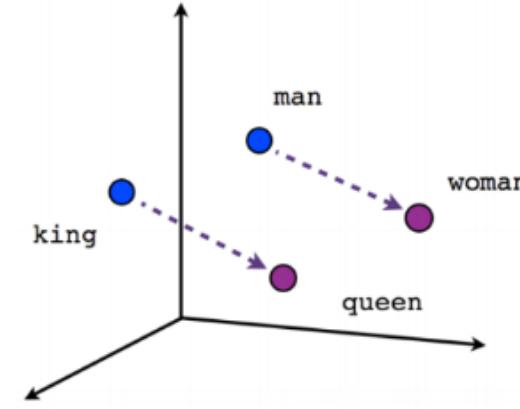
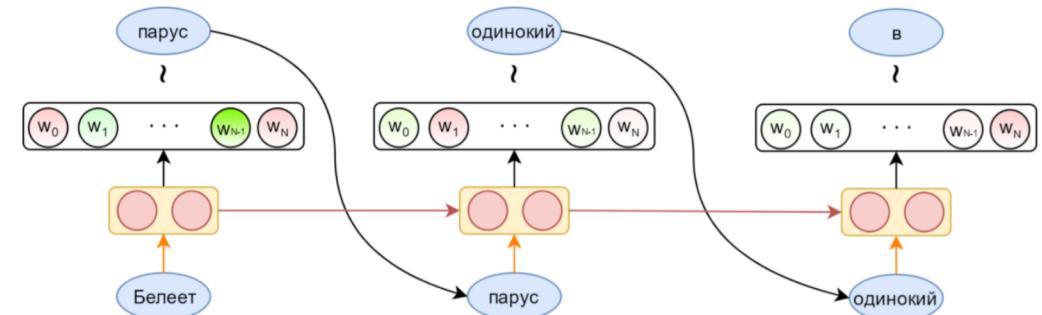


О чём была первая часть?

- Основные задачи, проблемы, успехи NLP
- Word2Vec
- GloVe
- RNN, backpropagation through time
- Языковое моделирование
- Attention
- Машинный перевод



Male-Female

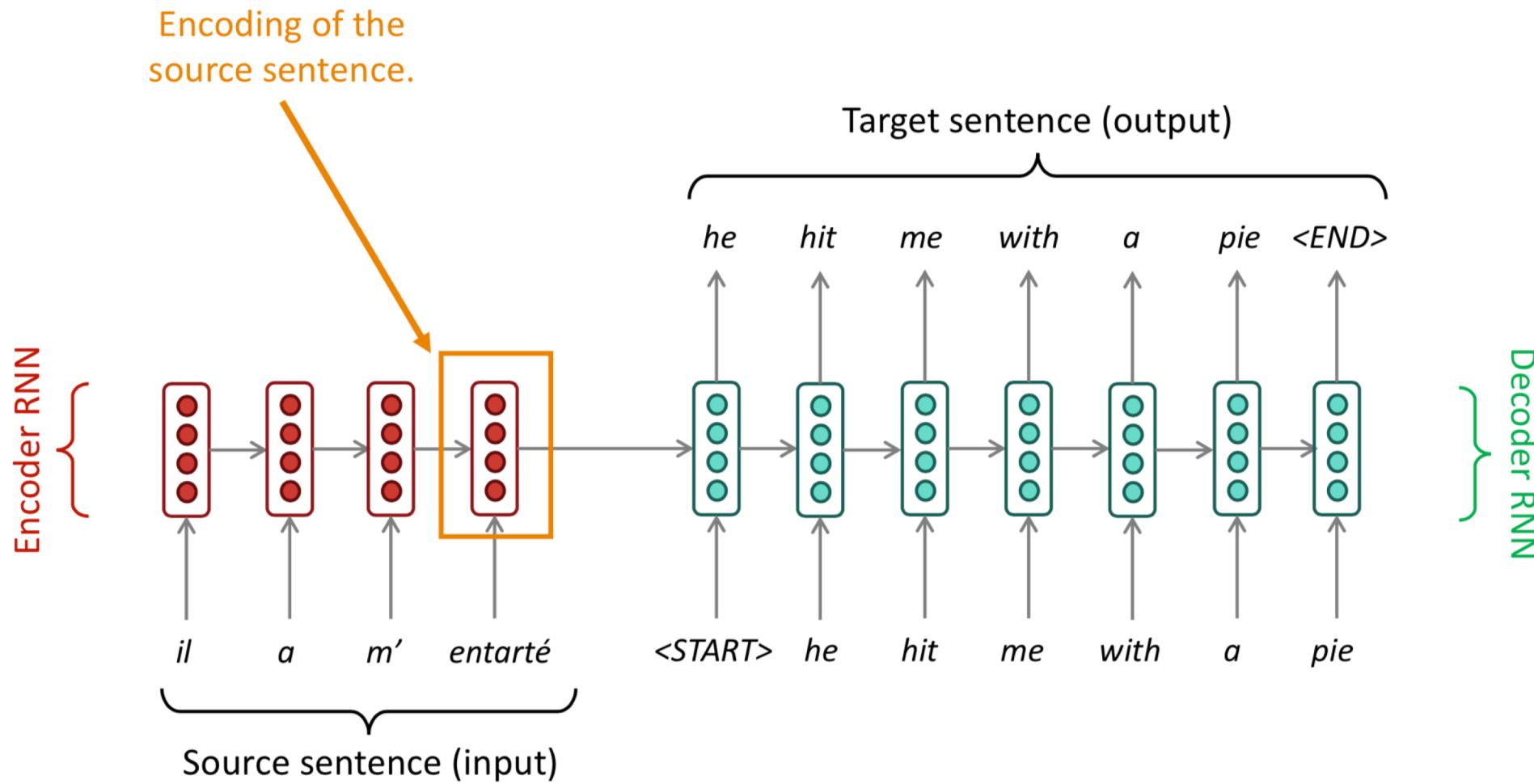


План

- Проблема Seq2Seq
- Трансформер
 - Positional encoding
 - Layer Norm
- Self-attention
- BERT

