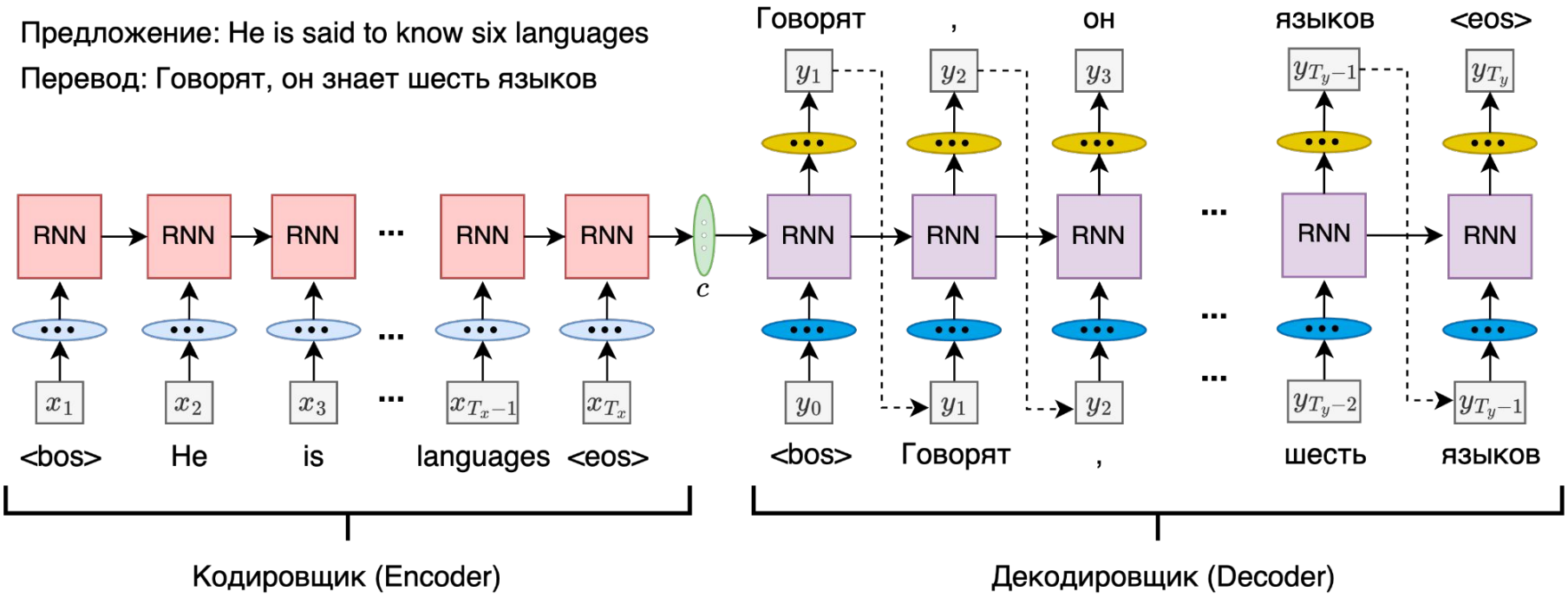
Архитектура трансформера, GPT, BERT

Артём Степанов

education

Энкодер-декодер

Encoder-decoder на основе RNN



Модель состоит из двух частей: кодировщик (encoder) и декодировщик (decoder)

Трансформер и его предпосылки

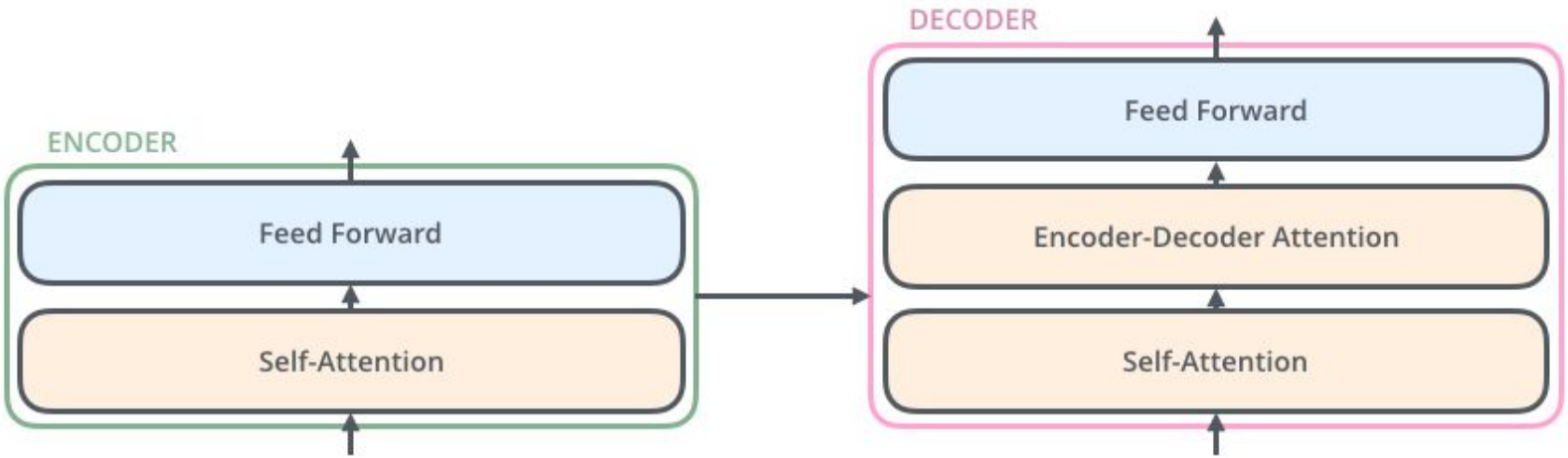
Модель в идеальном мире

- effective
- efficient

Проблемы RNN

- почему не effective- потеря информации
- почему не efficient- не параллелится

Transformer architecture



Картинка взята [отсюда]

Self-attention

Аналогия со словарём в python:

$$d = \{k_1 : v_1, k_2 : v_2, \dots\}$$

$$\text{importance}(q_i, k_j) \in [0, 1]$$

$$h_i = 0$$

$$\text{for } k_j, v_j \text{ in } d.items() :$$

$$h_i += \text{importance}(q_i, k_j) * v_j$$

Self-attention

$\widetilde{E} \in \mathbb{R}^{n \times d}$ — матрица эмбеддингов, n — длина последовательности, d — внутренняя размерность

Даны матрицы $Q, K, V \in \mathbb{R}^{d \times d}$

$$\widetilde{Q} = \widetilde{E} \times Q, \widetilde{K} = \widetilde{E} \times K, \widetilde{V} = \widetilde{E} \times V$$

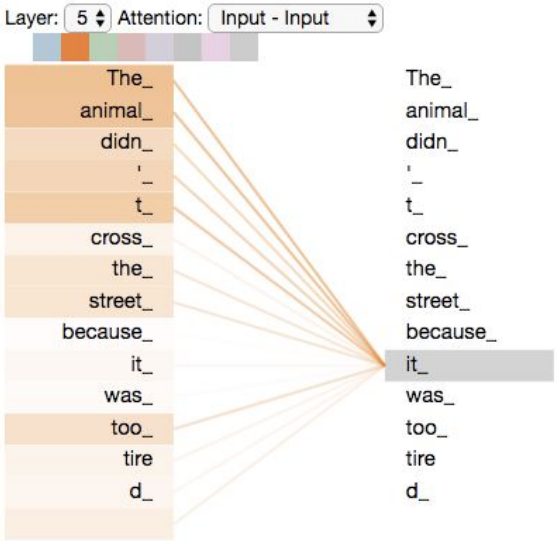
Attention для i-ого токена будет считаться как:

$$h_i = \sum_{j=1}^{n} \alpha_j * \widetilde{V_j}$$

$$\alpha = softmax(\widetilde{K} \times \widetilde{Q_i^T}) \in \mathbb{R}^n$$

alpha- attention scores

Self-attention



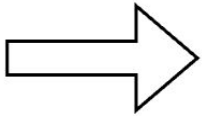
Картинка взята [отсюда]

