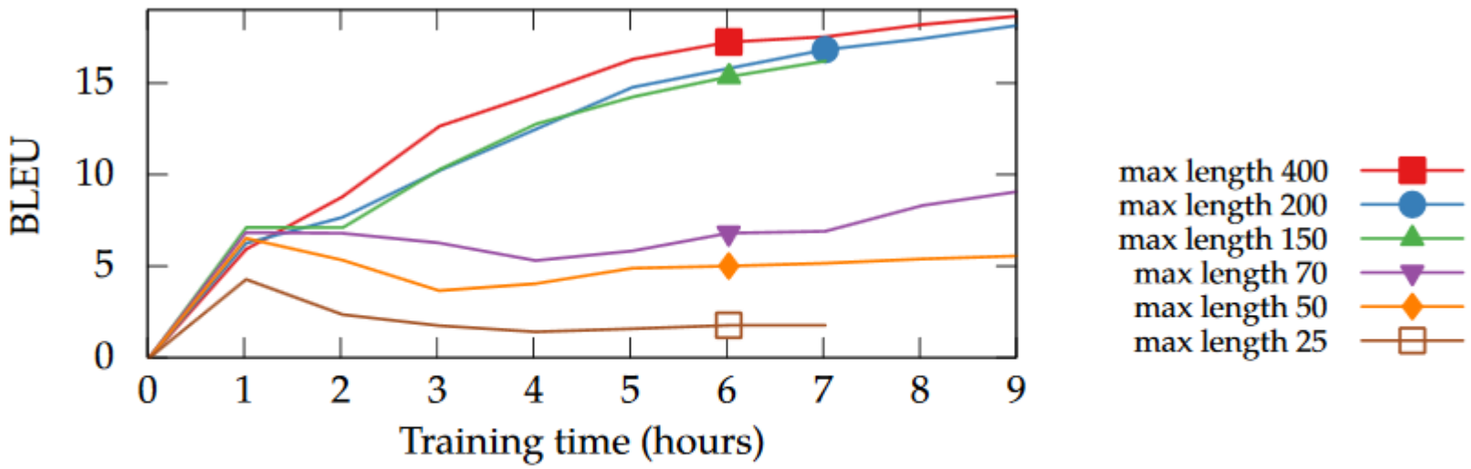


Figure 4: Effect of restricting the training data to various `max_length` values. All trained on a single GPU with the BIG model and `batch_size`=1500. An experiment without any `max_length` is not shown, but it has the same curve as `max_length`=400.

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

визуализация обучения:

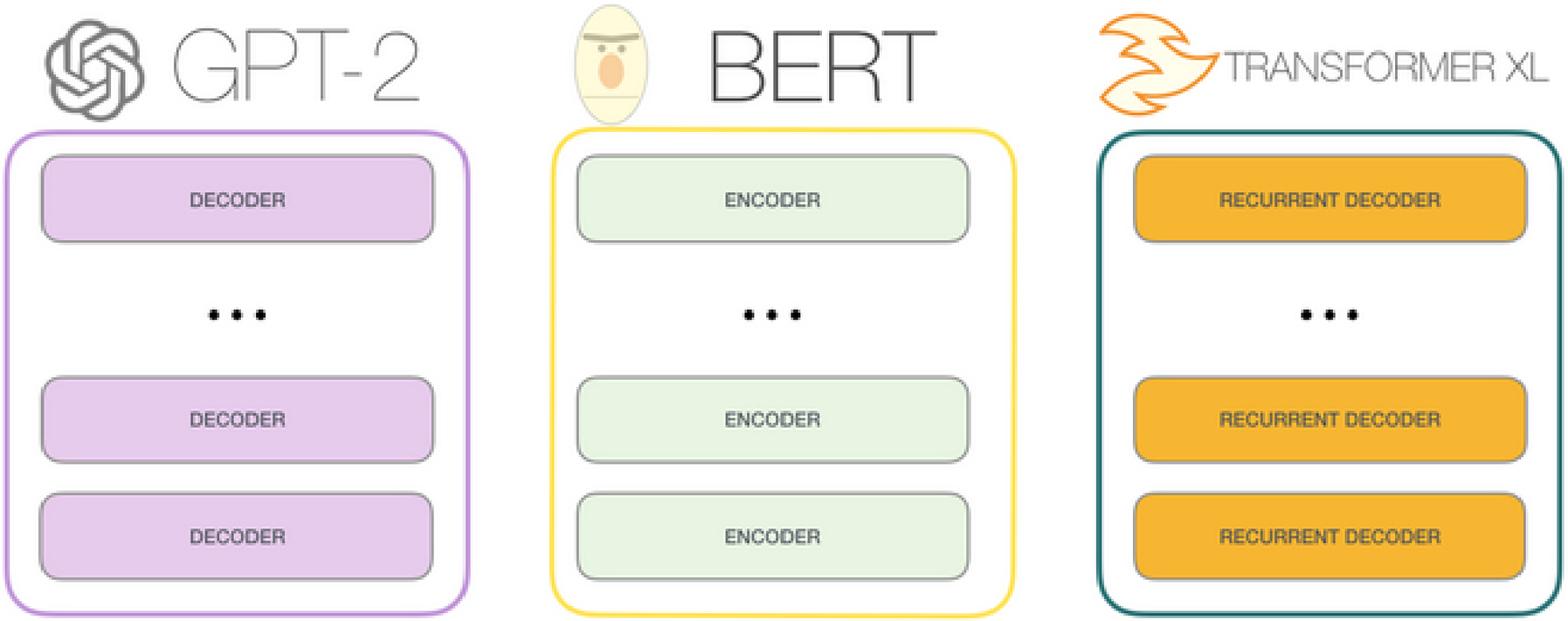
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

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

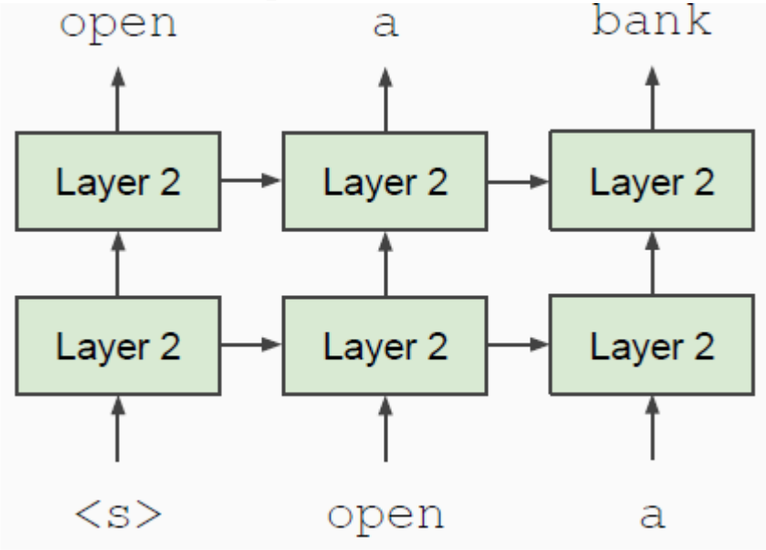
J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova «Bert: Pre-training of deep bidirectional transformers for language understanding». 2018. https://arxiv.org/abs/1810.04805

BERT: учёт контекста

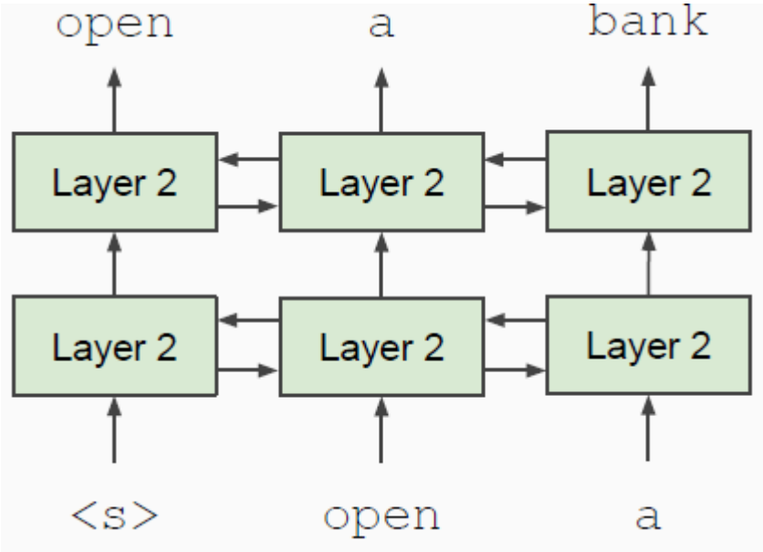
обычно левый или правый контекст, а надо «двусторонний»

главный подводный камень – слова увидят себя

односторонний контекст



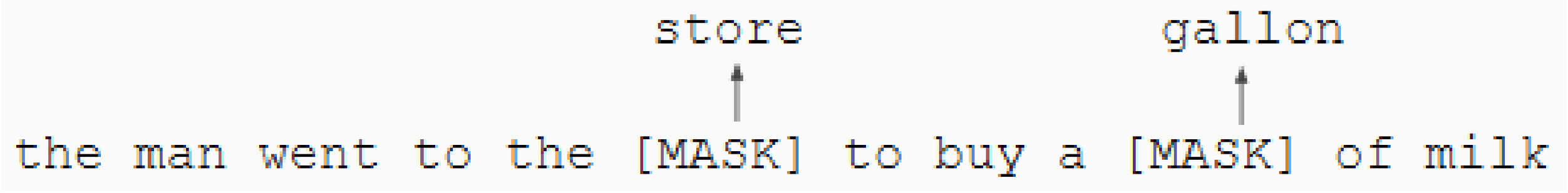
двусторонний контекст



слова видят друг друга

BERT: Mask language model (MLM)

решение: маскировать k% слов и предсказывать их



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)