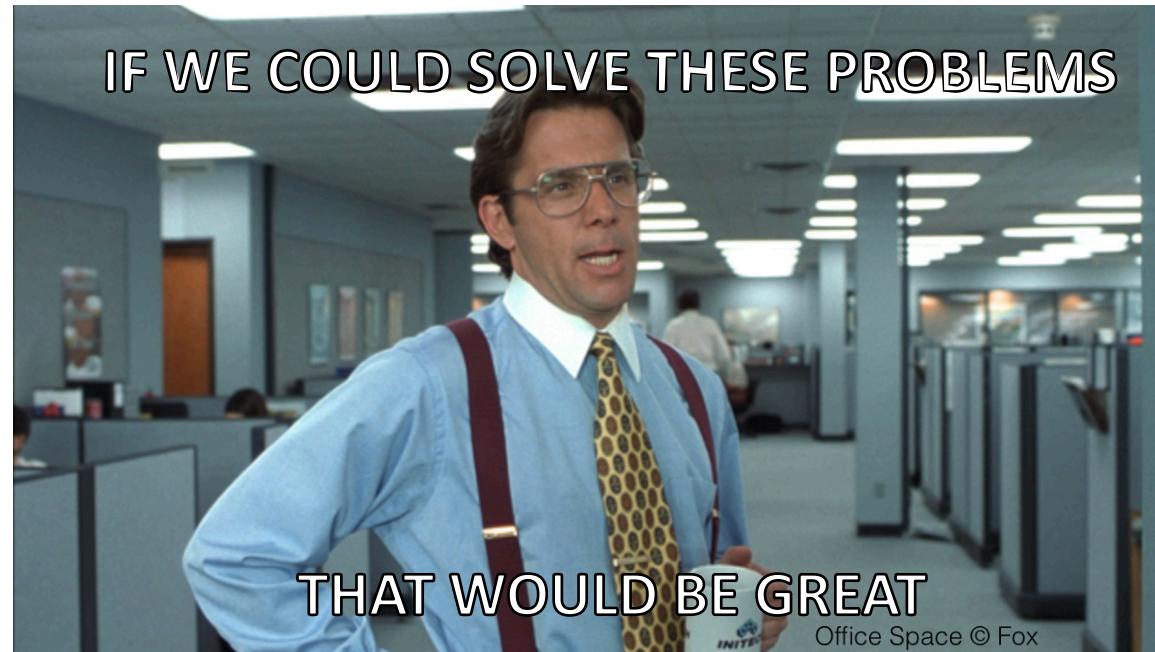


Проблема Seq2Seq модели



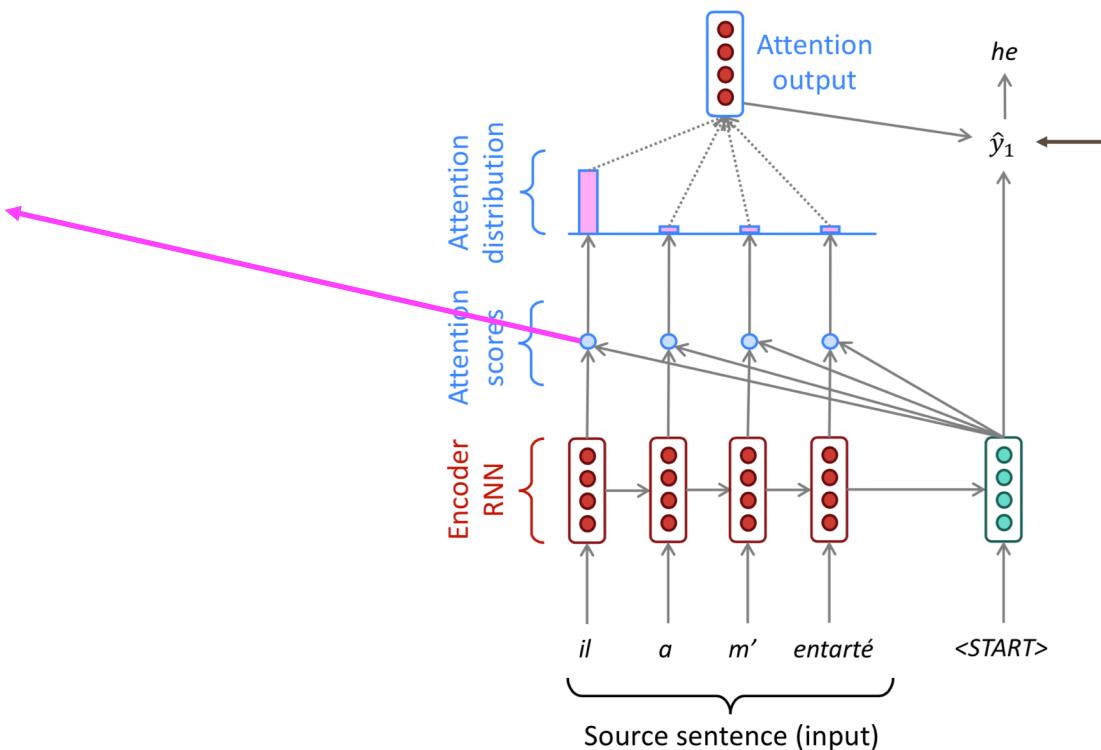
Проблемы Seq2Seq

- Входная последовательность символов кодируется в один вектор
- Долго обучаются – обучение нельзя распараллелить



Attention в Seq2seq

- Attention позволяет решить проблему раскодирования целого предложения из одного вектора
- Помогает определить вклад каждого слова в текущее
- На каждом этапе модель “смотрит” на разные входные элементы, что позволяет учитывать информацию с этапа энкодера



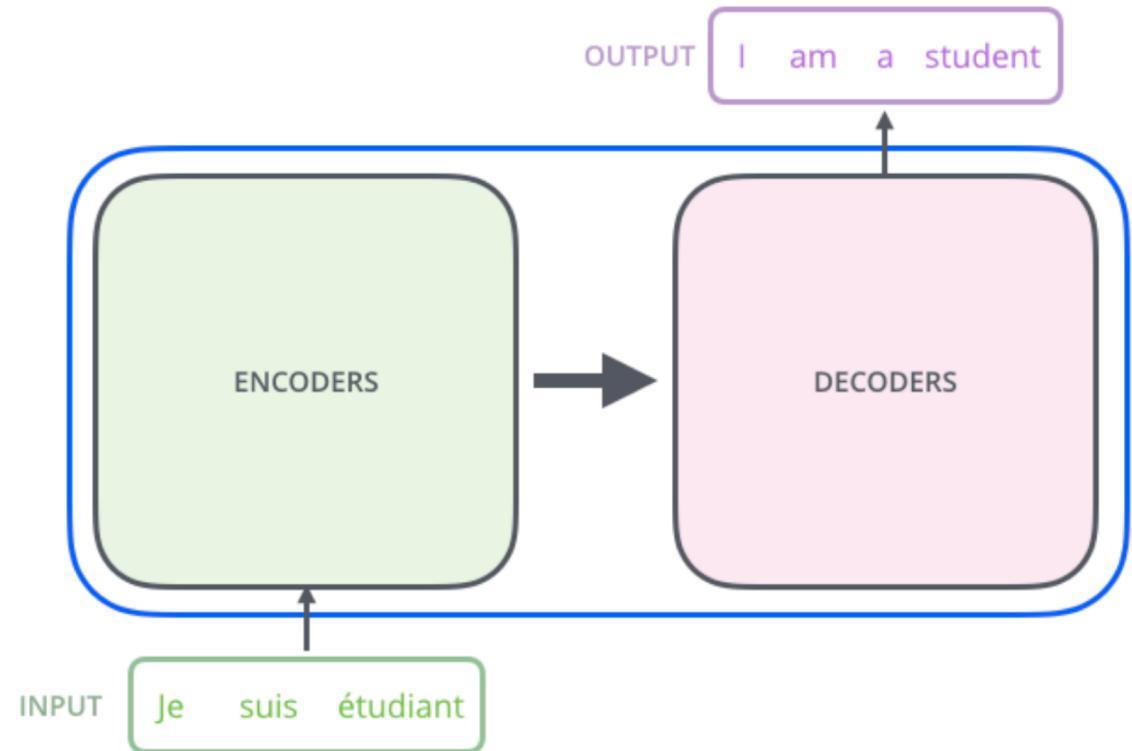
Конкатенируем attention
recalculated input либо
вместе с hidden state RNN
или с выходом RNN

She is eating a green apple.

A graph illustrating attention weights. The x-axis represents the words "eating", "a", "green", and "apple". The y-axis represents the attention score, ranging from "low attention" at the bottom to "high attention" at the top. The graph shows a sharp peak of high attention for the word "eating", followed by lower attention for "a", "green", and "apple".

Transformer overview

- Трансформер – новая архитектура нейронных сетей, пришедшая на смену RNN.
- Выполняет функцию и энкодера и декодера в задачах Seq2Seq
- SoTa для задач NLP