Self-attention

Attention сразу для всей последовательности:

$$\alpha = softmax(\widetilde{K} \times \widetilde{Q_i^T}) \in \mathbb{R}^n$$

$$h_i = \sum_{j=1}^{n} \alpha_j * \widetilde{V_j}$$



$$A = softmax(\widetilde{K} \times \widetilde{Q^T}) \in \mathbb{R}^{n \times n}$$

$$H = A * \widetilde{V}$$

Self-attention

Пусть содержимое векторов q,k- это н.с.в. с распределением $\mathcal{N}(0,1)$. Тогда:

$$\mathbb{E}(< q,k >) = \mathbb{E}(\sum_{j=1}^{d} q_j * k_j) = \sum_{j=1}^{d} \mathbb{E}q_j * \mathbb{E}k_j = 0$$

$$Var(< q,k >) = Var(\sum_{j=1}^{d} q_j * k_j) = \sum_{j=1}^{d} Var(q_j) * Var(k_j) = d$$

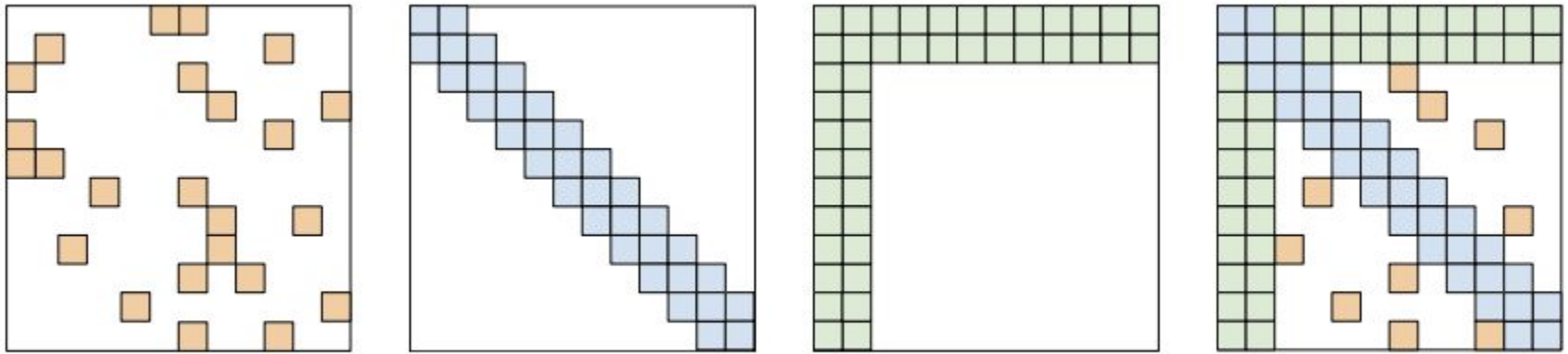
$$softmax(K \times Q^T) \rightarrow softmax(\frac{K \times Q^T}{\sqrt{d}})$$

Self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$

Картинка взята отсюда

Self-attention



(a) Random attention (b) Window attention (c) Global Attention (d) BIGBIRD

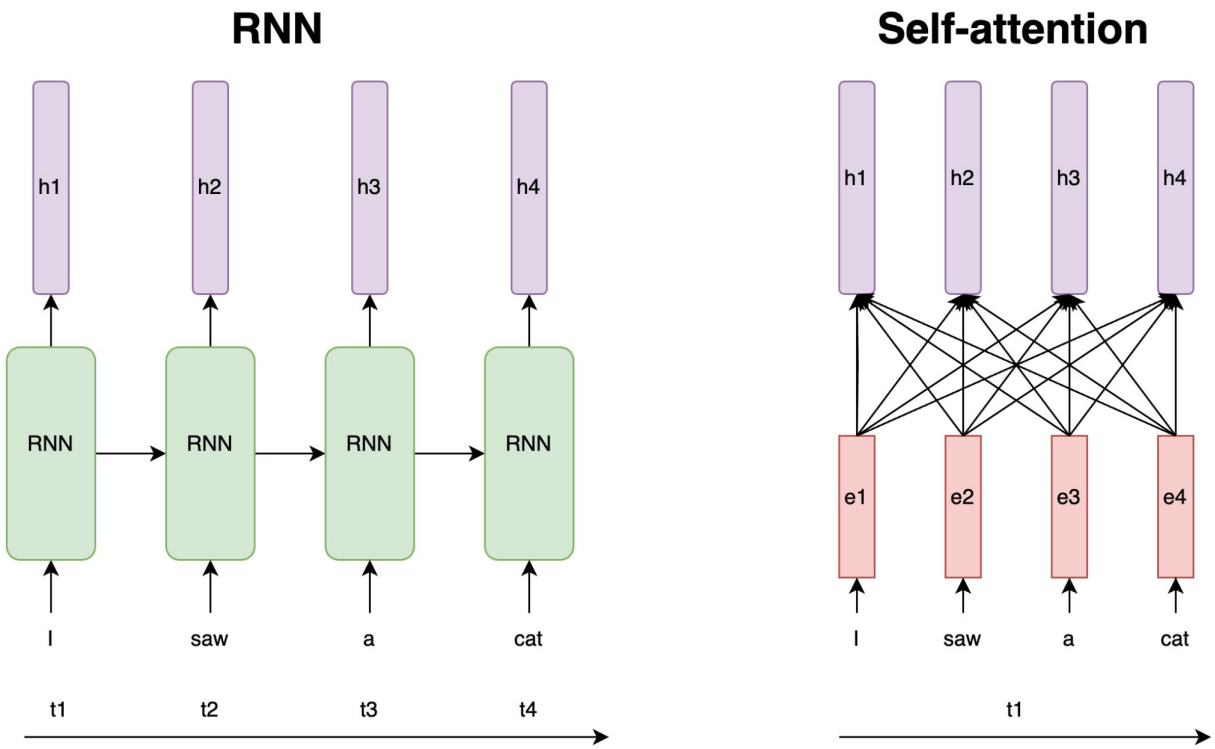
Картинка взята [отсюда]

Self-attention

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$

Картинка взята отсюда

Self-attention



Multi-head self-attention

Было: $Q, K, V \in \mathbb{R}^{\mathtt{d} \times \mathtt{d}}$

Стало: k матриц $(Q_1, K_1, V_1), \ldots, (Q_k, K_k, V_k)$. Каждая матрица $\in \mathbb{R}^{\mathtt{d} \times \frac{\mathtt{d}}{\mathtt{k}}}$

$$MultiHeadAttention = [SelfAttention_1(x), SelfAttention_2(x), \ldots, SelfAttention_k(x)] * O, O \in \mathbb{R}^{\mathtt{d} \times \mathtt{d}}$$