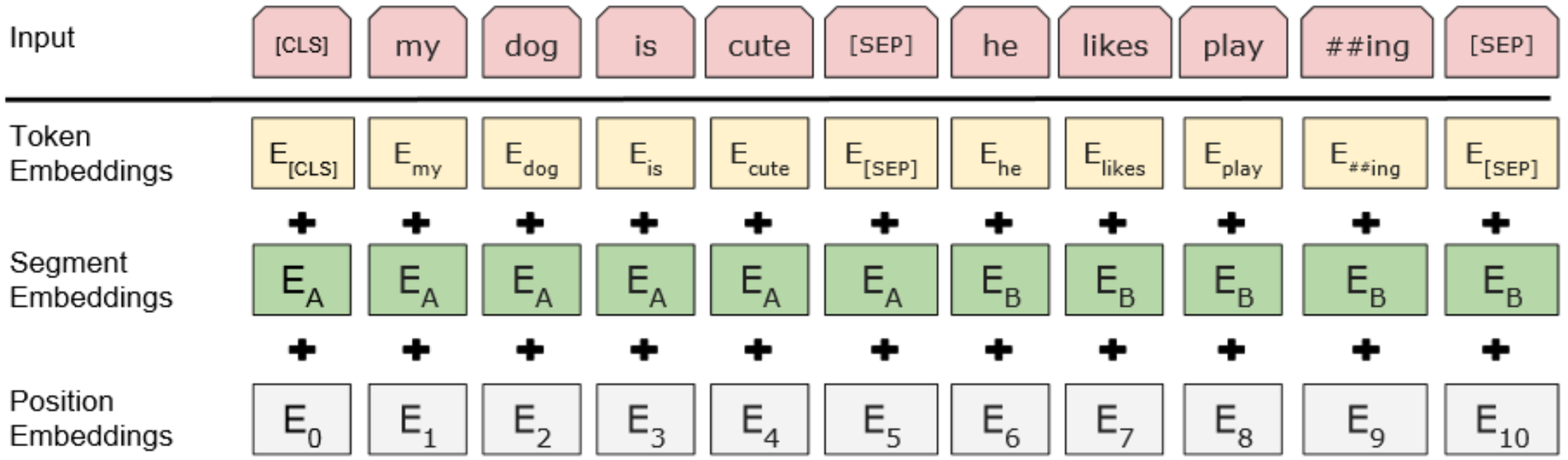


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

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

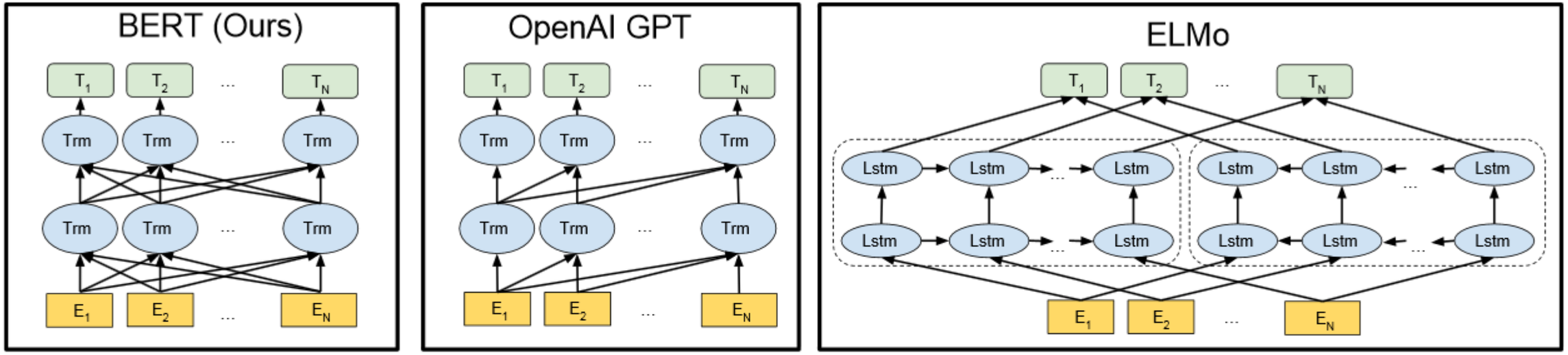


Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT 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

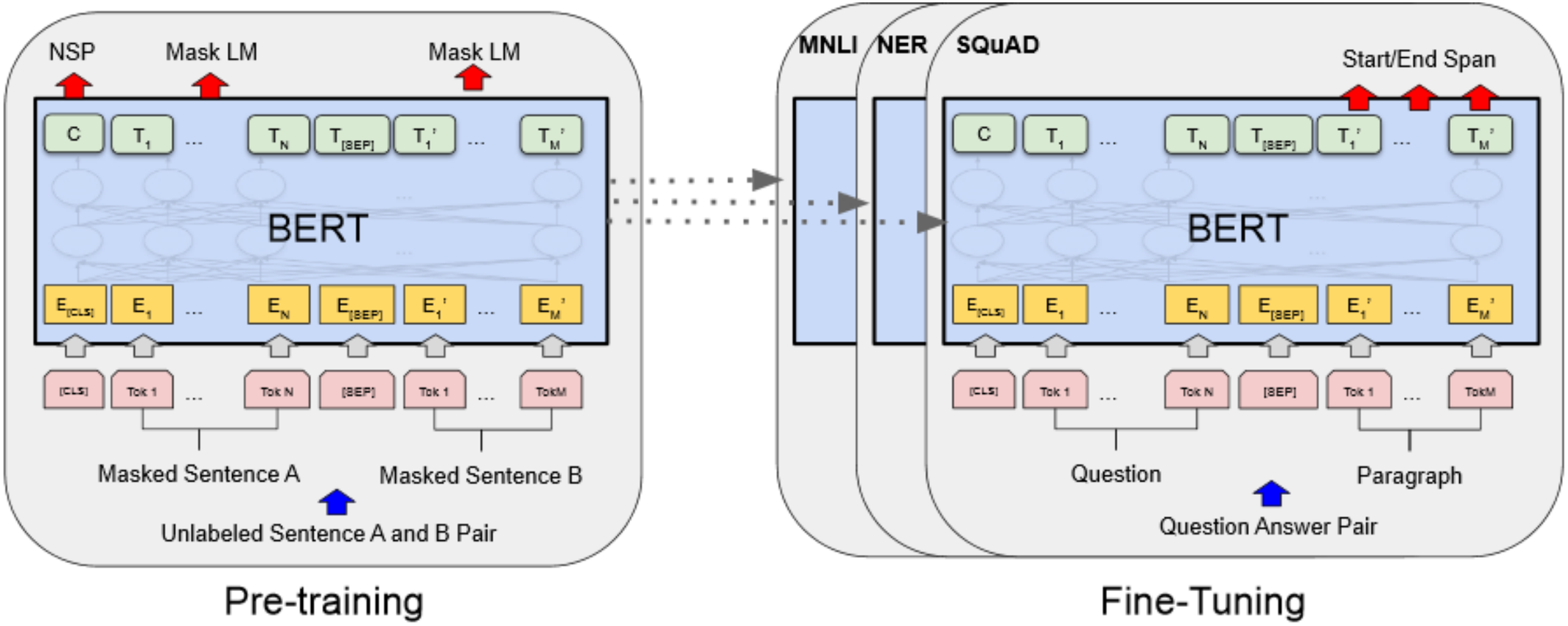
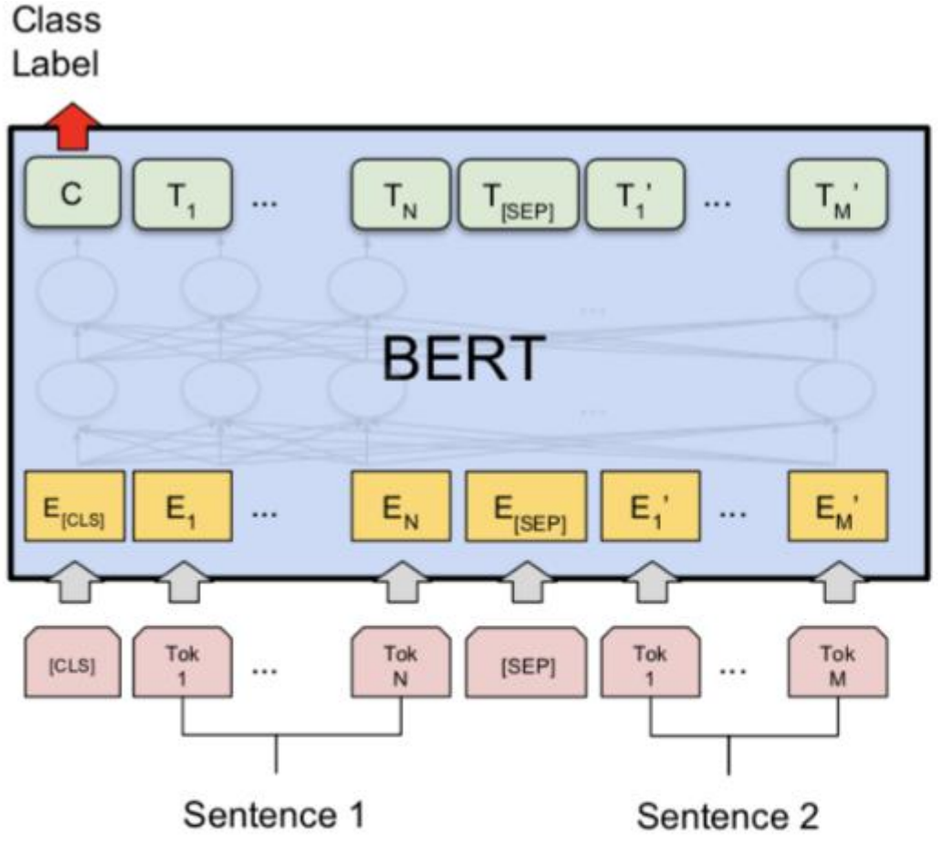
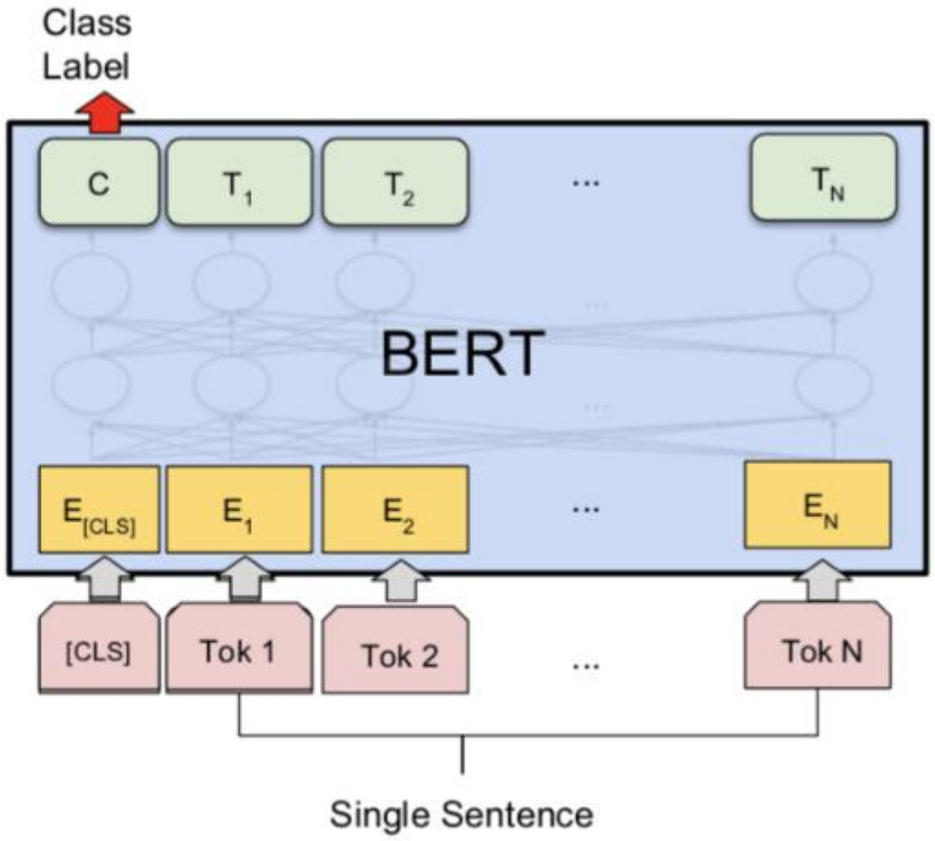