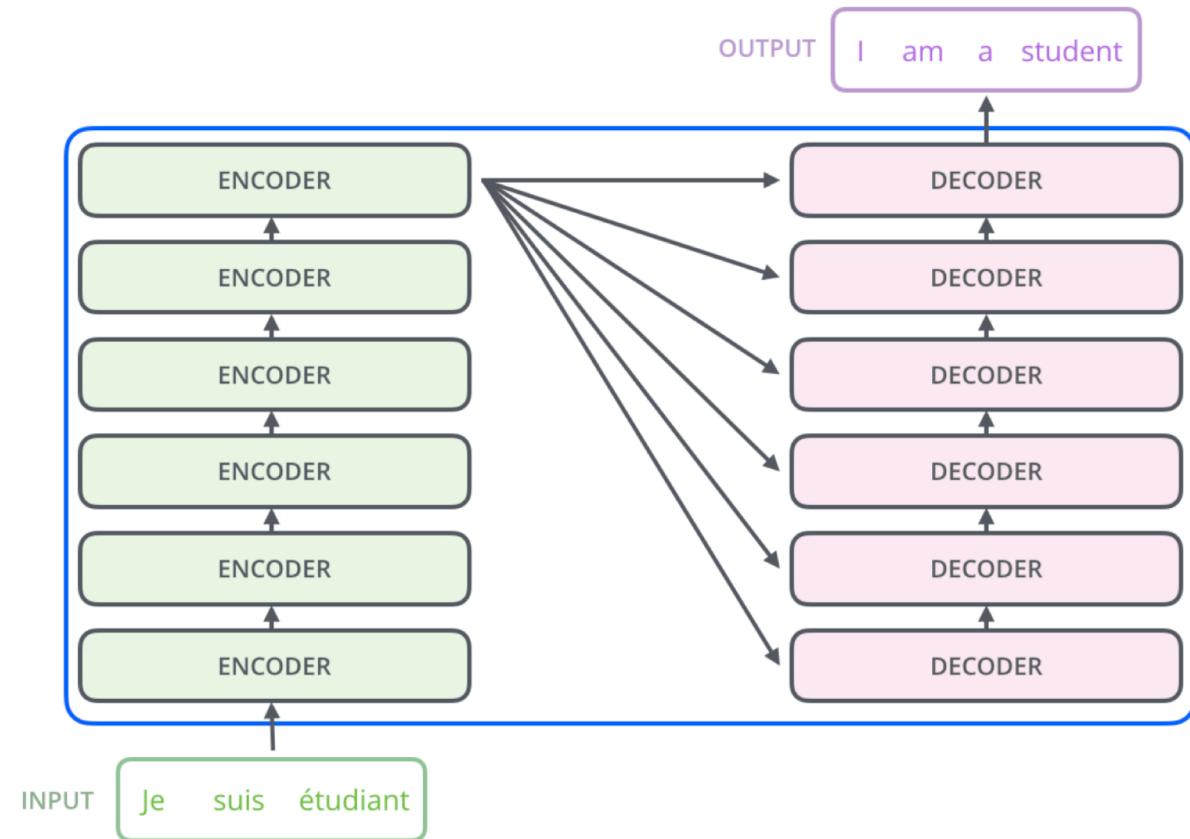


<http://jalammar.github.io/illustrated-transformer/>

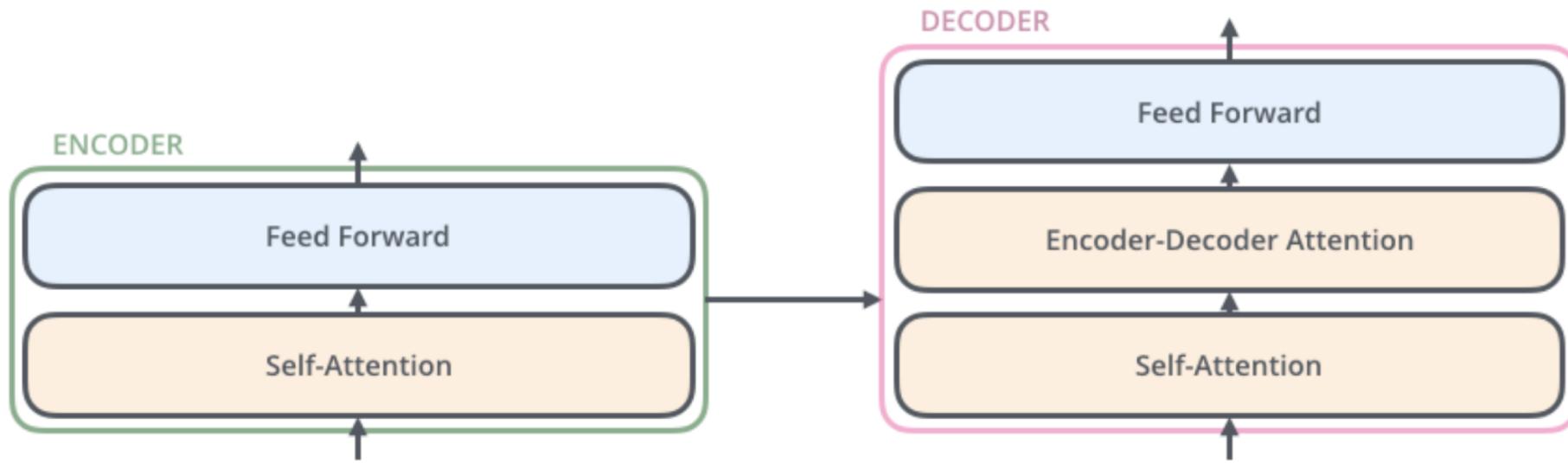
Transformer overview

Обычно используется стек из N трансформеров для решения задач.

- Базовый векторизатор
- Transfer Learning
- Генерация текста
- Машинный перевод



Transformer block

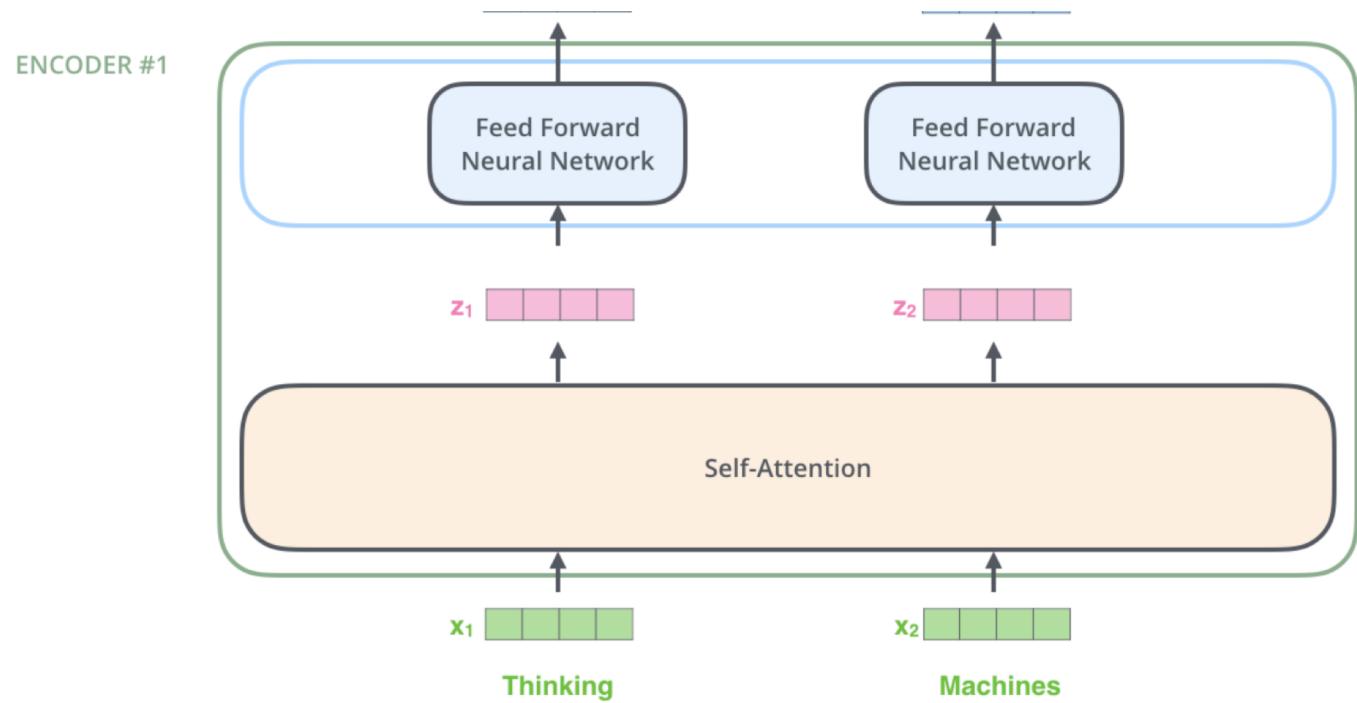


<http://jalammar.github.io/illustrated-transformer/>

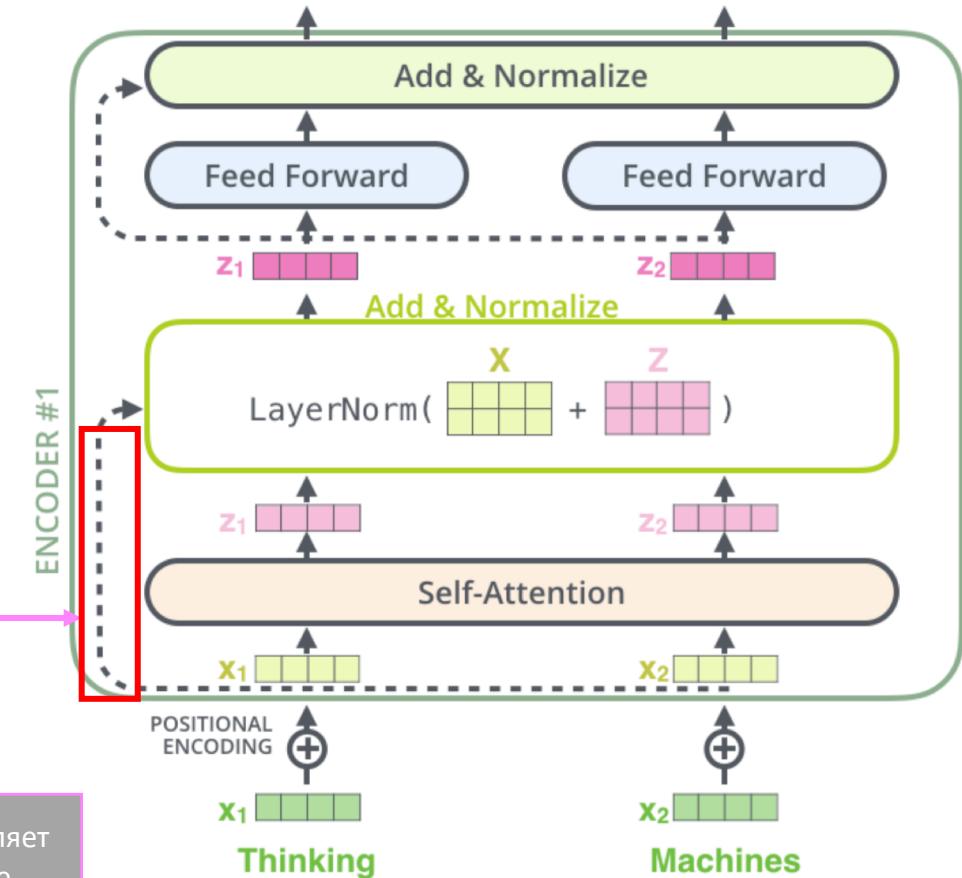
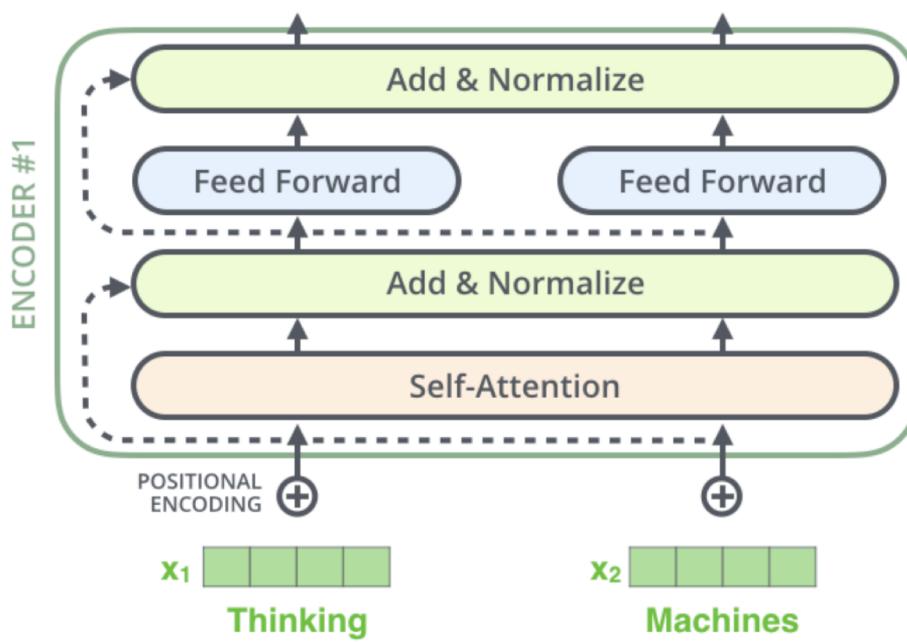
Transformer Block