Positional encoding

Не хватает позиционной информации. Решение:

$$\tilde{E} = E + PE$$

Для позиции pos имеем:

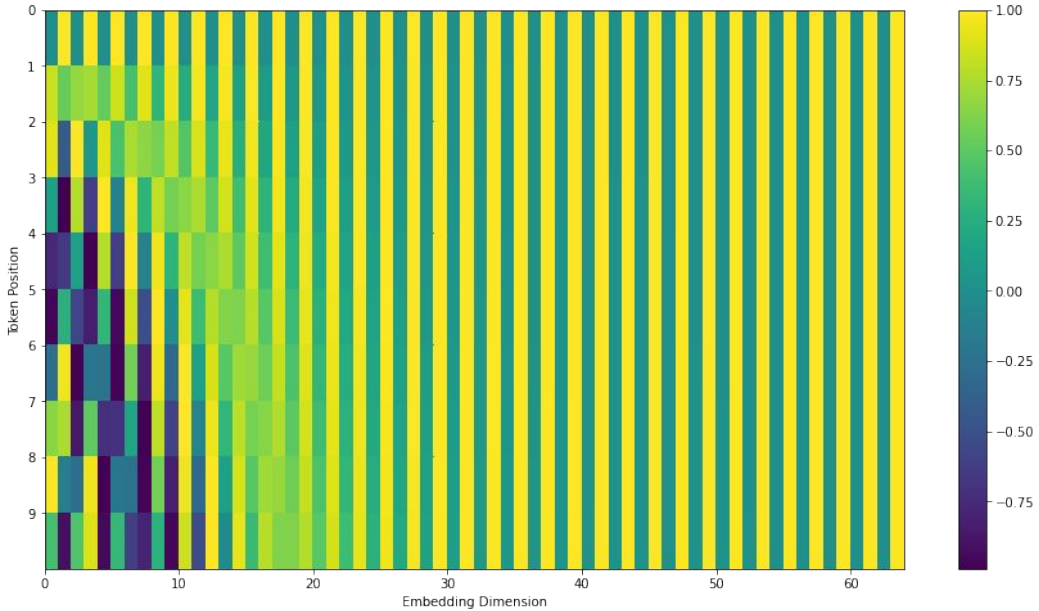
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

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

Нижняя картинка взята отсюда

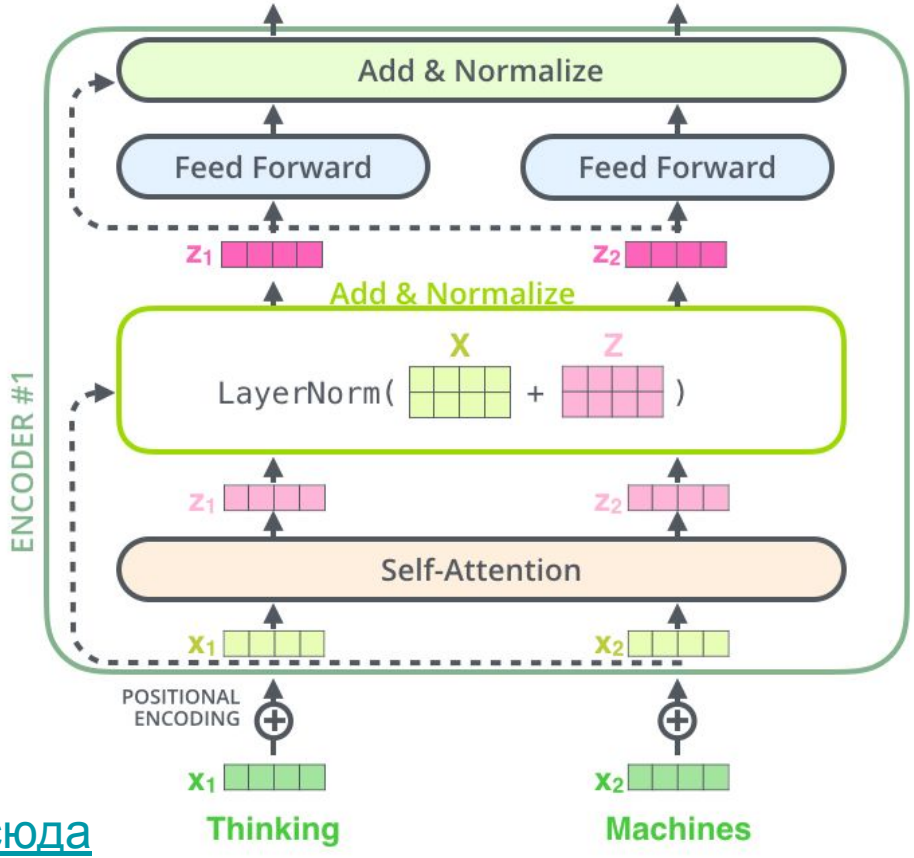
Positional encoding

Пример матрицы с позиционной информацией:



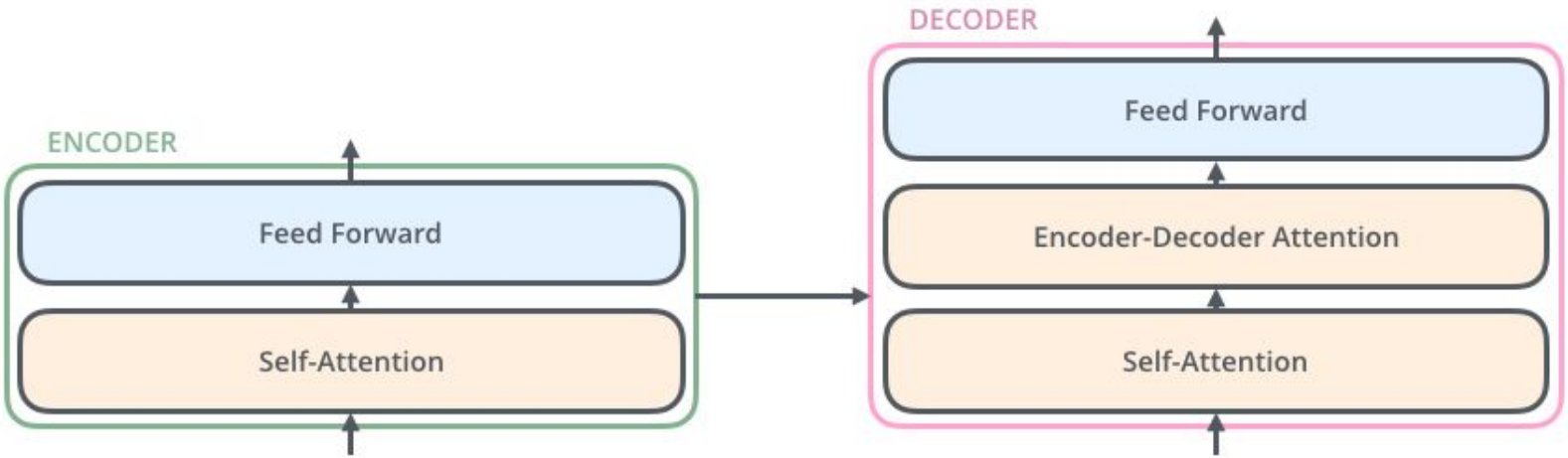
Картинка взята [отсюда]

Encoder block



Картинка взята [отсюда]