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are 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



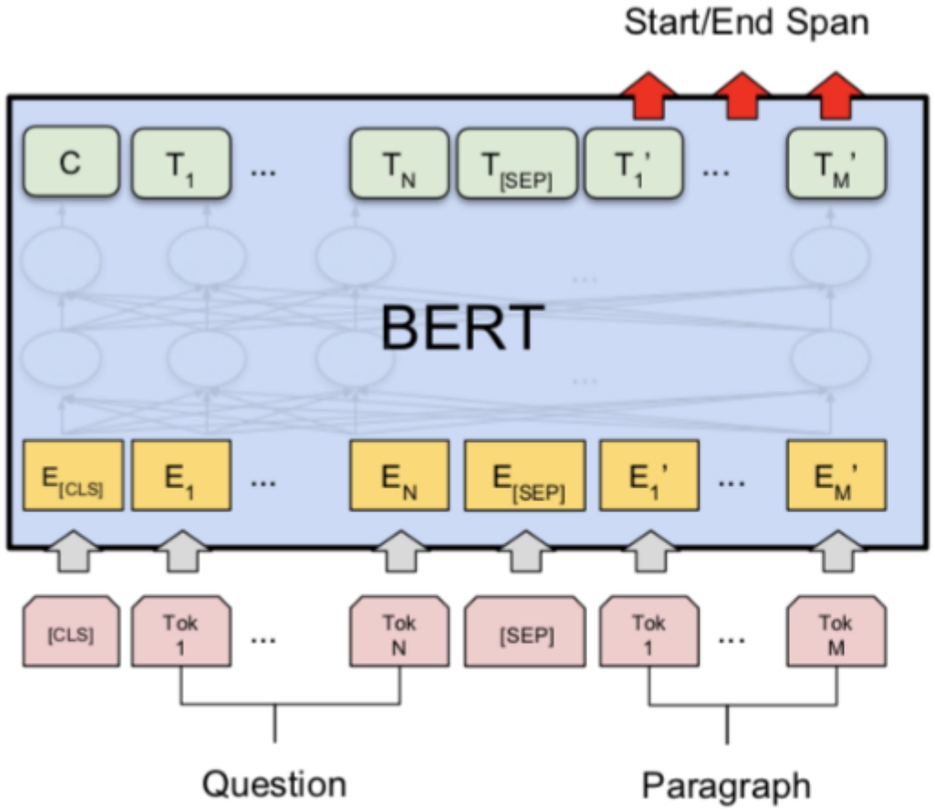
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



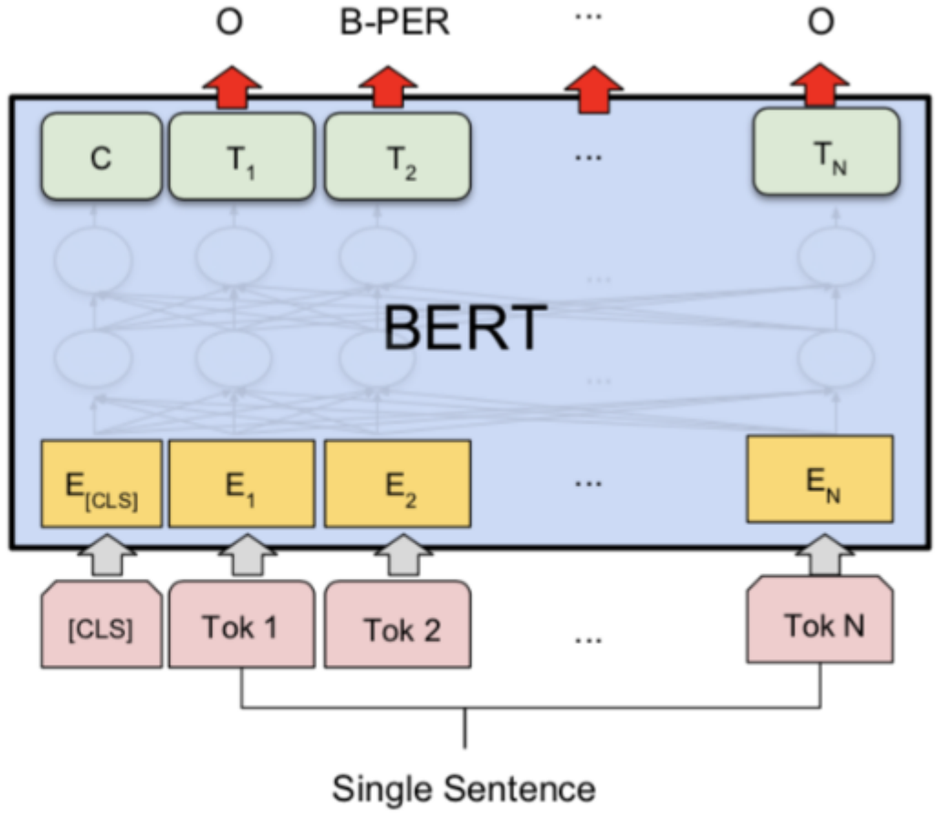
(b) Single Sentence Classification Tasks:
SST-2, CoLA

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



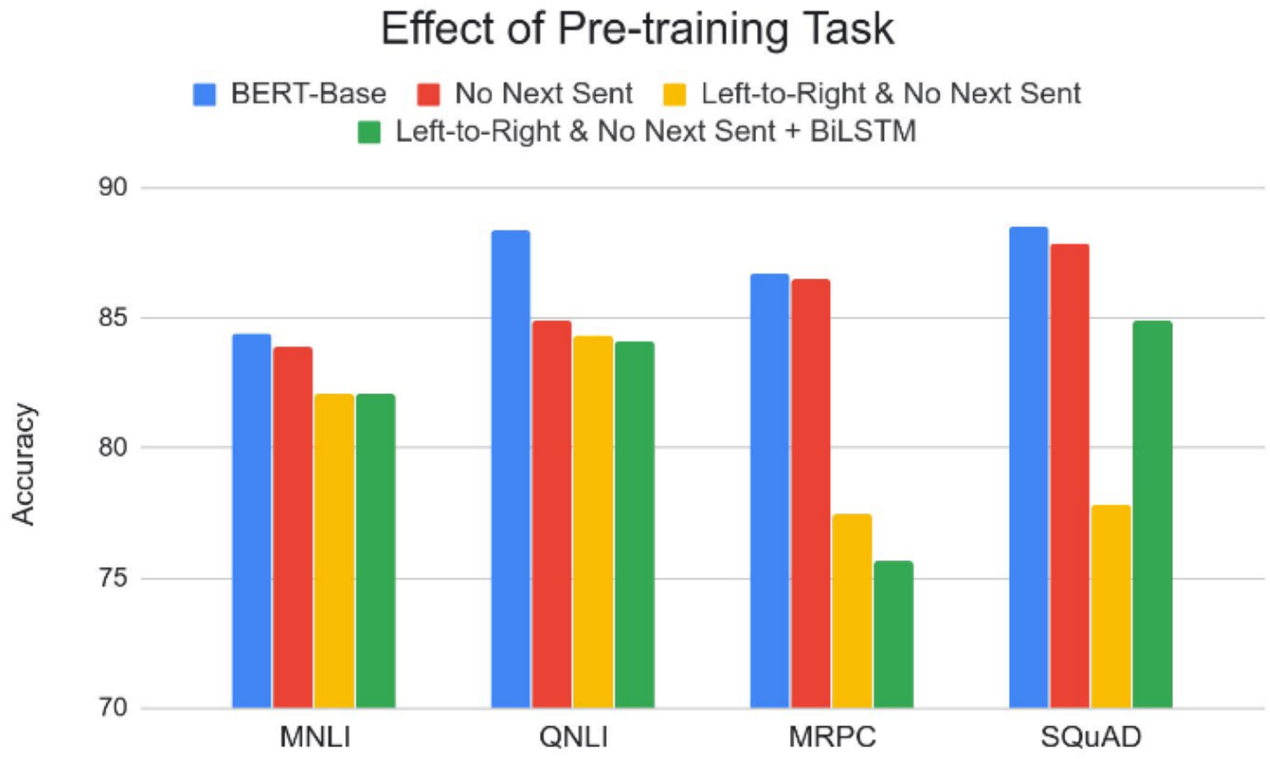
(c) Question Answering Tasks:
SQuAD v1.1



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

QA: ответ «кусок 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



Tasks	Dev Set MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	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 $\text{BERT}_{\text{BASE}}$ 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.