Ключевые элементы трансформера:

- Self-attention – учитывает другие элементы последовательности
- Feed-Forward – добавляет нелинейность в преобразование входной последовательности



Transformer Encoder Block

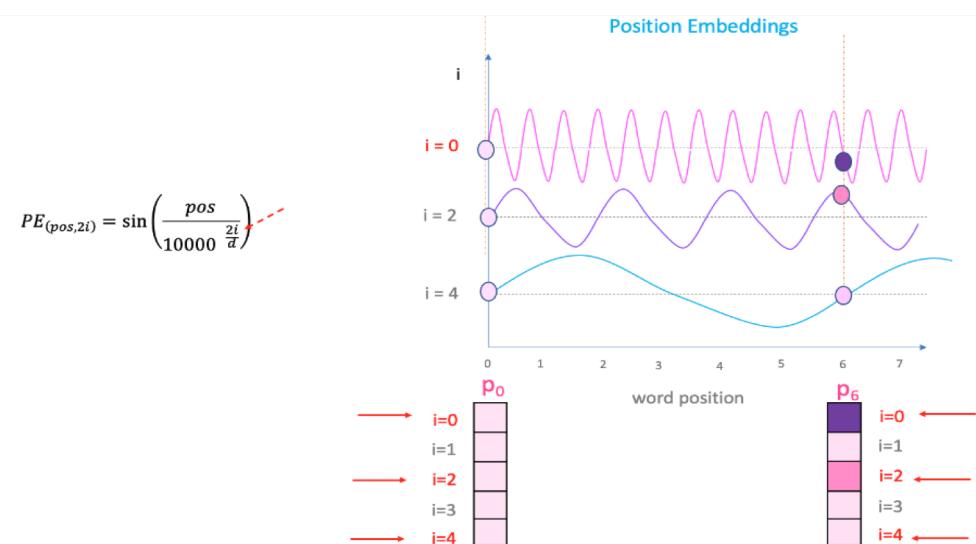
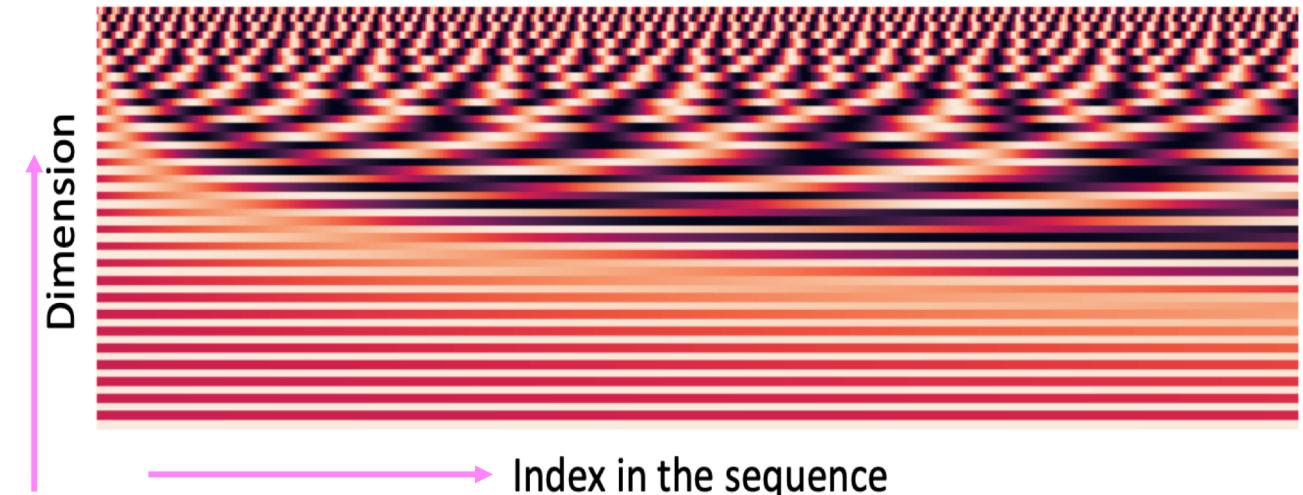


Residual connection – позволяет избегать взрыв/затухание градиентов, так как вектор не проходит через функции активации, которые обычно способствуют этому.

Positional encoding

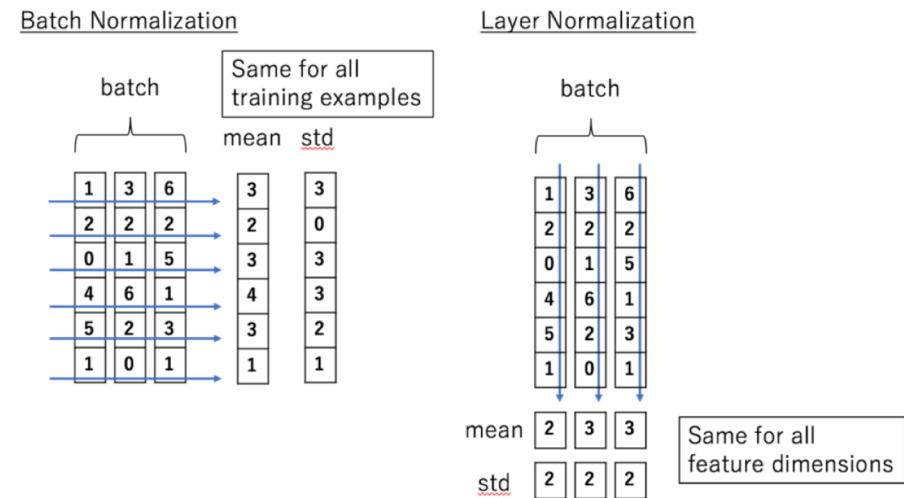
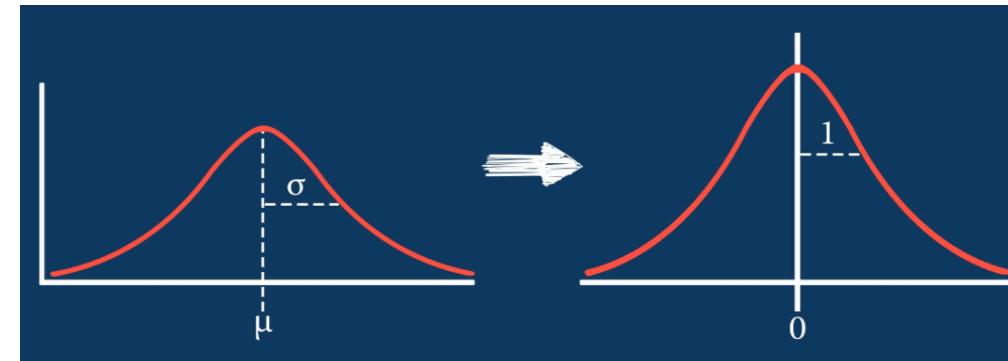
- Self-attention и сам трансформер не обладают рекуррентной архитектурой и не учитывают порядок слов в предложении. Как учитывать порядок слов в предложении? **Positional encoding**
- Представим каждый элемент последовательности вектором характеризующим позицию - $p_i \in R^d$, где $i \in \{0, 1, \dots, T\}$
- Информацию легко внести в модель – просто добавим к эмбеддингам слов вектор позиции – $\hat{e}_i = e_i + p_i$
- Операцию добавления можно сделать:
 - Сложением
 - Конкатенацией
- Требования к positional encoding:
 - **Монотонность** - $\forall x, m, n \in N: m > n \Leftrightarrow \phi(x, x + m) < \phi(x, x + n)$
 - **Инвариантность** - $\forall x_1, \dots, x_n, m \in N: \phi(x_1, x_1 + m) = \phi(x_2, x_2 + m) = \dots = \phi(x_n, x_n + m)$
 - **Симметрия** - $\forall x, y \in N: \phi(x, y) = \phi(y, x)$
- **Sinusoidal position representations:**

$$p_{pos, denc} = \begin{cases} \sin\left(\frac{pos}{10000}\frac{2*denc}{d}\right), & denc - \text{четное} \\ \cos\left(\frac{pos}{10000}\frac{2*(denc-1)}{d}\right), & denc - \text{нечетное} \end{cases}$$