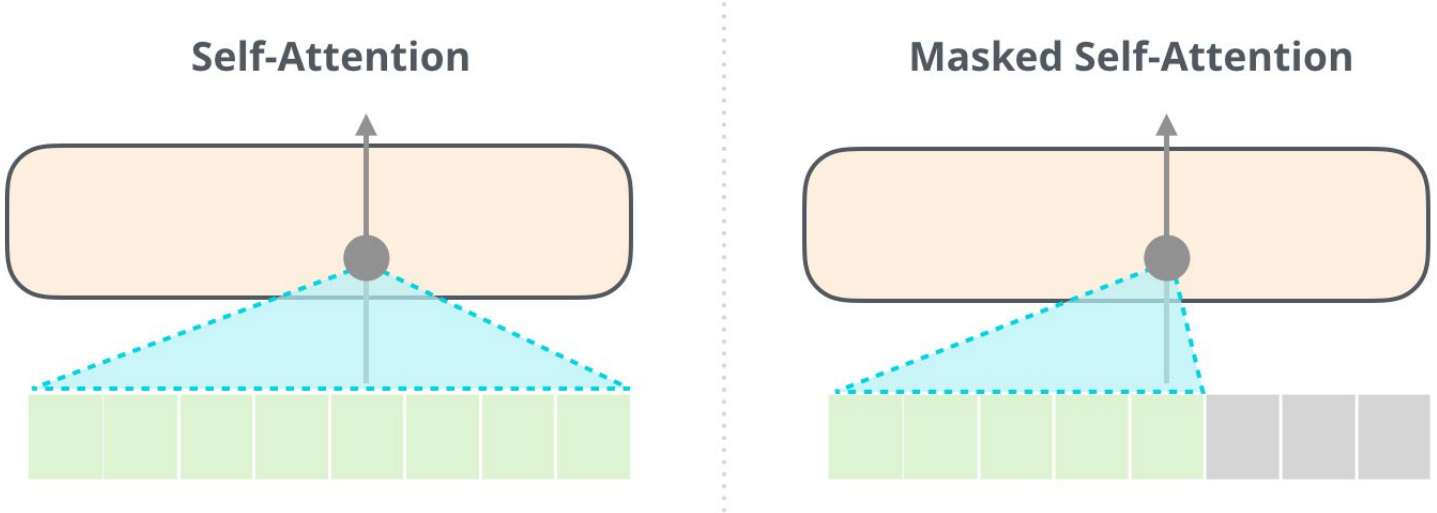
Decoder block



Картинка взята [отсюда]

Decoder block

Отличие работы self-attention в энкодере и в декодере:



Картинка взята [отсюда]

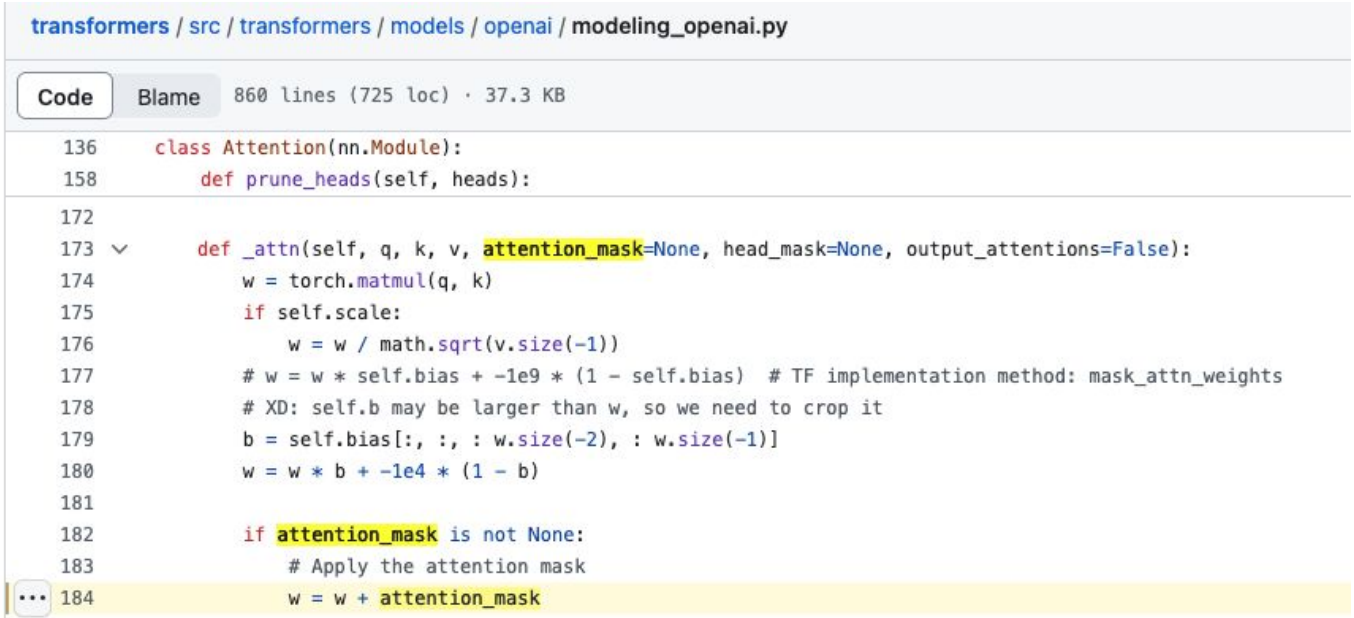
Attention mask

Пример формирования attention mask для masked self-attention:

	I	am	a	student
I	0	$-\infty$	$-\infty$	$-\infty$
am	0	0	$-\infty$	$-\infty$
a	0	0	0	$-\infty$
student	0	0	0	0

Attention mask

Что происходит с маской дальше:



transformers / src / transformers / models / openai / modeling_openai.py

Code | Blame | 860 lines (725 loc) · 37.3 KB

```
136  class Attention(nn.Module):
158      def prune_heads(self, heads):
172
173      def _attn(self, q, k, v, attention_mask=None, head_mask=None, output_attentions=False):
174          w = torch.matmul(q, k)
175          if self.scale:
176              w = w / math.sqrt(v.size(-1))
177          # w = w * self.bias + -1e9 * (1 - self.bias)  # TF implementation method: mask_attn_weights
178          # XD: self.b may be larger than w, so we need to crop it
179          b = self.bias[:, :, : w.size(-2), : w.size(-1)]
180          w = w * b + -1e4 * (1 - b)
181
182          if attention_mask is not None:
183              # Apply the attention mask
184              w = w + attention_mask
```

[Ссылка](#) на исходный код

Pretraining и fine-tuning

Создаём модель под задачу

Обычно:

- собираем большую выборку под вашу задачу
- учим модель на этой выборке

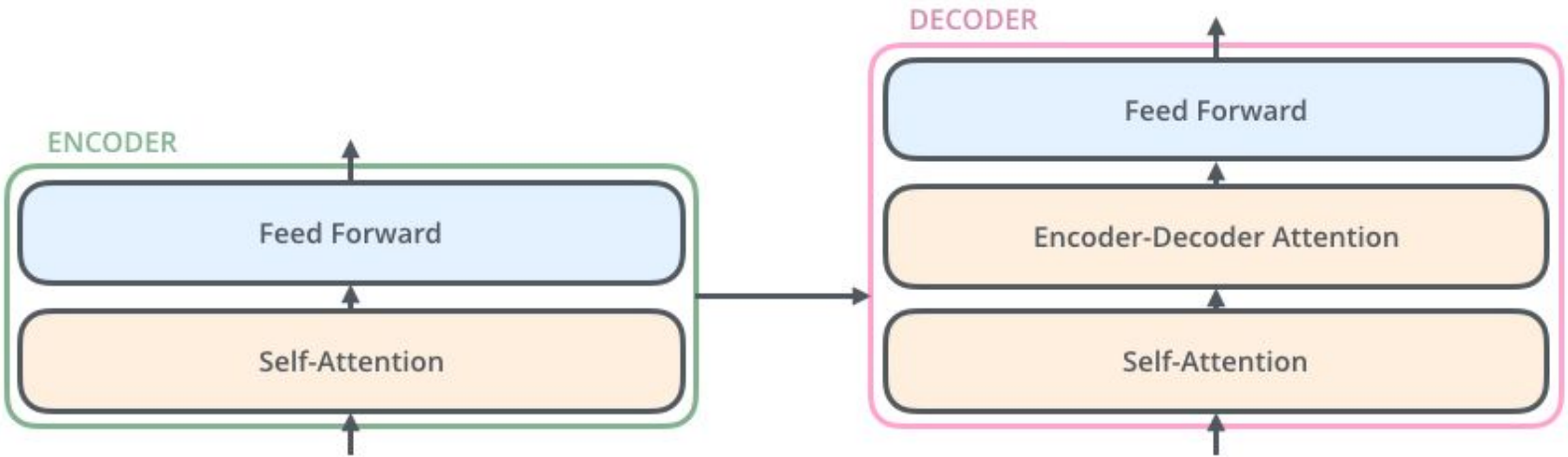
Создаём модель под задачу

Transfer learning: pretraining(self-supervised) → fine-tuning

GPT

GPT

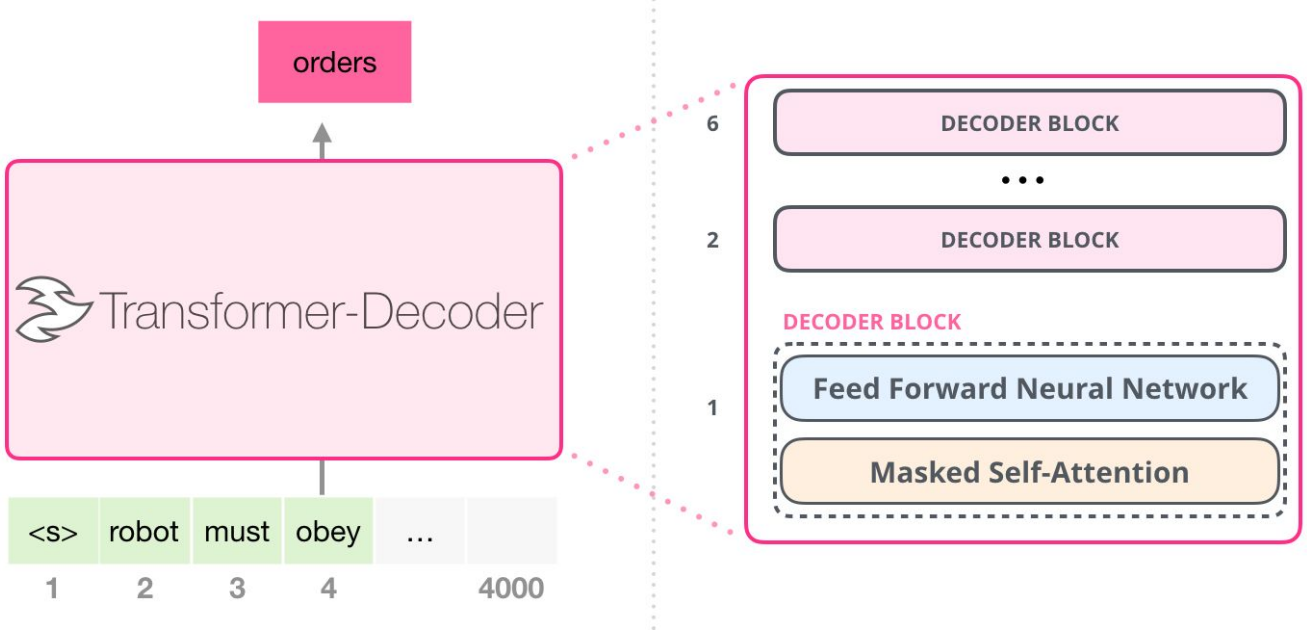
Архитектура: декодер-блок трансформера



Картинка взята [отсюда]

GPT

Архитектура: декодер-блок трансформера



Картинка взята [отсюда]

GPT

Идея:

1. предобучаем декодер трансформера как языковую модель(generative pre-training)

$$L_1(\mathcal{U}) = \sum_i \log P(u_i|u_{i-k}, \ldots, u_{i-1}; \Theta)$$

2. файнтюним языковую модель под свою задачу(supervised fine-tuning)

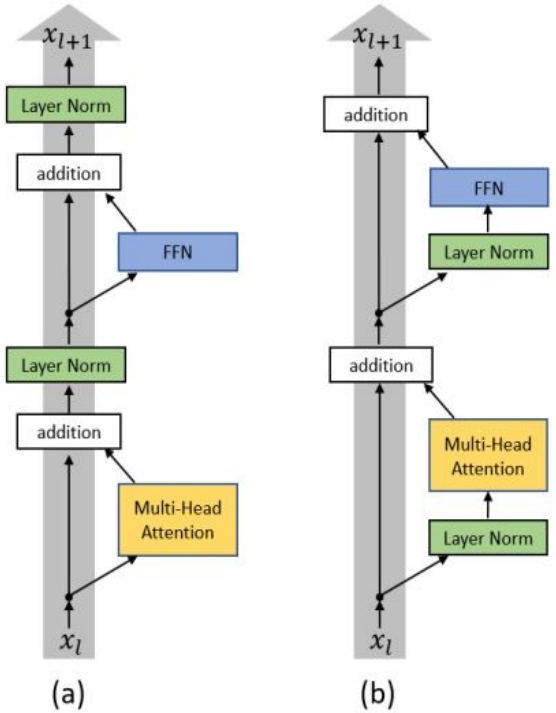
$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

GPT

Архитектурный хак:

вставка LayerNorm внутрь residual connections



Картинка взята [отсюда]