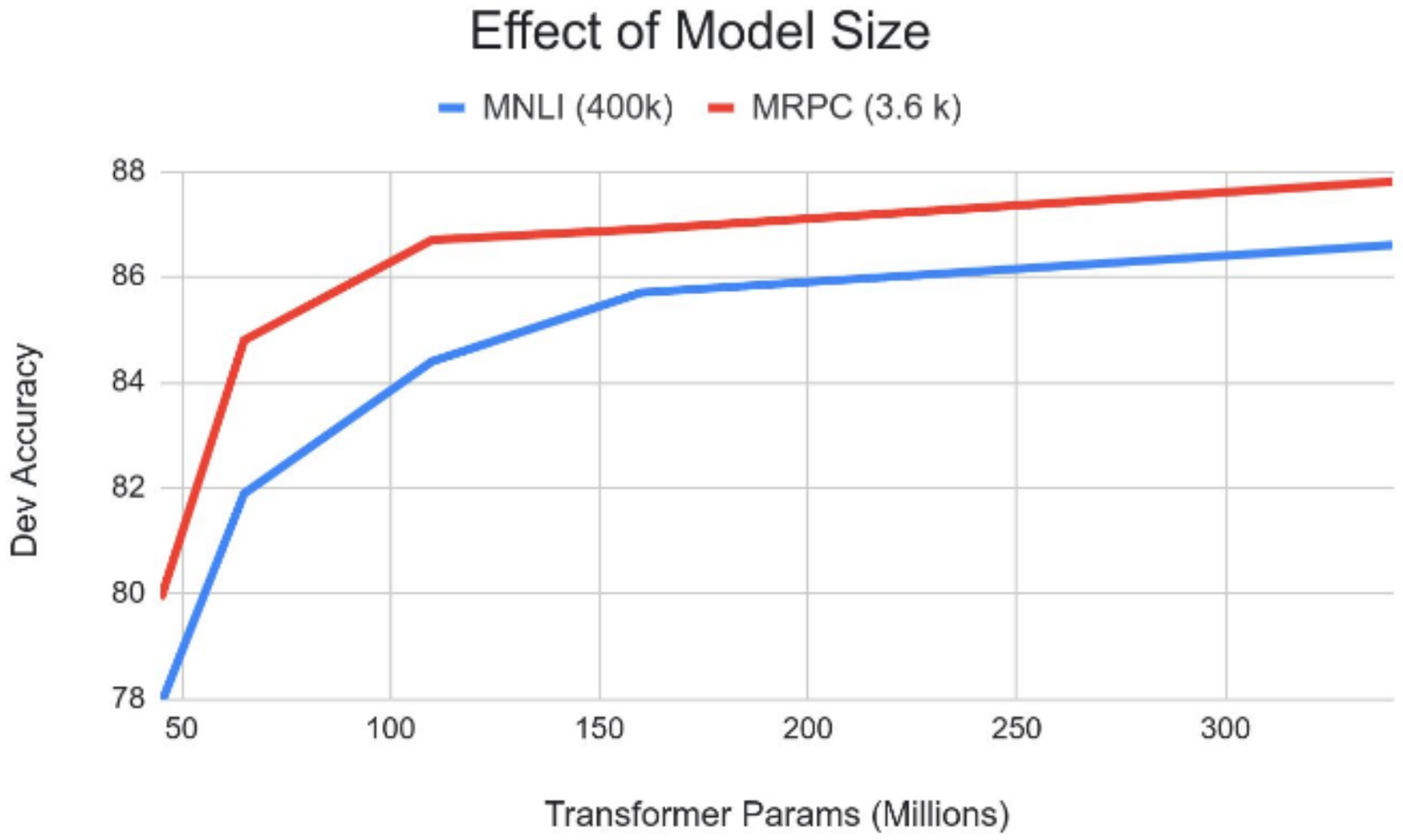
- **Для каких-то задач важнее Masked LM, для каких-то NSP**
- **на SQuAD плохи модели «слева-направо»**

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
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

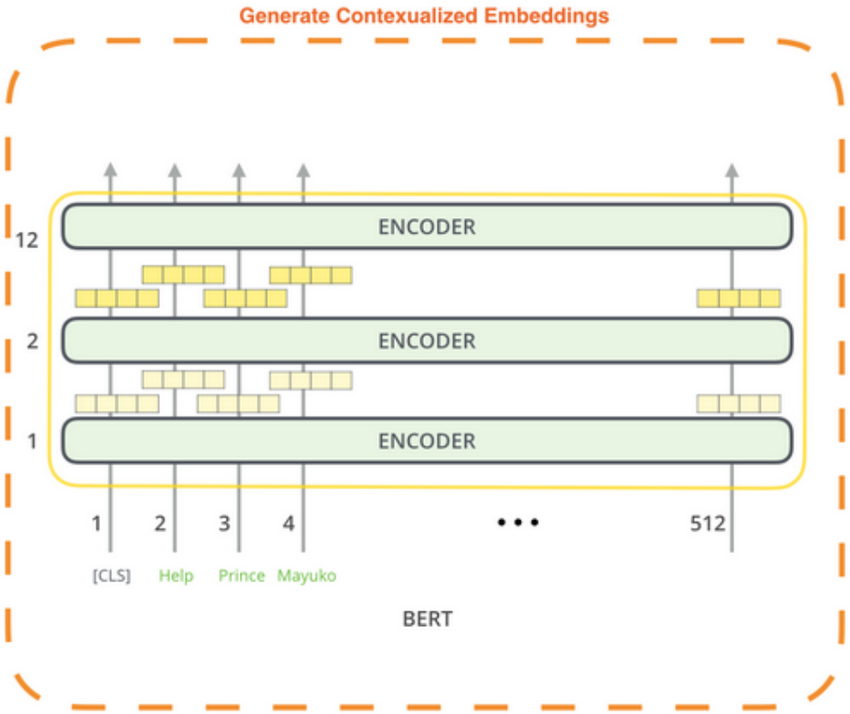
Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI 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.

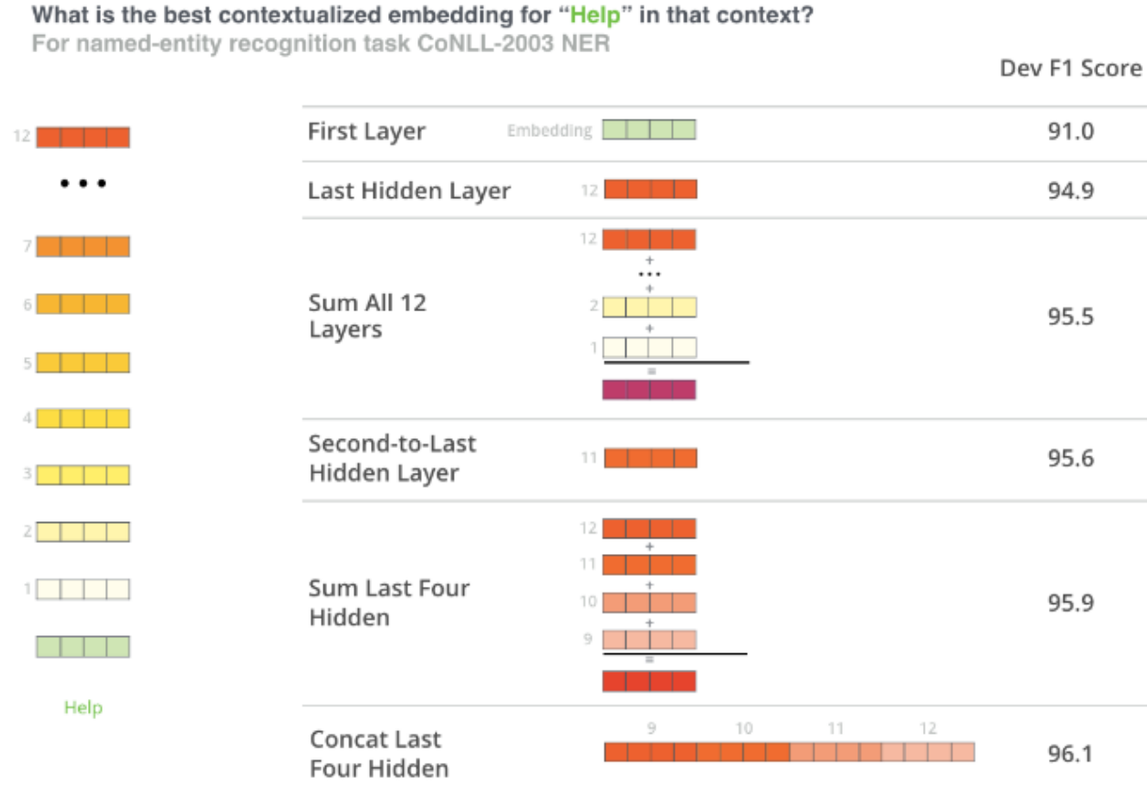
BERT: эффект размера модели



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

Представления слов с помощью BERT





как в ElMo можно комбинировать состояния разных уровней

После выхода BERT….

MT-DNN https://arxiv.org/abs/1901.11504
ERNIE https://arxiv.org/abs/1905.07129
XLNet https://arxiv.org/abs/1906.08237
KERMIT https://arxiv.org/abs/1906.01604
ERNIE 2.0 https://arxiv.org/abs/1907.12412
RoBERTa https://arxiv.org/abs/1907.11692
SpanBERT https://arxiv.org/abs /1907.11692
ALBERT https://arxiv.org/abs/1909.11942
T5 https://arxiv.org/abs/1910.10683
ELECTRA https://arxiv.org/abs/2003.10555

RoBERTa: A Robustly Optimized BERT Pretraining Approach

https://arxiv.org/abs/1907.11692

качество BERT ↑

первое место – GLUE лидерборд

SOTA на многих NLP задачах

Предложили способы тренировать BERT
без изменения самой архитектуры модели

Обнаружили, что BERT был «существенно недотренирован»!