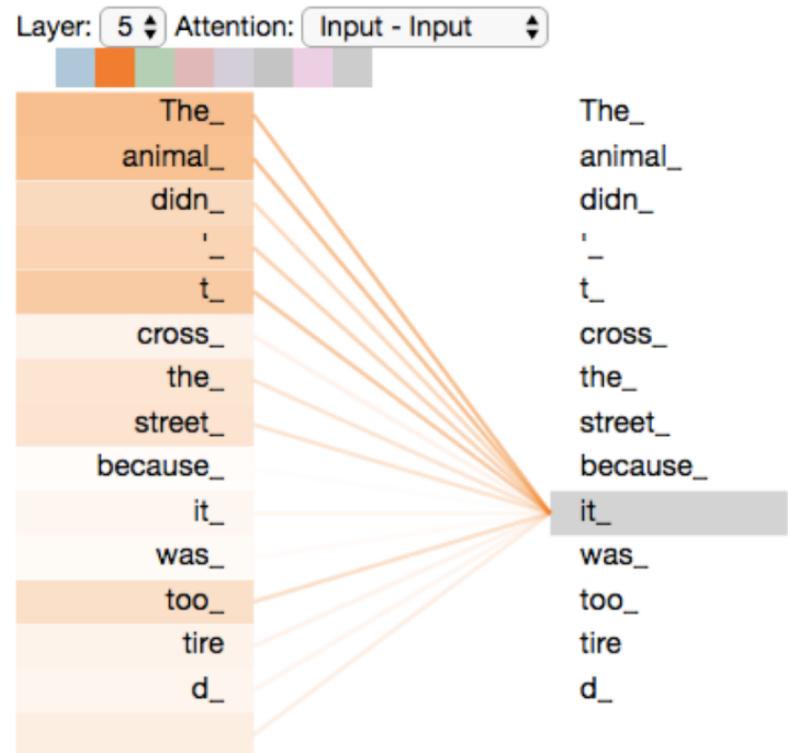
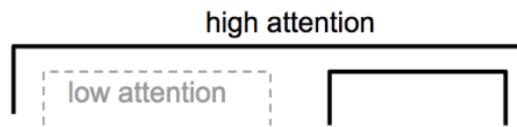
Layer Norm

- **Standardization** - $x_i = \frac{(x_i - \mu)}{\sigma}$
- **Batch Norm**
 - Идея – удаляем ненужную информацию из hidden вектора стандартизируя выход для каждого батча
 - Добавляем 2 обучаемых параметра в формулу стандартизации
 - γ – контролируем масштаб нормализации
 - β – контролируем воздействие операции
 - $\widehat{batch} = \frac{(batch - \mu_{batch})}{\sigma_{batch} + \epsilon} * \gamma + \beta$
 - Плюс:
 - Быстрее обучается модель
 - Проблемы:
 - Не очень эффективен при маленьком размере батча
 - Если нет батчей – вообще не используется
- **Layer Norm**
 - Идея – стандартизуем не по параметрам распределения внутри батчам, а по параметрам распределения после слоя
 - $\widehat{layer_output} = \frac{(layer_output - \mu_{layer_output})}{\sigma_{layer_output} + \epsilon} * \gamma + \beta$



Self-attention

She is eating a green apple.



Self-attention

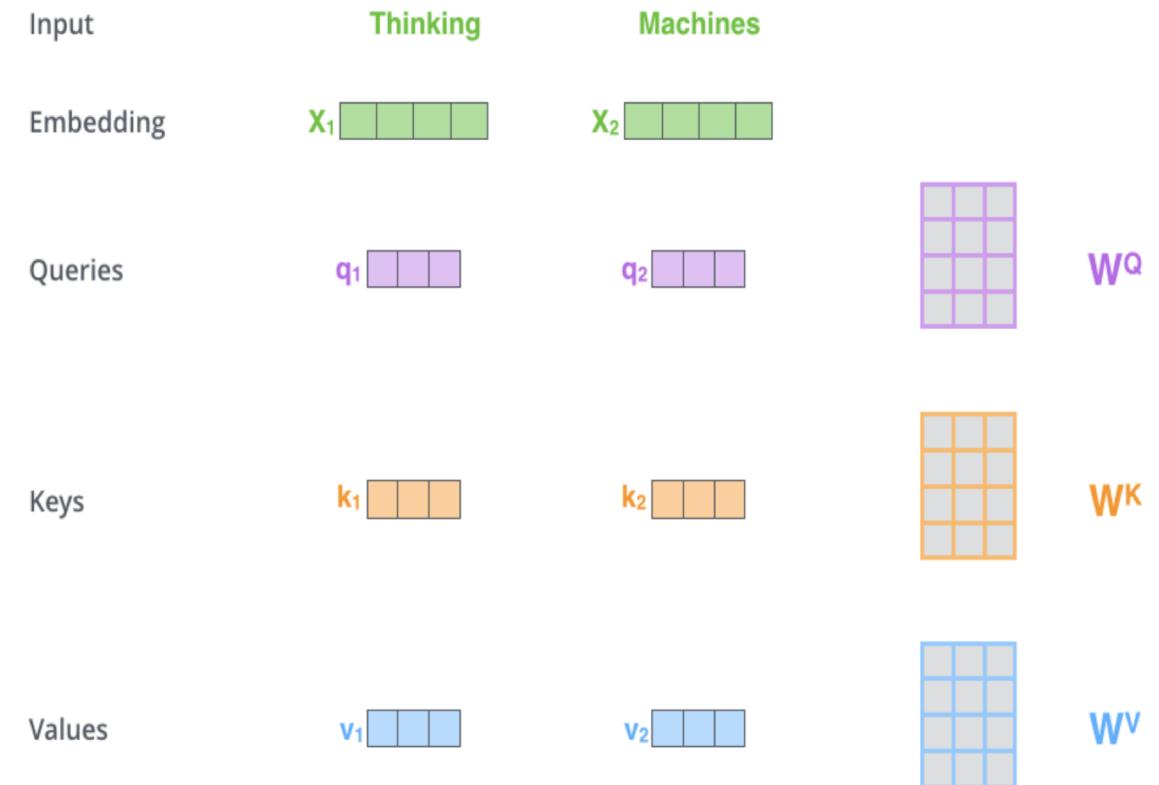
Self-attention - кодирует информацию о влиянии других слов на текущее слово.

Есть 3 вектора

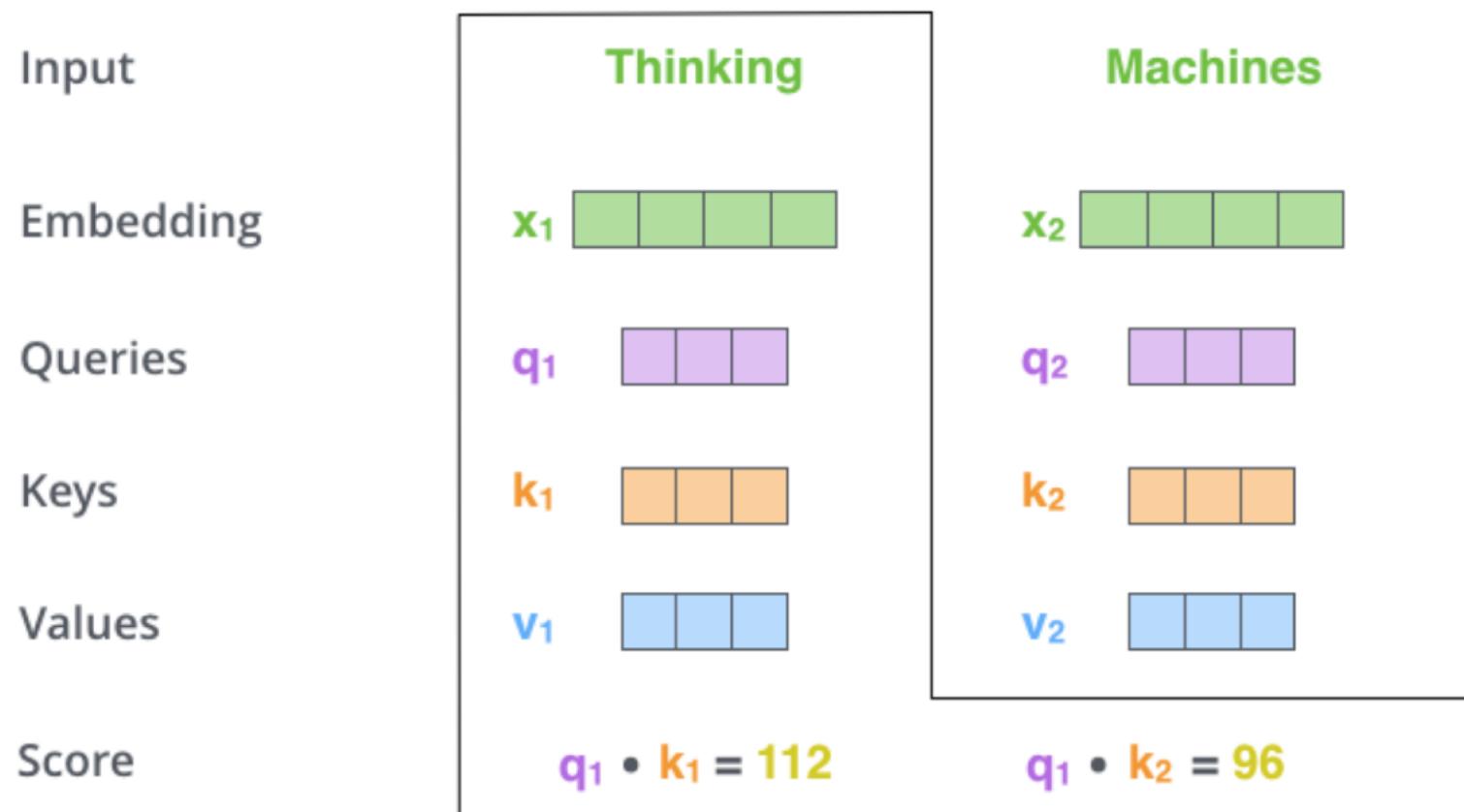
- Query – вектор текущего слова в последовательности
- Key – вектор характеризующий остальные слова
- Value – вектор, учитывающий влияние всех keys на query

$$\text{Attention vector} = \text{softmax} \left(\frac{\text{Key} * \text{Queries}}{\sqrt{d_k}} \right) * \text{Value}$$