GPT

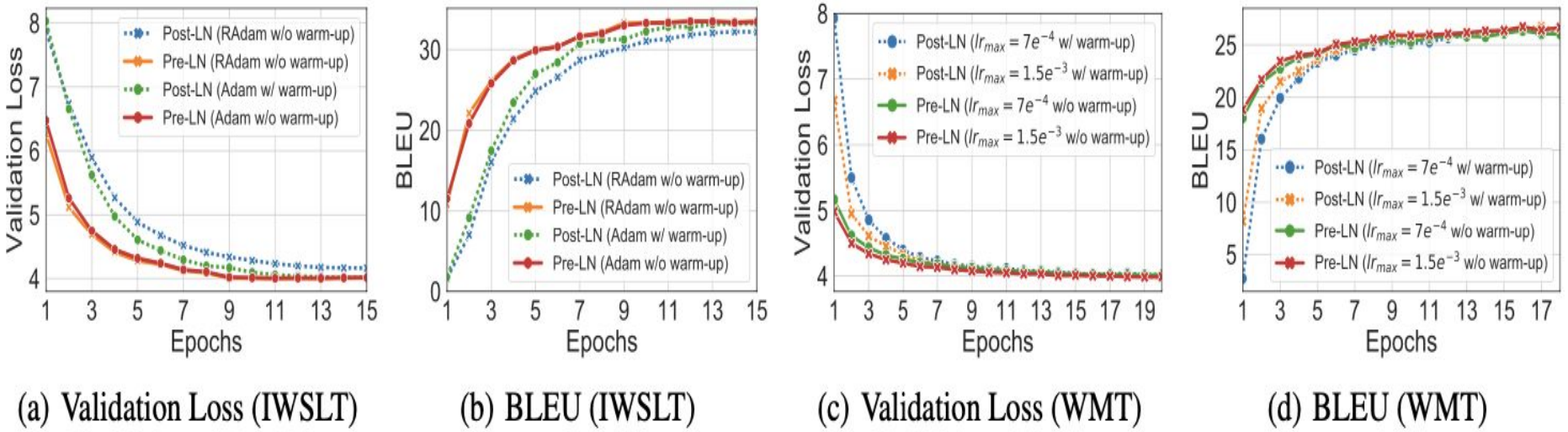
Архитектурный хак:

вставка LayerNorm внутрь residual connections. Эффекты:

- меньше гиперпараметров
- ускорение сходимости

GPT

Для PostLN нужен lr-warmup



(a) Validation Loss (IWSLT) (b) BLEU (IWSLT) (c) Validation Loss (WMT) (d) BLEU (WMT)

Картинка взята [отсюда]

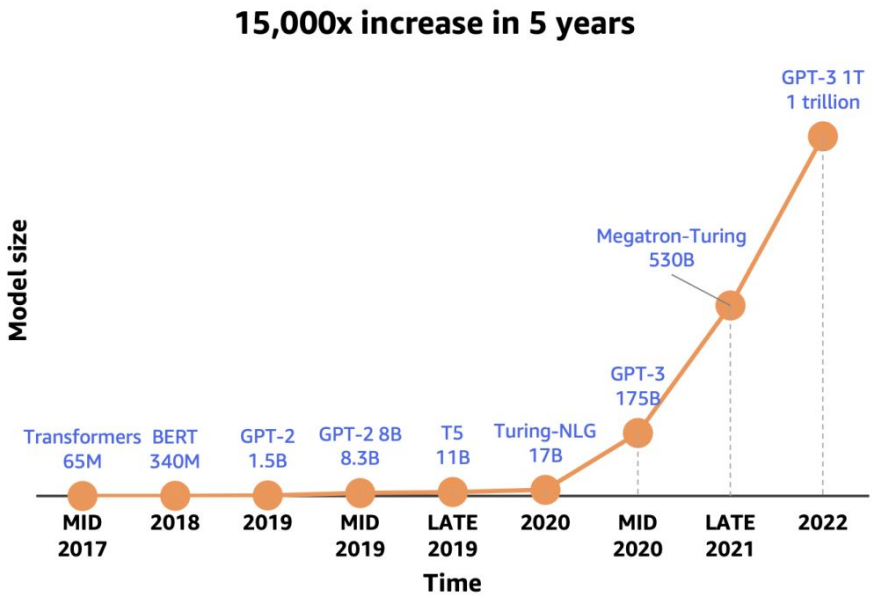
GPT

Что получаем?

- получаем модель, которая справляется лучше доменно-специфичной модели, обученной на тонне размеченных данных
- одна модель под все задачи(при условии, что мы добавим один линейный слой)

GPT

Эволюция кол-ва параметров в языковых моделях



Картинка взята [отсюда]

GPT

Без всякого файнтюнинга!

Dataset	Metric	Our result	Previous record	Human
Winograd Schema Challenge	accuracy (+)	70.70%	63.7%	92%+
LAMBADA	accuracy (+)	63.24%	59.23%	95%+
LAMBADA	perplexity (–)	8.6	99	~1–2
Children's Book Test Common Nouns (validation accuracy)	accuracy (+)	93.30%	85.7%	96%
Children's Book Test Named Entities (validation accuracy)	accuracy (+)	89.05%	82.3%	92%
Penn Tree Bank	perplexity (–)	35.76	46.54	unknown
WikiText-2	perplexity (–)	18.34	39.14	unknown
enwik8	bits per character (–)	0.93	0.99	unknown
text8	bits per character (–)	0.98	1.08	unknown
WikiText-103	perplexity (–)	17.48	18.3	unknown

GPT-2 achieves state-of-the-art on Winograd Schema, LAMBADA, and other language modeling tasks.

Картинка взята отсюда

GPT

Откуда берётся магия?

- Дискриминативная парадигма: мы моделируем p(y|x)
- Мультизадачная парадигма: мы моделируем p(y|x, task)

Как внедрить знание о задаче в модель?

GPT

Как внедрить знание о задаче в модель?

(translate from english to french <english text>)

(answer the question: <question>)

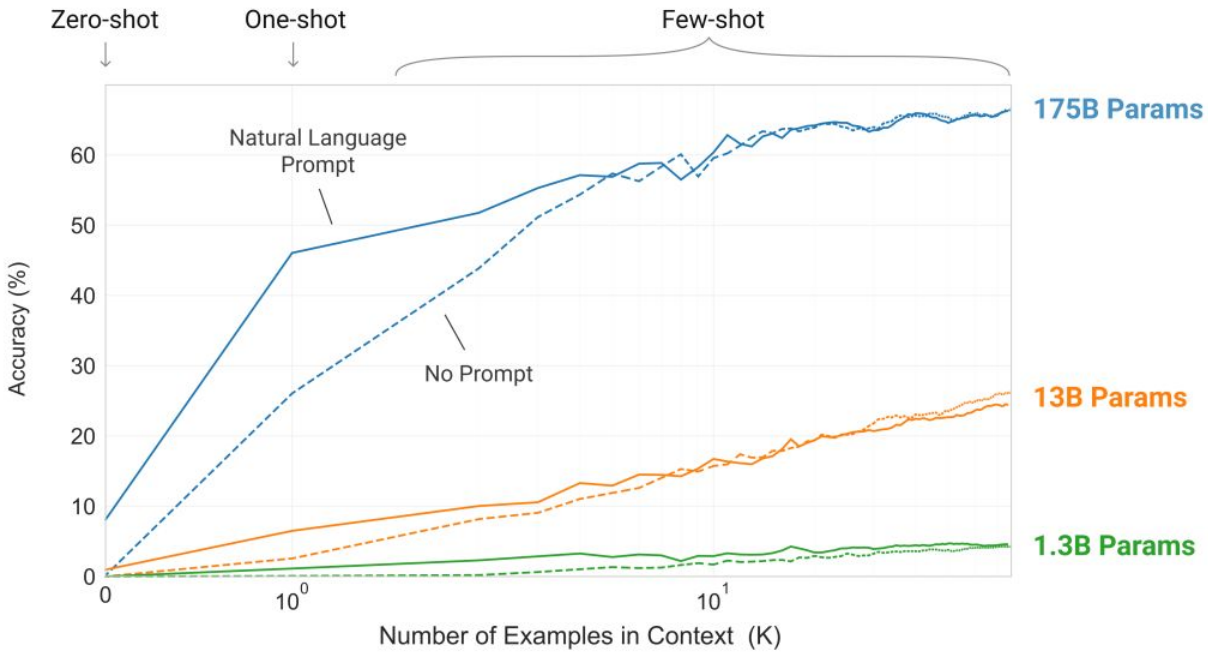
GPT

Методология работы с промтами:

- Few-shot learning- инструкция + 10-100 примеров
- One-shot learning- инструкция + 1 пример
- Zero-shot learning- инструкция

GPT

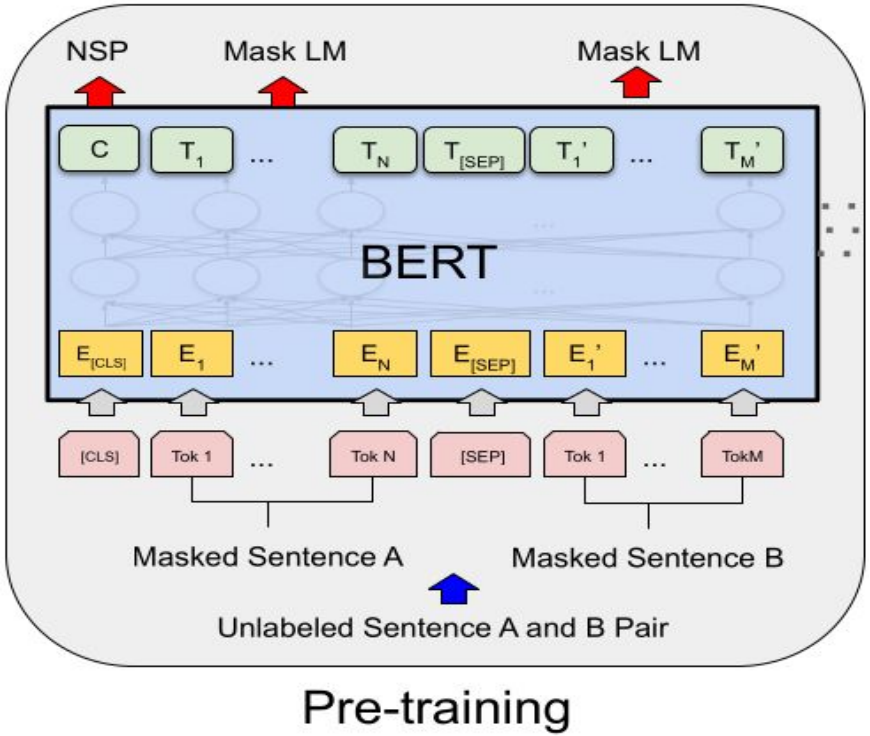
Это круто работает!



Картинка взята [отсюда]

BERT

BERT



Картинка взята [отсюда]

BERT

Pretraining-задачи:

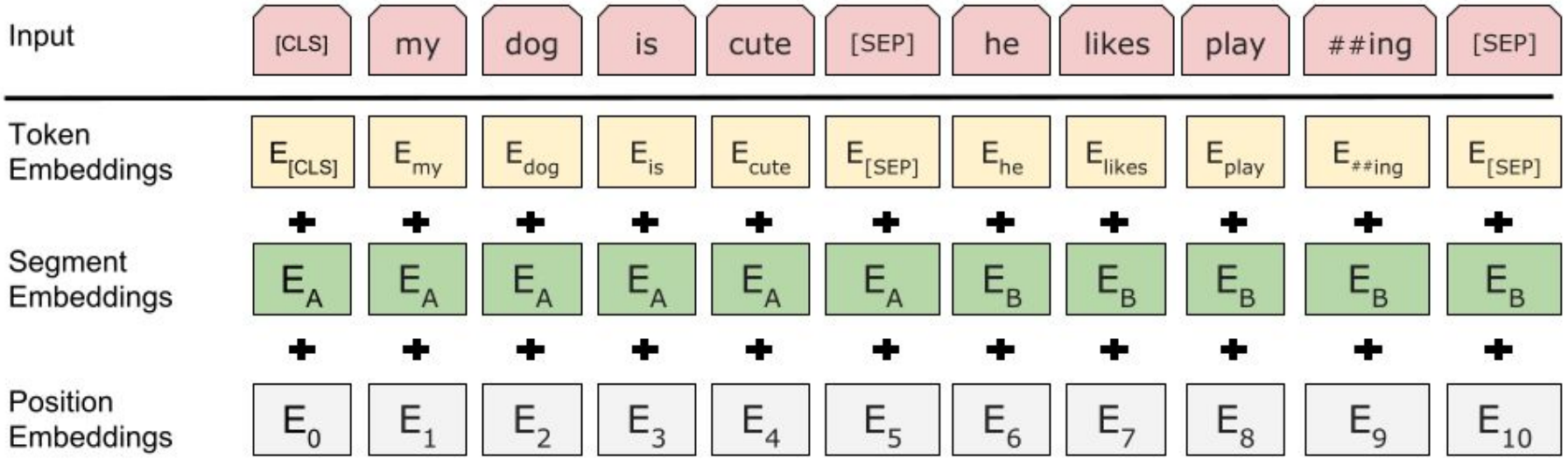
- Next Sentence Prediction(NSP)
- Masked Language Modelling(MLM)

Как маскируем токены в MLM

Из 15% токенов, выбранных для маскирования:

- В 80% случаев заменяем на токен [MASK]
- В 10% случаев заменяем на случайный токен
- В 10% случаев ничего не делаем

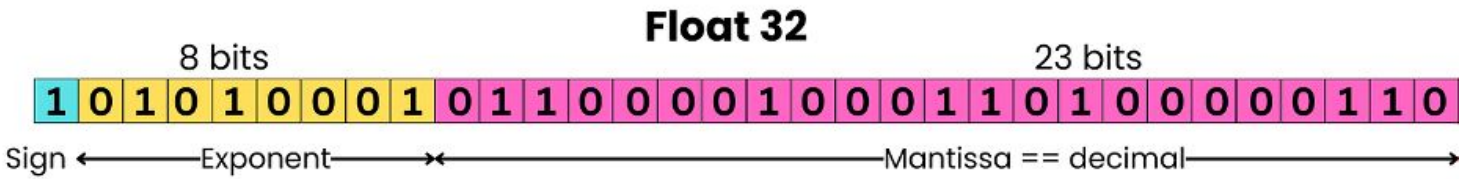
BERT

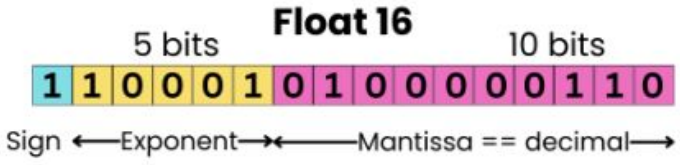


Картинка взята [отсюда]

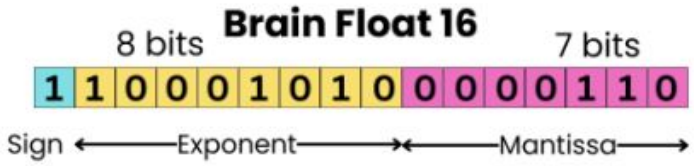
Making transformers go brrr with mixed-precision