код

https://github.com/pytorch/fairseq

Различия с оригинальным BERT:

обучать дольше, с большими батчами, на большем объёме

- train $\times$10: 16GB $\rightarrow$ 160GB
- мини-батч: 256 сэмплов $\rightarrow$ 8к
- более длинные предложения

- убрали использование NSP (next sentence prediction) objective
- динамический маскинг для каждого сэмпла
- улучшили BPE-энкодинг переведя базу с юникода на байты

Обучение

1024 карт Nvidia V100 (128 DGX-1 серверов)
в течение 5 дней

Данные оригинального BERT

Wiki-корпус + BookCorpus (16GB)

Новые данный (на английском)

- **CC-News** 63 миллиона новостей за 2.5 года (76GB)
- **OpenWebText** - корпус для GPT2 – скрауленные статьи, на которые были приведены ссылки в постах на реддите минимум с тремя апвоутами (38GB)
- **Stories** - корпус историй из CommonCrawl (31GB)

Оригинальный маскинг в BERT

в каждом сэмпле маскируются 15% токенов,
и предсказываются по немаскированным

Маска генерируется для каждого сэмпла один раз при препроцессинге и не меняется.

Обучающая выборка множится ×10,
поэтому для каждого сэмпла можно встретить 10 масок
при обучении

Динамический маскинг

формируем маску, когда последовательность поступает на вход

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for $\text{BERT}_{\text{BASE}}$. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019).

тут различают заявленное в BERTe и свою реализацию

NSP (next sentence prediction) loss

Просто убрали и посмотрели…
то что не нужен заметили ещё авторы XLNet

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Table 2: Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. We report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over five random initializations (seeds). Results for $\text{BERT}_{\text{BASE}}$ and $\text{XLNet}_{\text{BASE}}$ are from Yang et al. (2019).

Увеличение размера мини-батча

«чем больше мини-батч, тем лучше финальные результаты трейна»

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

Более длинные предложения

Каждый вход – полные предложения длины не больше 512 токенов

BPE (Byte Pair Encoding)

– кодировка частых слов и букв

Оригинальный BERT:
юникод символы как subword units

В GPT2 предложили использовать не юникод символы, а байты
(если использовать BPE словарь размером 50k,
то у нас не будет unknown токенов)

Размер модели BERT по сравнению с оригинальным вырос на 5-10%:
+ 15M параметров base
+ 20M large

Результаты

Для сравнения используют BERT-large и XLNet-large

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$\text{BERT}_{\text{LARGE}}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$\text{XLNet}_{\text{LARGE}}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Table 5: Results on GLUE. All results are based on a 24-layer architecture. $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of *single-task* models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

new SOTA on 4/9 of the GLUE tasks: MNLI, QNLI, RTE and STS-B

Результаты GLUE

9 datasets (natural language understanding systems)
6 single-sentence / sentence-pair classification

Использовали single-task файнтюнинг,
в отличии от многих других подходов из топа GLUE бенчмарка,
которые делают multi-task файнтюнинг

RACE (The ReAding Comprehension from Examinations)

Экзамены по английскому в Китае (28,000 passages ~ 100,000) текста,
вопрос и 4 варианта ответа
надо выбрать правильный

конкатенируют текст, вопрос и ответ → BERT → репрезентация из CLF токена → один полносвязанный слой и предсказывают является ли ответ правильным (×4 раза – для каждого варианта ответа)

ALBERT = A Lite BERT

существенное «сжатие» модели
обучается в 1.7 раз быстрее
параметров – /18

«лёгкий» в смысле параметров, но не скорости!

Zhenzhong Lan et. al. «ALBERT: A Lite BERT for Self-supervised Learning of Language Representations» // https://arxiv.org/abs/1909.11942

ALBERT: факторизация матрицы параметров

В BERT размерность пространства представлений (WordPiece) E = размерность пространства состояний (hidden state) H
но по смыслу они разные – контекстно-независмое / зависимое

пусть размер словаря (vocabulary) = V

V×H-матрица раскладывается V×E • E×H

ALBERT: Sentence-Order Prediction (SOP)

Задача next sentence prediction (NSP) оказалась лёгкой…
вместо этого «предложения в правильном порядке» / «в неправильном»
– всё из одного документа

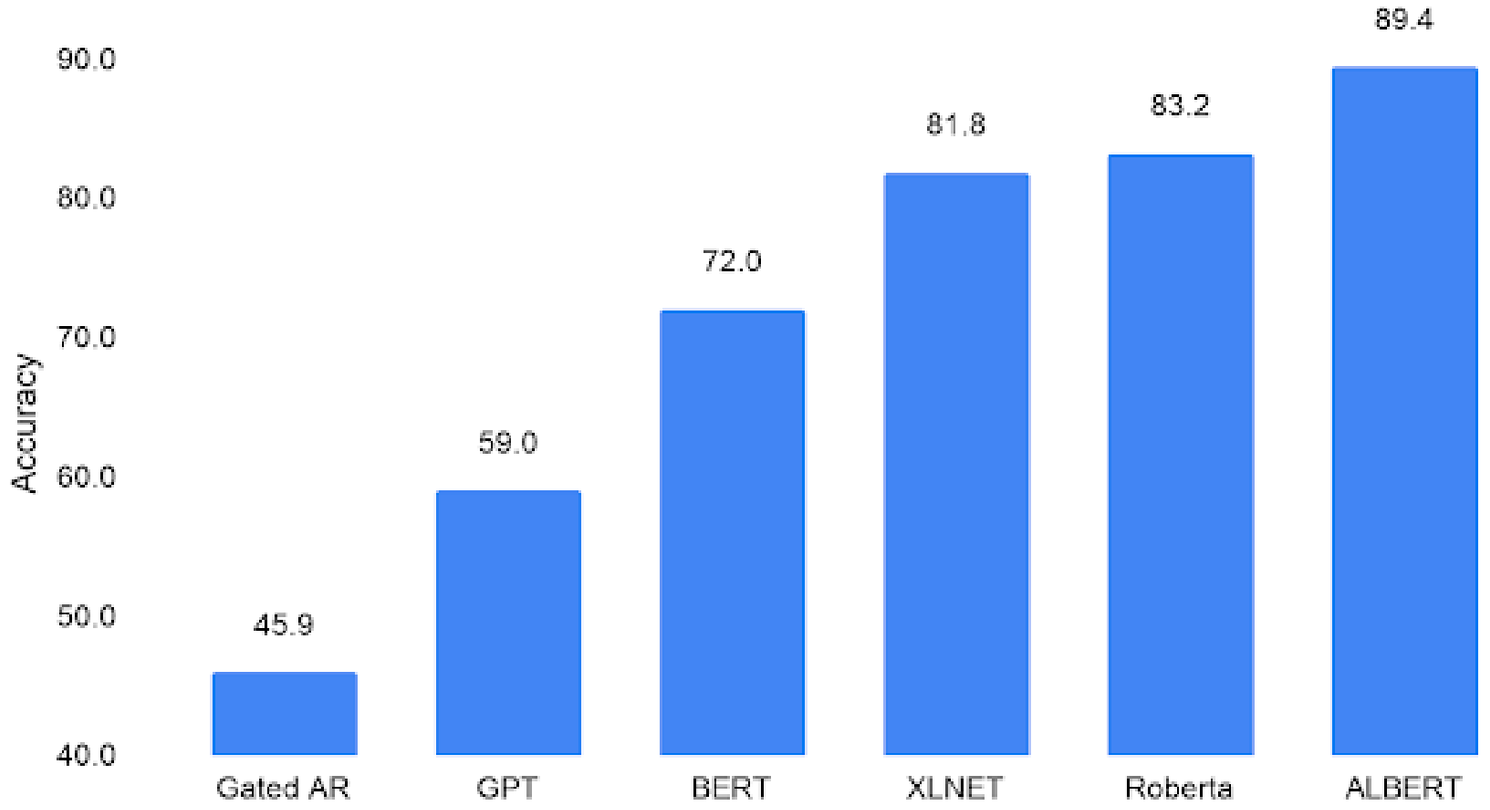
ALBERT: Cross-layer Parameter Sharing

немного уменьшает качество, но увеличивает компактность

разные техники:

- **только в feed-forward part**
- **только во внимании**
- **все параметры**

ALBERT: RACE (задача на понимание текстов из экзаменов SAT)



T5: Text-To-Text Transfer Transformer (Google)