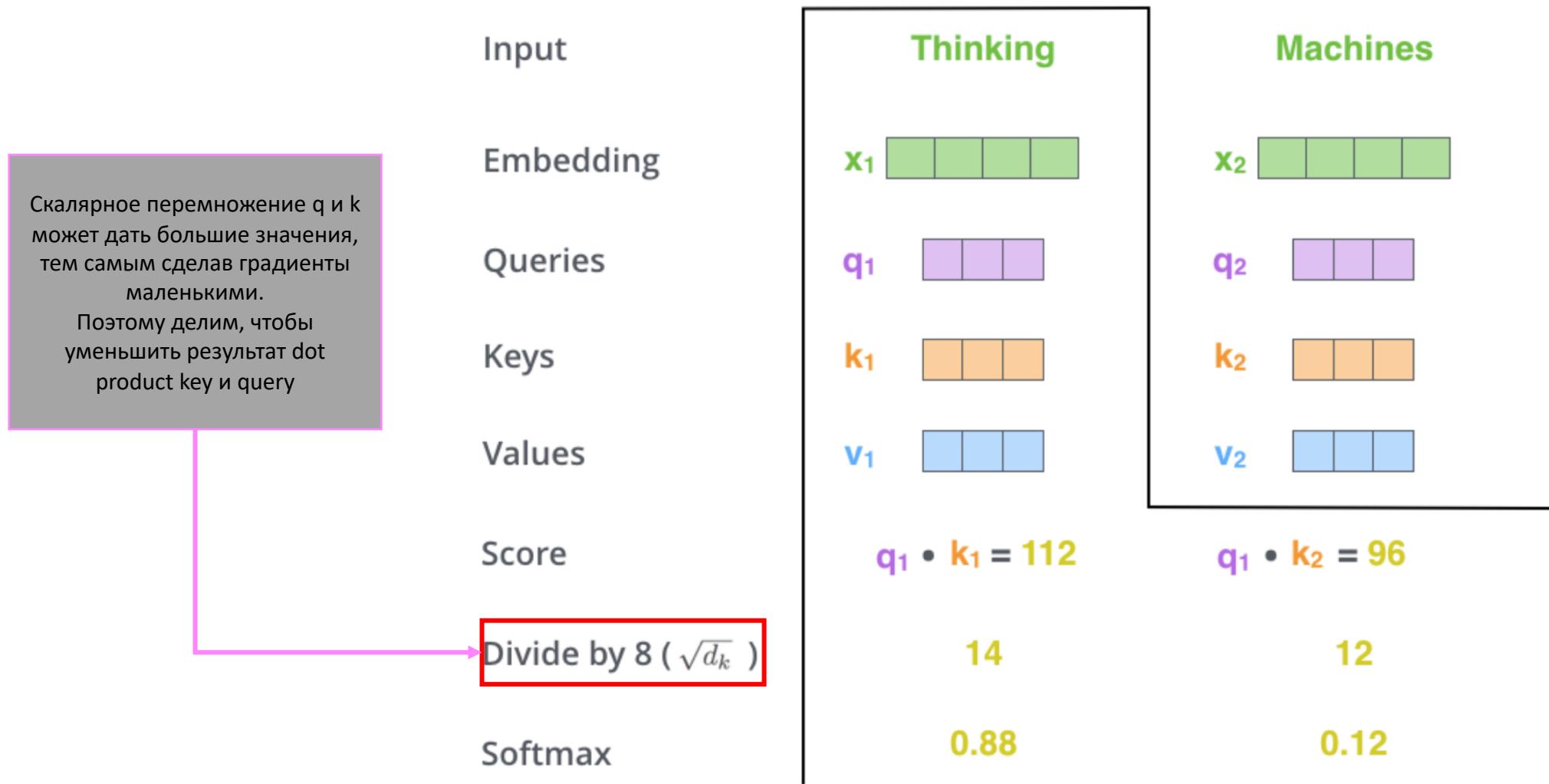
W_q, W_k, W_v – обучающиеся матрицы для преобразования query, key, value в новое пространство