Mixed-precision





$$(-1)^{sign} 2^{(exponent-15)} \times 1.mantissa$$

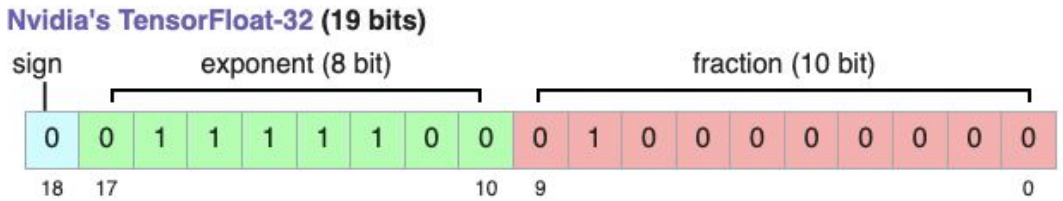


$$(-1)^{sign} 2^{(exponent-127)} \times 1.mantissa$$

Картинки взяты [отсюда]

Mixed-precision

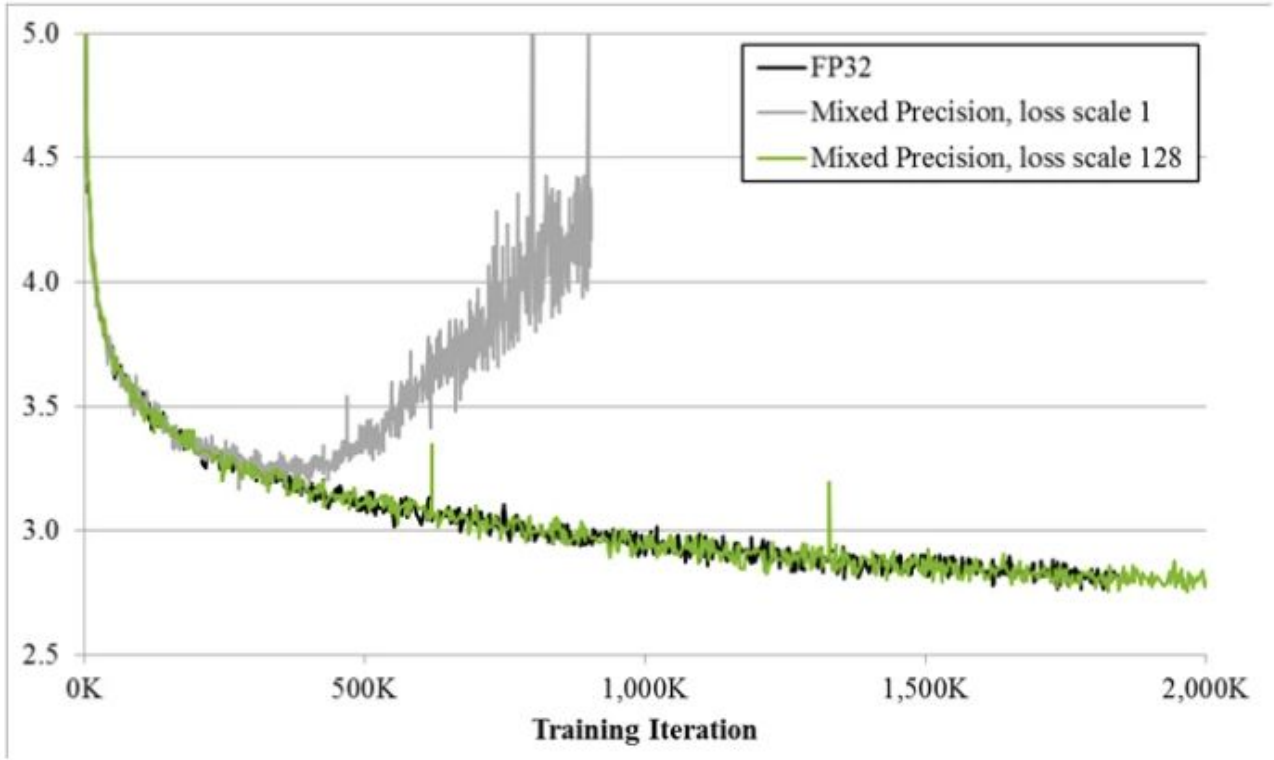
Пример с tensor cores и tf32:



A x B + C

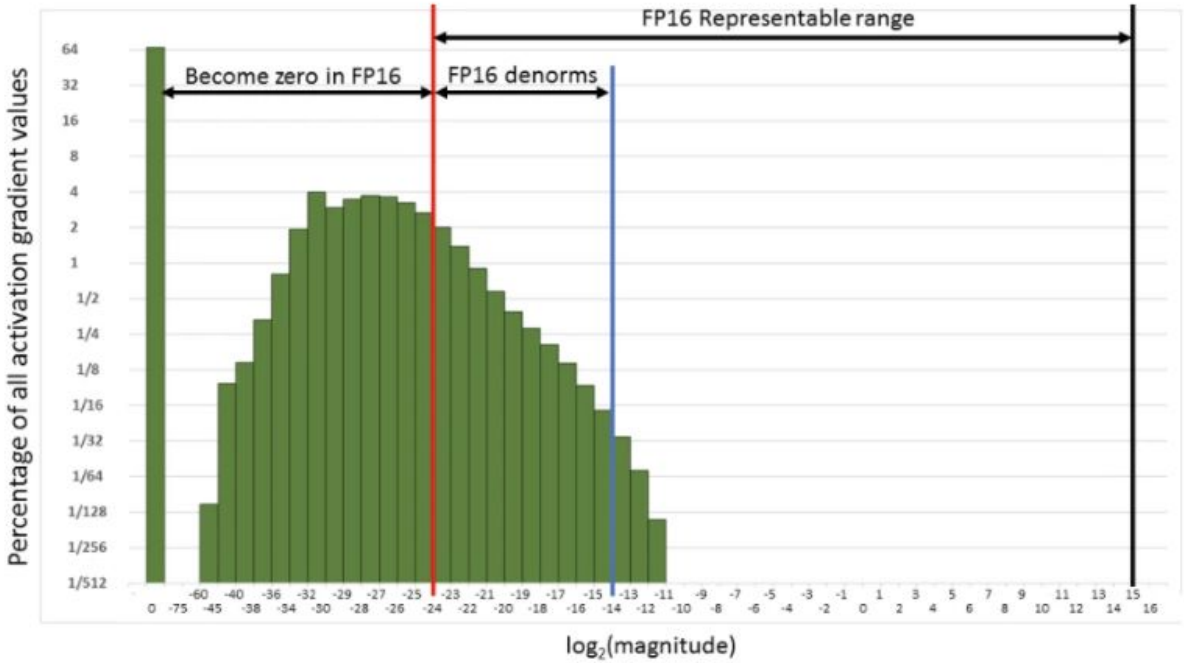
Картинка взята [отсюда]

Mixed-precision



Картинка взята [отсюда]

Mixed-precision



Картинка взята [отсюда]

Mixed-precision

Алгоритм обучения в формате mixed-precision

на каждой итерации:

a. делаем forward pass
b. умножаем лосс на S, считаем градиенты
c. получившиеся градиенты умножаем на 1/S
d. обновляем веса

Mixed-precision

Динамический подбор множителя

- Гиперпараметры:
  - S- стартовый множитель
  - K- число итераций
- На текущей итерации:
  - Если последние K итераций при подсчёте градиентов не было underflow/overflow, увеличиваем S
  - Иначе не обновляем веса и уменьшаем S

Mixed-precision

Не ко всем операциям применим переход в формат с более низкой точностью. Примеры:

- softmax
- статистики для batchnorm

Спасибо за внимание!