1) интерфейс текст → текст

а не как принято в BERT – метка или отрезок текста на выходе

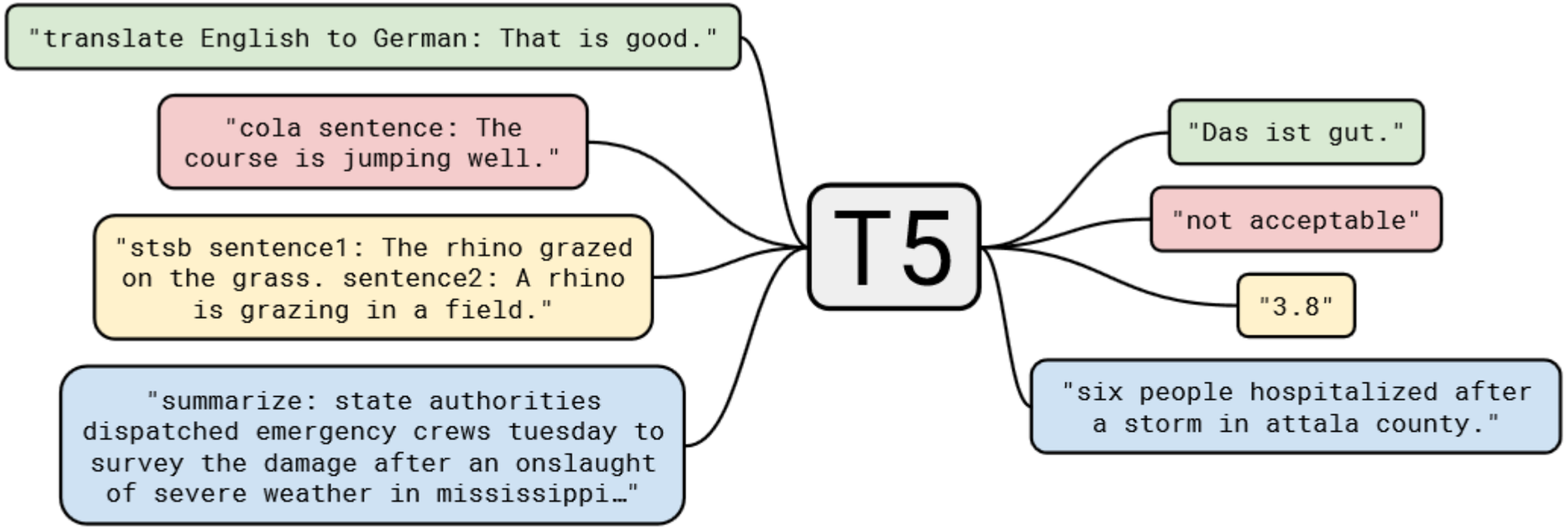


Figure 1: A diagram of our text-to-text framework. Every task we consider – including translation, question answering, and classification – is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "Text-to-Text Transfer Transformer".

T5: Text-To-Text Transfer Transformer

2) A Large Pre-training Dataset (C4)

Для предтренировки нужен хороший и большой датасет

Wikipedia – хороший, но однообразный по стилю и «небольшой»

the Common Crawl web scrapes – большой и разнообразный, но низкого качества

сделали его очищенную версию

- **удаление дубликатов**
- **удаление неполных предложений**
- **удаление оскорблений и шума**

Dataset	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

Table 8: Performance resulting from pre-training on different datasets. The first four variants are based on our new C4 dataset.

https://www.tensorflow.org/datasets/catalog/c4

T5: Text-To-Text Transfer Transformer

3) систематическое изучение «Transfer Learning Methodology»

- **архитектуры (model architectures)**

лучше кодировщик-декодировщик, а не чистый декодировщик в LM

- **функции преднастройки (pre-training objectives)**

подтвердили, что надо «заполнять пропуски»

- **неразмеченные данные (unlabeled datasets)**

лучше предобучать на том же домене, но если данных мало, то очень плохо

- **стратегии обучения (training strategies)**

multitask learning ~ pre-train-then-fine-tune approach

- **масштабирование**

Colin Raffel «Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer» // https://arxiv.org/abs/1910.10683

T5: Text-To-Text Transfer Transformer

4) финальная модель

11 млрд. параметров

SOTA: GLUE, SuperGLUE, SQuAD, CNN/Daily Mail

За базу – BERT-BASE

T5: Text-To-Text Transfer Transformer

	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline average	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Baseline standard deviation	0.235	0.065	0.343	0.416	0.112	0.090	0.108
No pre-training	66.22	17.60	50.31	53.04	25.86	**39.77**	24.04

Table 1: Average and standard deviation of scores achieved by our baseline model and training procedure. For comparison, we also report performance when training on each task from scratch (i.e. without any pre-training) for the same number of steps used to fine-tune the baseline model. All scores in this table (and every table in our paper except Table 14) are reported on the validation sets of each data set.

что даёт предобучение

T5: Text-To-Text Transfer Transformer

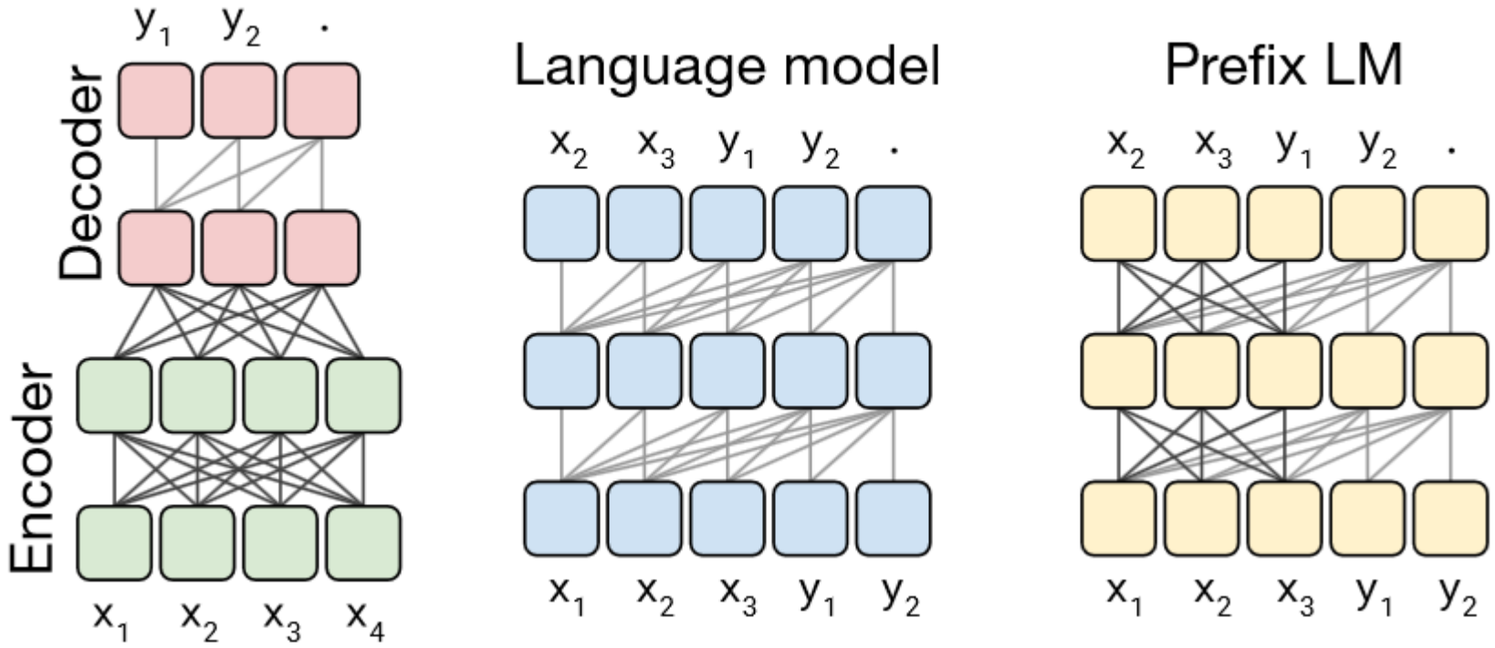


Figure 4: Schematics of the Transformer architecture variants we consider. In this diagram, blocks represent elements of a sequence and lines represent attention visibility. Different colored groups of blocks indicate different Transformer layer stacks. Dark grey lines correspond to fully-visible masking and light grey lines correspond to causal masking. We use "." to denote a special end-of-sequence token that represents the end of a prediction. The input and output sequences are represented as x and y respectively. Left: A standard encoder-decoder architecture uses fully-visible masking in the encoder and the encoder-decoder attention, with causal masking in the decoder. Middle: A language model consists of a single Transformer layer stack and is fed the concatenation of the input and target, using a causal mask throughout. Right: Adding a prefix to a language model corresponds to allowing fully-visible masking over the input.

T5: Text-To-Text Transfer Transformer

Architecture	Objective	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	Denoising	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	Denoising	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	Denoising	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	Denoising	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	Denoising	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39
Encoder-decoder	LM	$2P$	M	79.56	18.59	76.02	64.29	26.27	39.17	26.86
Enc-dec, shared	LM	P	M	79.60	18.13	76.35	63.50	26.62	39.17	27.05
Enc-dec, 6 layers	LM	P	$M/2$	78.67	18.26	75.32	64.06	26.13	38.42	26.89
Language model	LM	P	M	73.78	17.54	53.81	56.51	25.23	34.31	25.38
Prefix LM	LM	P	M	79.68	17.84	76.87	64.86	26.28	37.51	26.76

Table 2: Performance of the different architectural variants described in Section 3.2.2. We use P to refer to the number of parameters in a 12-layer base Transformer layer stack and M to refer to the FLOPs required to process a sequence using the encoder-decoder model. We evaluate each architectural variant using a denoising objective (described in Section 3.1.4) and an autoregressive objective (as is commonly used to train language models).

T5: Text-To-Text Transfer Transformer

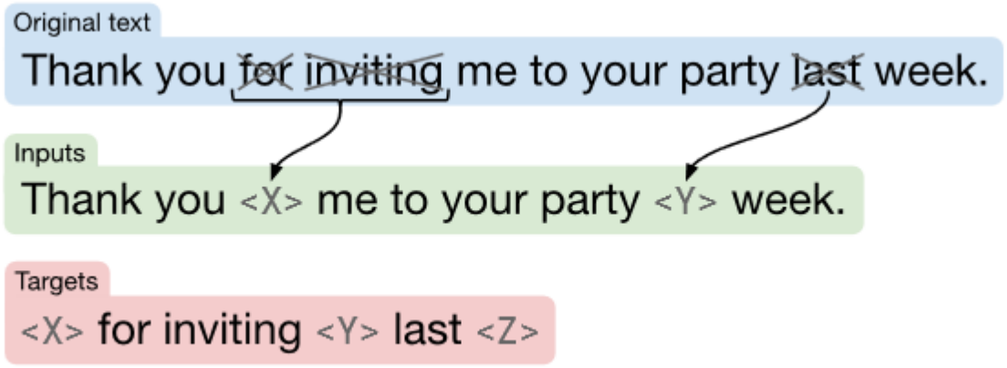


Figure 2: Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>) that is unique over the example. Since "for" and "inviting" occur consecutively, they are replaced by a single sentinel <X>. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.