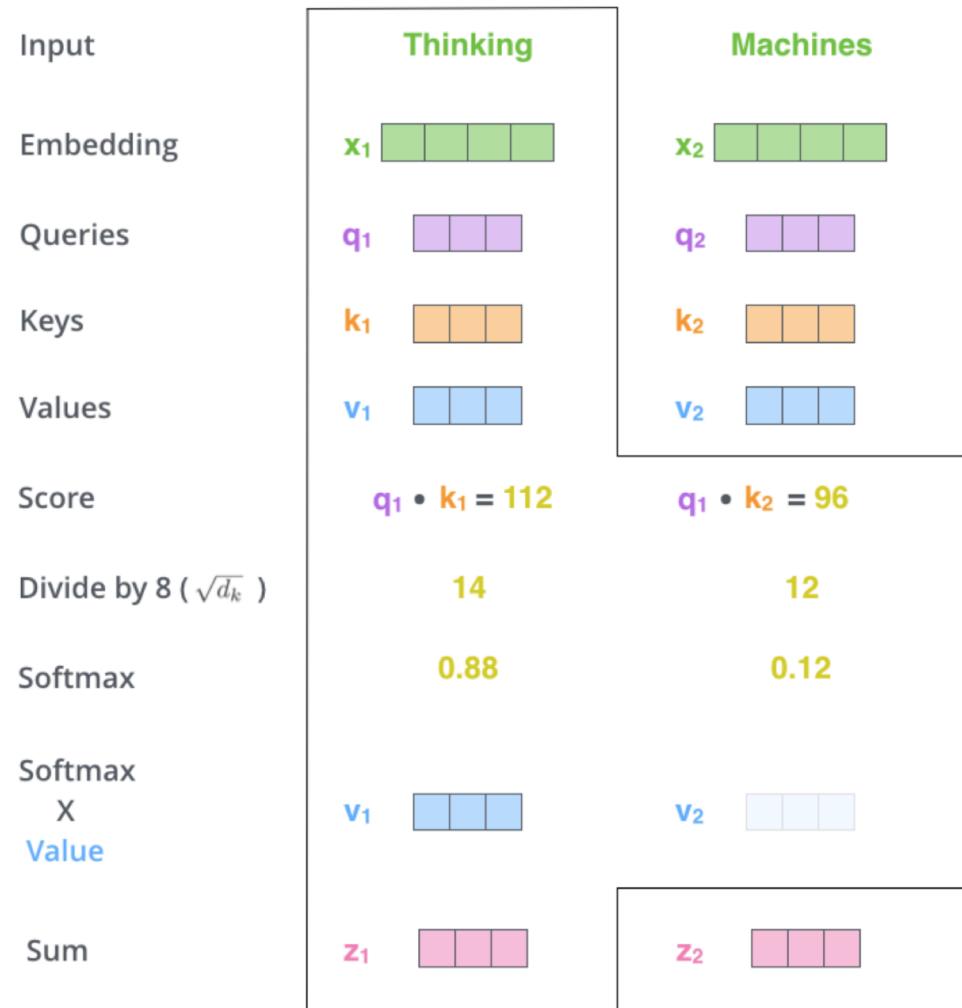
Self-attention



Self-attention



Self-attention



Self-attention

$$\mathbf{X} \times \mathbf{W}^Q = \mathbf{Q}$$

$$\mathbf{X} \times \mathbf{W}^K = \mathbf{K}$$

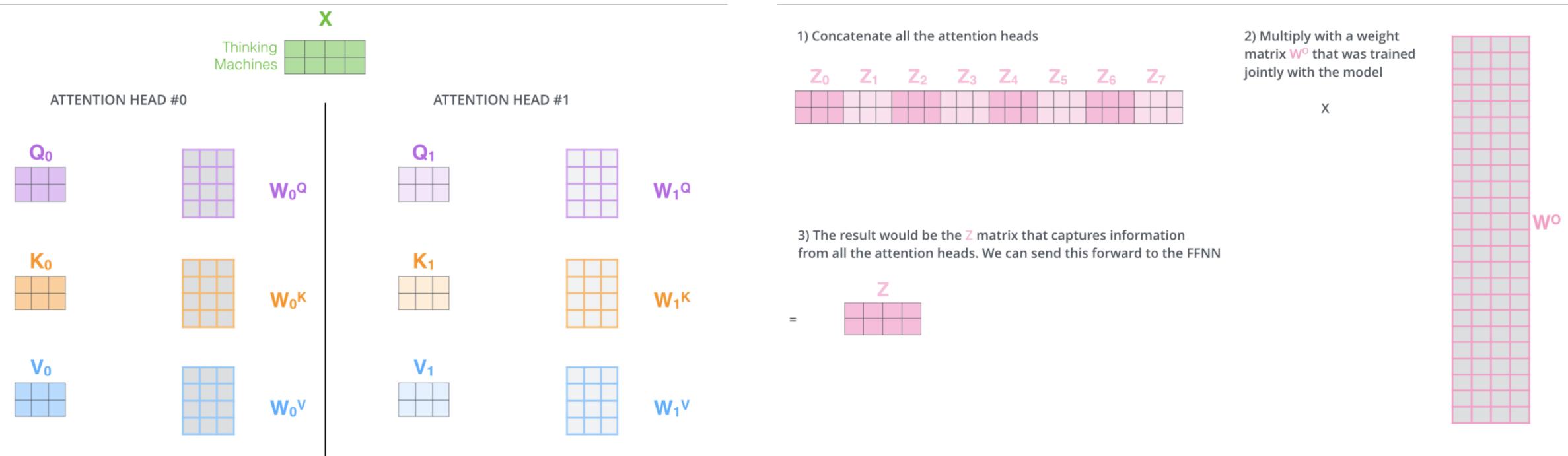
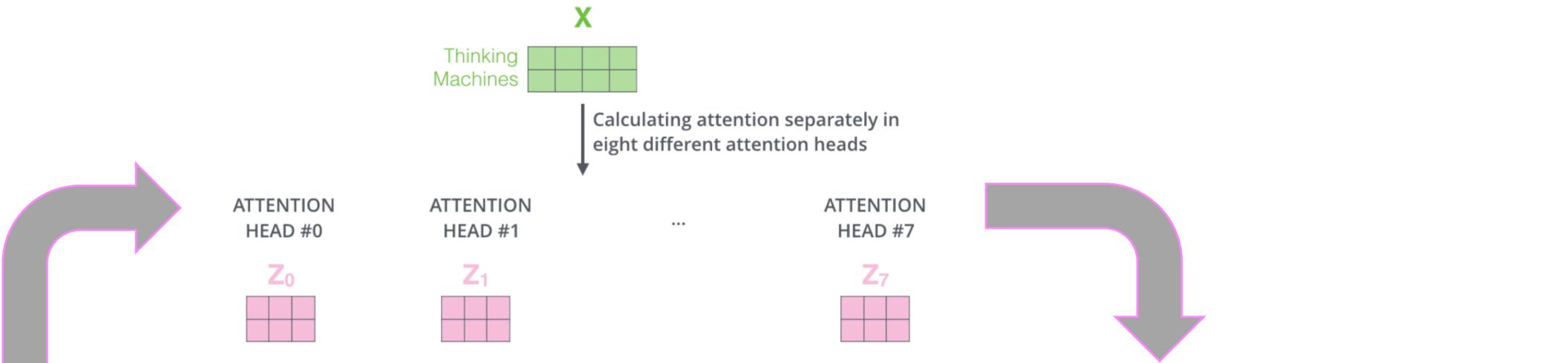
$$\mathbf{X} \times \mathbf{W}^V = \mathbf{V}$$

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

Multi-head Self-attention

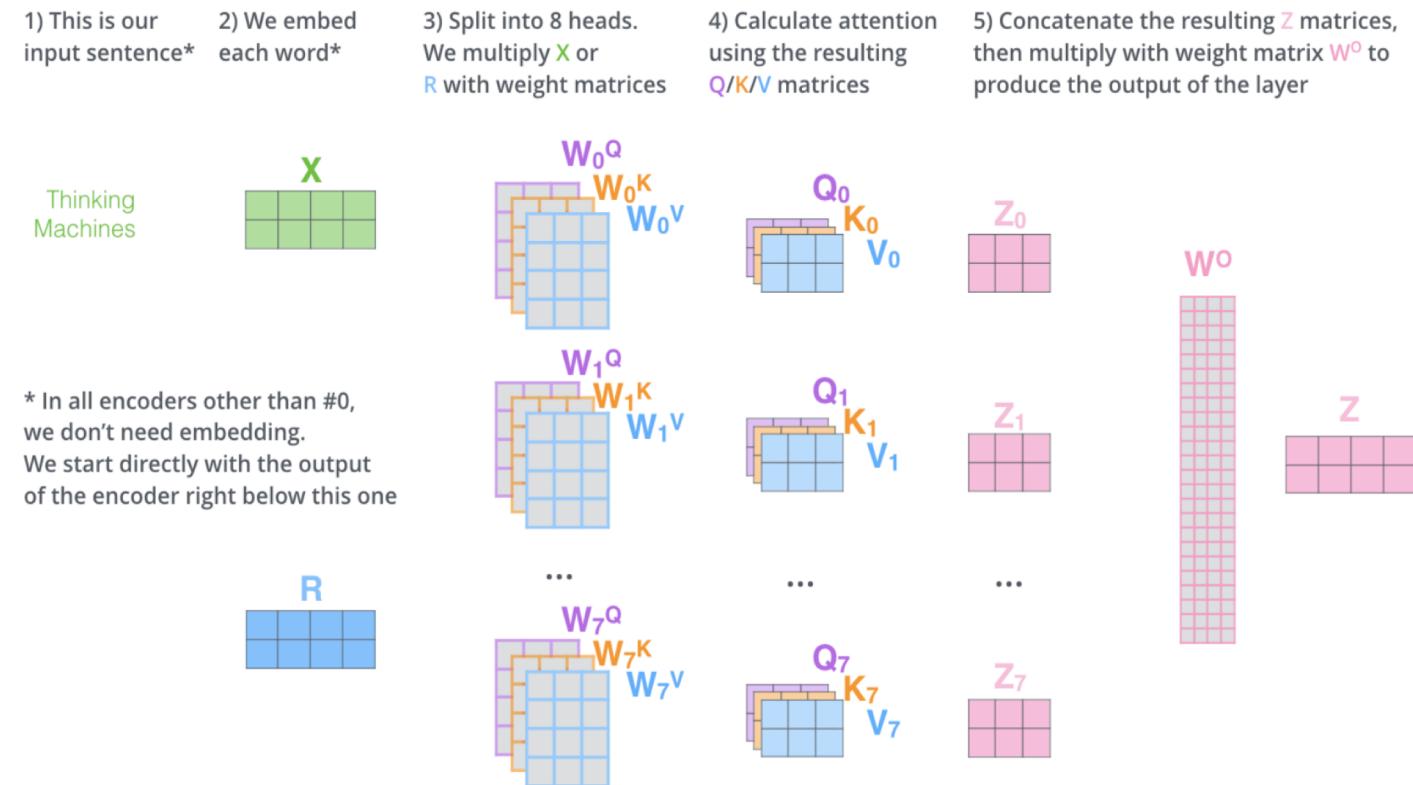
- W_k, W_q, W_v - случайно заинициализированные матрицы, которые позволяют оценивать влияние остальных слов на текущее
- Множество матриц W_k^i, W_q^i, W_v^i позволяют оценить это влияние с разных сторон
- Давайте увеличим количество self-attention матриц

Multi-head Self-attention



Multi-head Attention Full Picture

- Multi-head attention – позволяет увеличить количество информации о словах, которые влияют на текущее слово за счет увеличения кол-ва матриц.
- Каждое слово получает столько векторных взвешенных представлений, сколько есть “голов” у механизма внимания.