использованное маскирование

T5: Text-To-Text Transfer Transformer

Objective	Inputs	Targets
Prefix language modeling	Thank you for inviting	me to your party last week .
BERT-style	Thank you <M> <M> me to your party apple week .	*(original text)*
Deshuffling	party me for your to . last fun you inviting week Thank	*(original text)*
I.i.d. noise, mask tokens	Thank you <M> <M> me to your party <M> week .	*(original text)*
I.i.d. noise, replace spans	Thank you <X> me to your party <Y> week .	<X> for inviting <Y> last <Z>
I.i.d. noise, drop tokens	Thank you me to your party week .	for inviting last
Random spans	Thank you <X> to <Y> week .	<X> for inviting me <Y> your party last <Z>

Table 3: Examples of inputs and targets produced by some of the unsupervised objectives we consider applied to the input text "Thank you for inviting me to your party last week ." Note that all of our objectives process *tokenized* text. For this particular sentence, all words were mapped to a single token by our vocabulary. We write *(original text)* as a target to denote that the model is tasked with reconstructing the entire input text. <M> denotes a shared mask token and <X>, <Y>, and <Z> denote sentinel tokens that are assigned unique token IDs. The BERT-style objective (second row) includes a corruption where some tokens are replaced by a random token ID; we show this via the greyed-out word apple.

T5: Text-To-Text Transfer Transformer

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Prefix language modeling	80.69	18.94	77.99	65.27	**26.86**	39.73	**27.49**
BERT-style [Devlin et al., 2018]	**82.96**	**19.17**	**80.65**	**69.85**	**26.78**	**40.03**	**27.41**
Deshuffling	73.17	18.59	67.61	58.47	26.11	39.30	25.62

Table 4: Performance of the three disparate pre-training objectives described in Section 3.3.1.

Objective	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
BERT-style [Devlin et al., 2018]	82.96	19.17	**80.65**	69.85	26.78	**40.03**	27.41
MASS-style [Song et al., 2019]	82.32	19.16	80.10	69.28	26.79	**39.89**	27.55
★ Replace corrupted spans	83.28	**19.24**	**80.88**	**71.36**	**26.98**	39.82	**27.65**
Drop corrupted tokens	**84.44**	**19.31**	**80.52**	68.67	**27.07**	39.76	**27.82**

Table 5: Comparison of variants of the BERT-style pre-training objective. In the first two variants, the model is trained to reconstruct the original uncorrupted text segment. In the latter two, the model only predicts the sequence of corrupted tokens.

Corruption rate	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
10%	**82.82**	19.00	**80.38**	69.55	**26.87**	39.28	**27.44**
★ 15%	**83.28**	19.24	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
25%	**83.00**	**19.54**	**80.96**	70.48	**27.04**	**39.83**	**27.47**
50%	81.27	19.32	79.80	70.33	**27.01**	**39.90**	**27.49**

Table 6: Performance of the i.i.d. corruption objective with different corruption rates.

Number of tokens	Repeats	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Full dataset	0	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
2^{29}	64	**82.87**	**19.19**	**80.97**	**72.03**	**26.83**	**39.74**	**27.63**
2^{27}	256	82.62	**19.20**	79.78	69.97	**27.02**	**39.71**	27.33
2^{25}	1,024	79.55	18.57	76.27	64.76	26.38	39.56	26.80
2^{23}	4,096	76.34	18.33	70.92	59.29	26.37	38.84	25.81

Table 9: Measuring the effect of artificially shrinking our C4 dataset. This results in the dataset being repeated over the course of pre-training, which may result in memorization (see Figure 6).

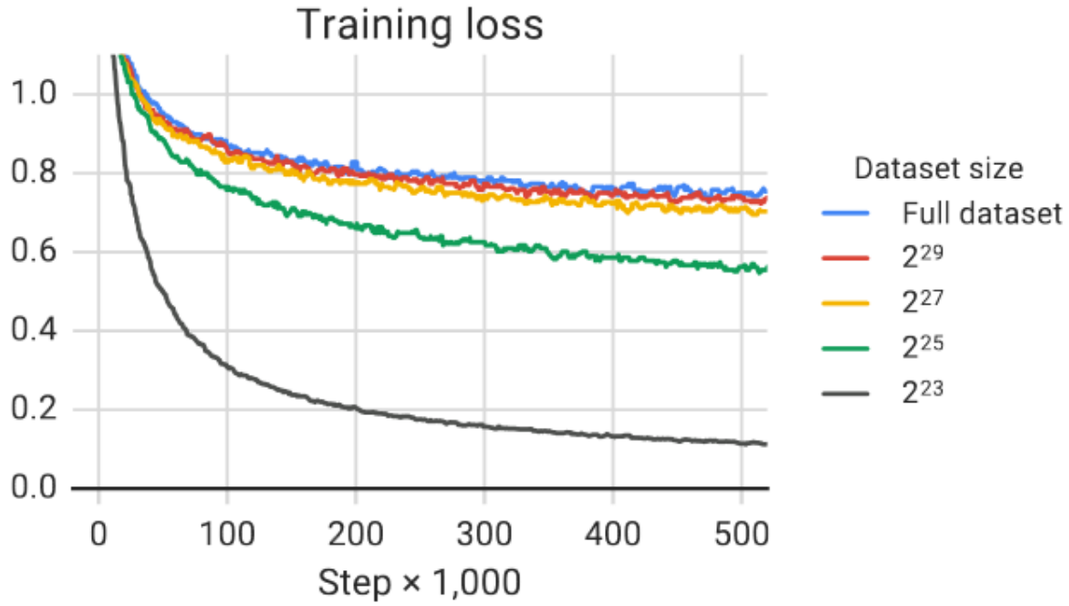


Figure 6: Pre-training loss for our original C4 dataset as well as 4 artificially truncated versions. The sizes listed refer to the number of tokens in each dataset. The four sizes considered correspond to repeating the dataset between 64 and 4,096 times over the course of pre-training. Using a smaller dataset size results in smaller training loss values, which may suggest some memorization of the unlabeled dataset.

T5: заполнение заданного числа пропусков

N=1 N=2 N=4 N=8 N=16 N=32 N=64 N=512

I love peanut butter and *jelly* sandwiches.

N=1 **N=2** N=4 N=8 N=16 N=32 N=64 N=512

I love peanut butter and *jelly on my* sandwiches.

N=1 N=2 **N=4** N=8 N=16 N=32 N=64 N=512

I love peanut butter and *jelly, which is what makes good* sandwiches.

N=1 N=2 N=4 **N=8** N=16 N=32 N=64 N=512

I love peanut butter and *jelly, Yum! You can't beat peanut butter and jelly* sandwiches.

https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html

ELECTRA = Efficiently Learning an Encoder that Classifies Token Replacements Accurately

pre-training task –«replaced token detection» (RTD)

вместо маскирования (как в BERT) заменять некоторые токены генератором

и пытаться распознать замены

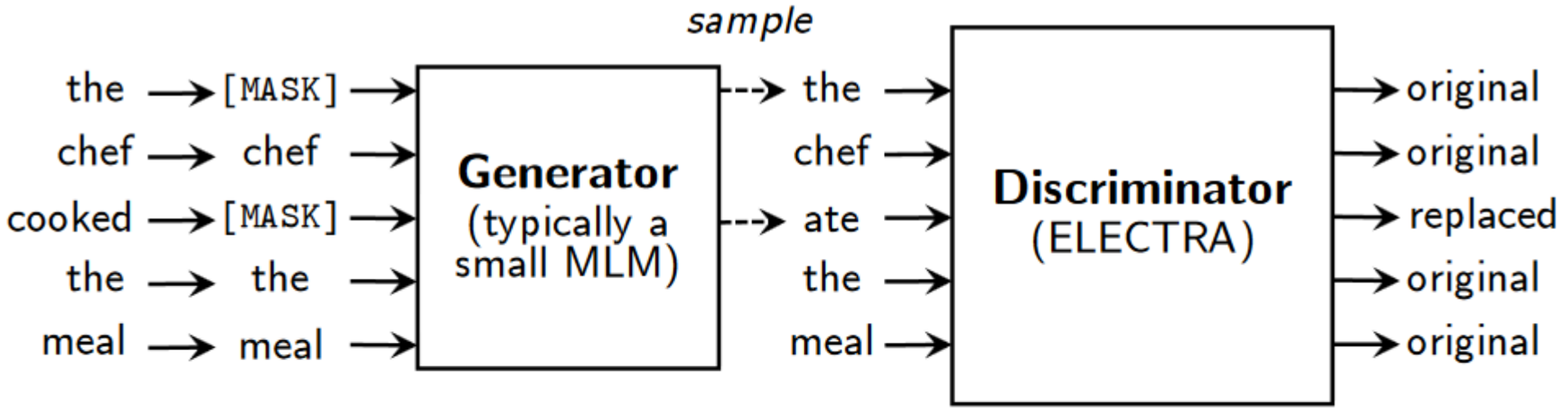


Figure 2: An overview of replaced token detection. The generator can be any model that produces an output distribution over tokens, but we usually use a small masked language model that is trained jointly with the discriminator. Although the models are structured like in a GAN, we train the generator with maximum likelihood rather than adversarially due to the difficulty of applying GANs to text. After pre-training, we throw out the generator and only fine-tune the discriminator (the ELECTRA model) on downstream tasks.

Двунаправленная модель, но учится по всем токенам

а не настраивается лишь на 15% маскированных…

ELECTRA

аналогия с GAN

генератор – masked language modeling (MLM) – маленькая модель

(в 2-4 раза меньше дискриминатора)

$$\mathcal{L}_{\text{MLM}}(\boldsymbol{x}, \theta_G) = \mathbb{E}\left(\sum_{i \in \boldsymbol{m}} -\log p_G(x_i | \boldsymbol{x}^{\text{masked}})\right)$$

$$\mathcal{L}_{\text{Disc}}(\boldsymbol{x}, \theta_D) = \mathbb{E}\left(\sum_{t=1}^{n} -\mathbb{1}(x_t^{\text{corrupt}} = x_t) \log D(\boldsymbol{x}^{\text{corrupt}}, t) - \mathbb{1}(x_t^{\text{corrupt}} \neq x_t) \log(1 - D(\boldsymbol{x}^{\text{corrupt}}, t))\right)$$

генератор и дискриминатор – трансформеры

нет MASK-токенов

weight sharing для входных эмбеддингов между генератором и дискриминатором

для снижения числа параметров

Очень быстро учится – small ELECTRA – 4 дня на одной GPU (и лучше GPT)

Clark et al «ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators», 2020 https://openreview.net/pdf?id=r1xMH1BtvB

ELECTRA = Efficiently Learning an Encoder that Classifies Token Re-placements Accurately

после обучения – генератор не нужен,
а дискриминатор доучивается на «downstream»-задачах

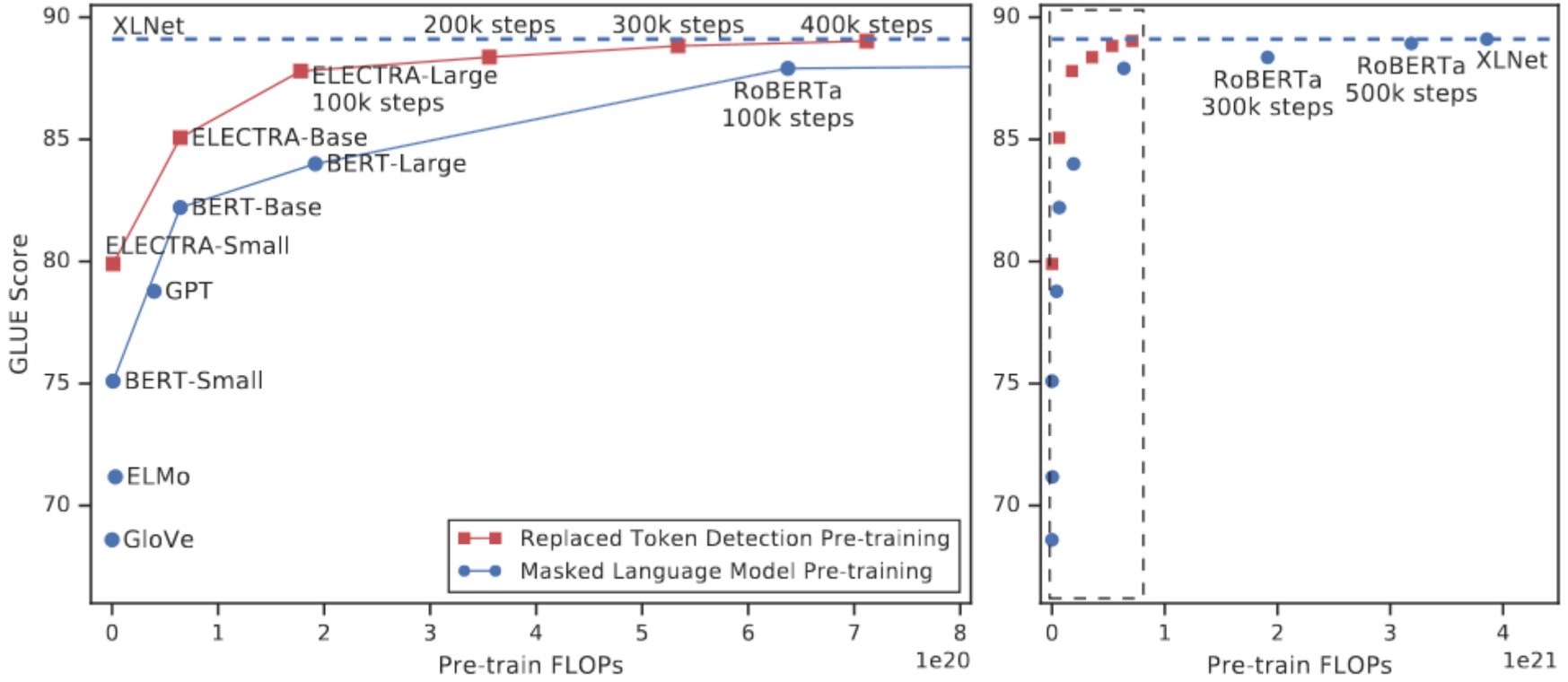


Figure 1: Replaced token detection pre-training consistently outperforms masked language model pre-training given the same compute budget. The left figure is a zoomed-in view of the dashed box.

ELECTRA

Model	Train FLOPs	Params	SQuAD 1.1		SQuAD 2.0	
			EM	F1	EM	F1
BERT-Base	6.4e19 (0.09x)	110M	80.8	88.5	–	–
BERT	1.9e20 (0.27x)	335M	84.1	90.9	79.0	81.8
SpanBERT	7.1e20 (1x)	335M	88.8	94.6	85.7	88.7
XLNet-Base	6.6e19 (0.09x)	117M	81.3	–	78.5	–
XLNet	3.9e21 (5.4x)	360M	**89.7**	**95.1**	87.9	**90.6**
RoBERTa-100K	6.4e20 (0.90x)	356M	–	94.0	–	87.7
RoBERTa-500K	3.2e21 (4.5x)	356M	88.9	94.6	86.5	89.4
ALBERT	3.1e22 (44x)	235M	89.3	94.8	87.4	90.2
BERT (ours)	7.1e20 (1x)	335M	88.0	93.7	84.7	87.5
ELECTRA-Base	6.4e19 (0.09x)	110M	84.5	90.8	80.5	83.3
ELECTRA-400K	7.1e20 (1x)	335M	88.7	94.2	86.9	89.6
ELECTRA-1.75M	3.1e21 (4.4x)	335M	**89.7**	94.9	**88.1**	**90.6**

Семейство BERT

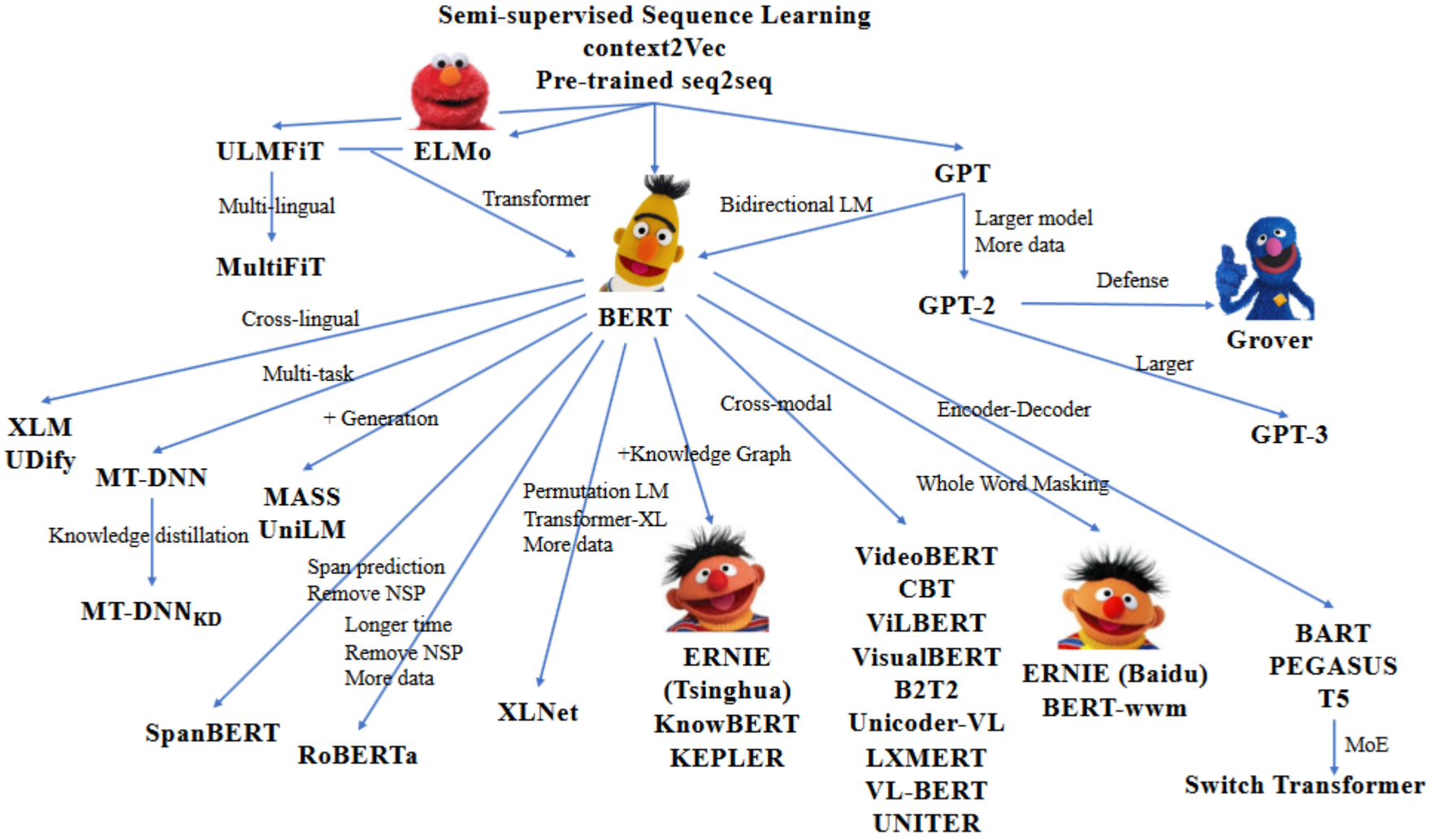


Figure 9: The family of recent typical PTMs, including both pre-trained language models and multimodal models.

https://arxiv.org/pdf/2106.07139.pdf

Итог

Трансформер – оригинальная нерекуррентная архитектура
Сейчас почти все лучшие решения основаны на трансформере

Тренды:
обучение на большом корпусе
подстраивание под конкретные задачи
«упрощение»: уменьшение числа параметров, слоёв

Центральная идея современного NLP –
«Transfer Learning»
предобучаемся на задаче с большими данными и дешёвой разметкой
переносим модель и дообучаем под нашу задачу

Пример аннотированного кода
http://nlp.seas.harvard.edu/2018/04/03/attention.html