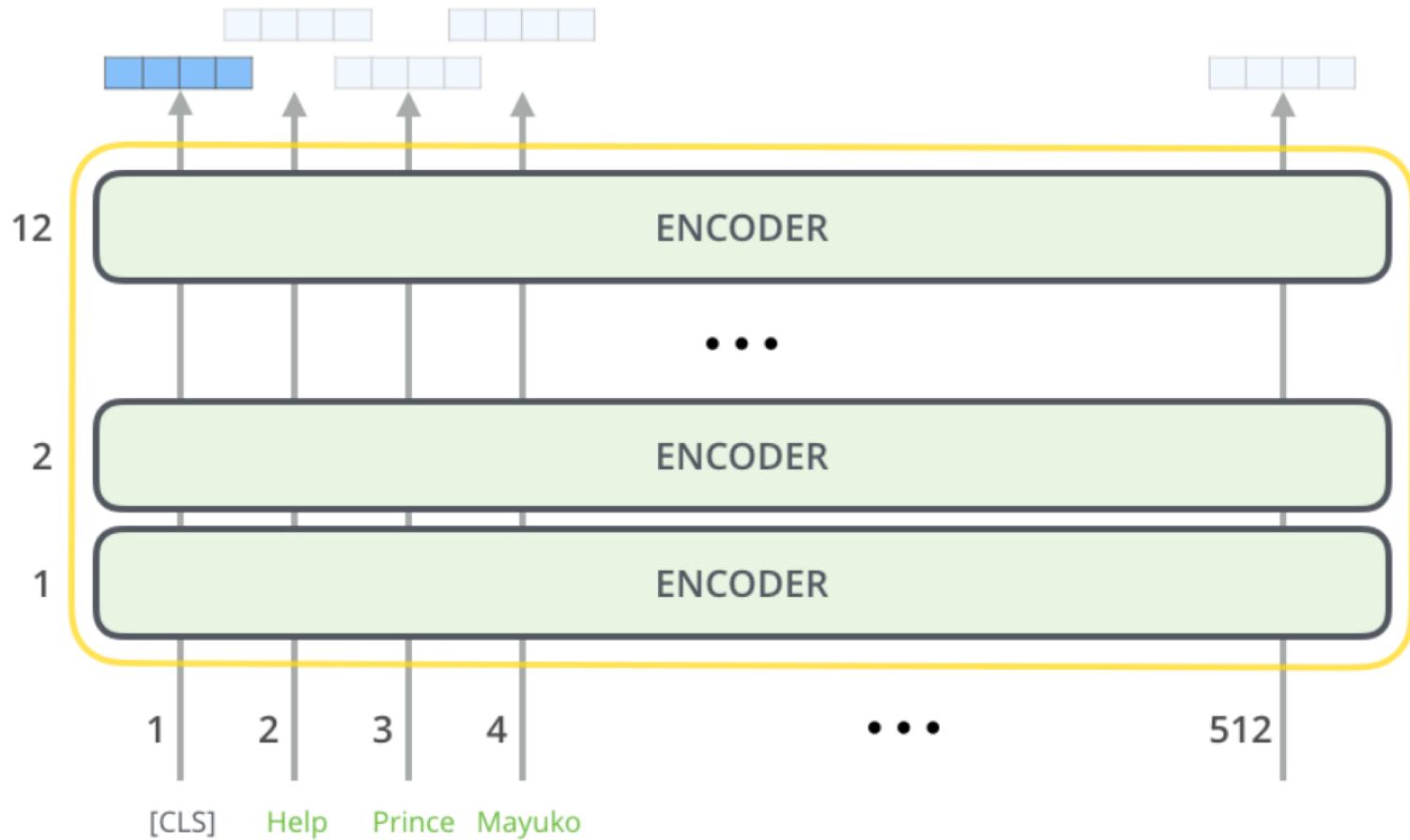
Проблемы трансформера

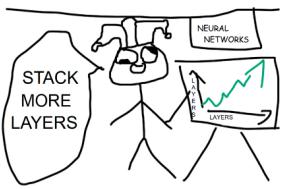
- Проблемы
 - $O(n^2)$, где n – длина последовательности в self-attention
 - Чем больше последовательность, тем дольше работает трансформер
 - В RNN сложность росла линейно
 - Способ обозначения позиции не точно отображает порядок слов

Transfer Learning

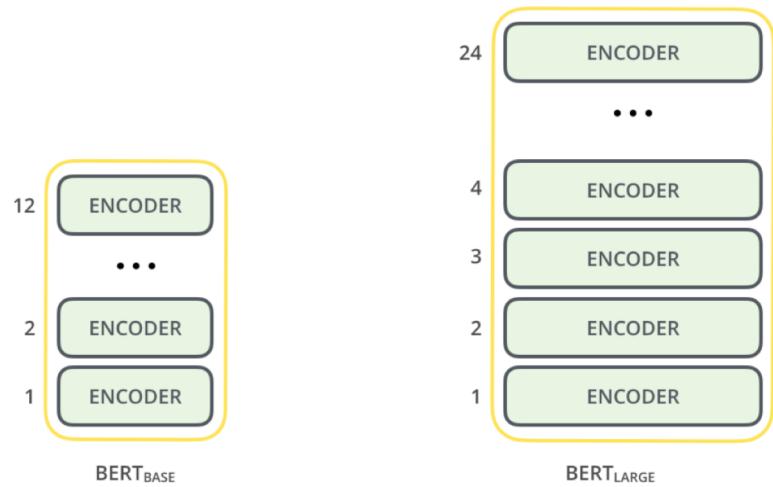
- Основная идея – использовать большую предобученную модель языкового моделирования для решения небольшой конкретной задачи.
- Способы использования
 - Векторизация и использование, как эмбеддингов
 - Fine-tuning
- Дешевый способ значительно улучшить качество предсказания

BERT(Bidirectional Encoder Representations from Transformers)

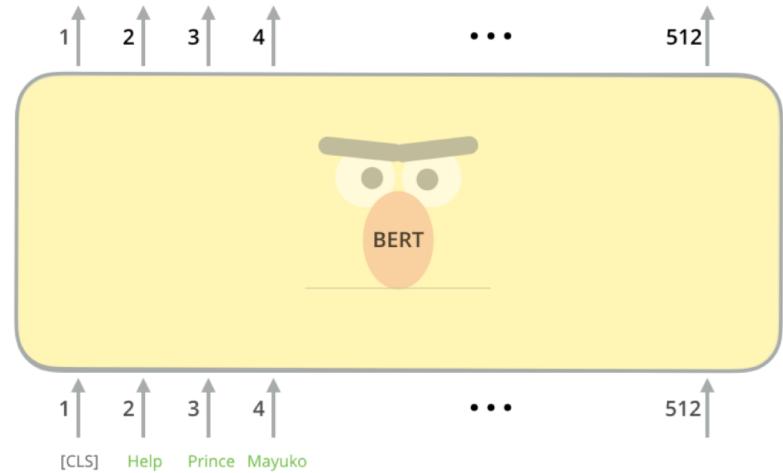




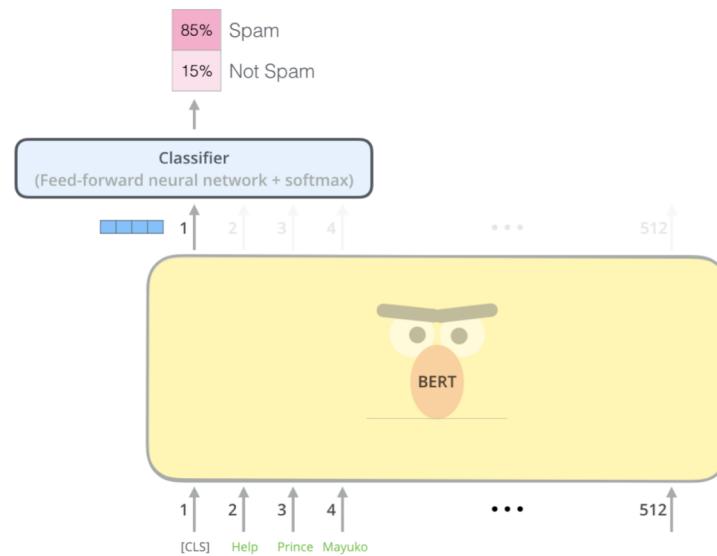
BERT Lifecycle



Стек энкодеров



Один большой энкодер



Инструмент для решения задач NLP

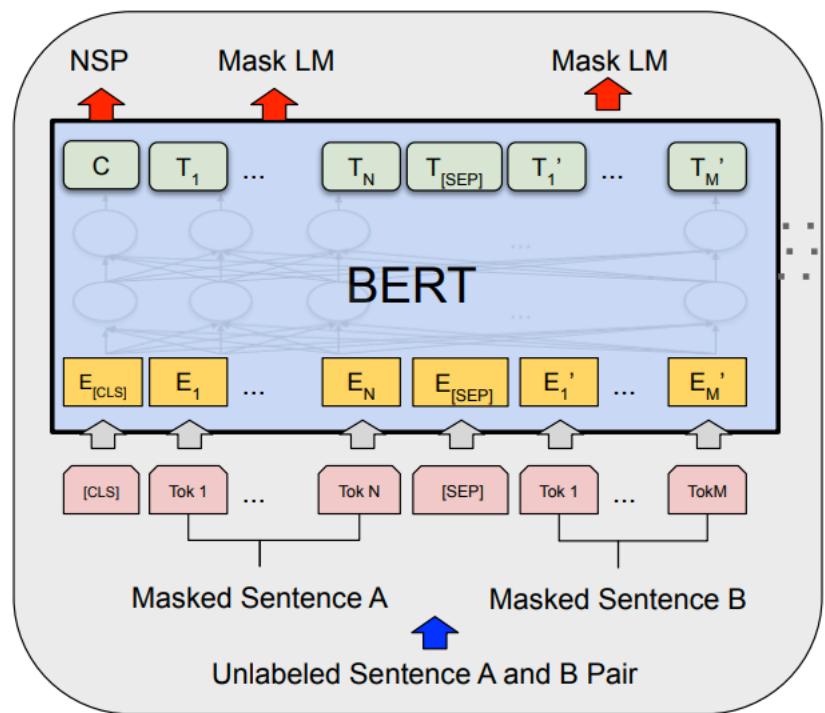
BPE токенизация

- В процессе обучения получаются огромные словари:
 - В русском языке очень сложная морфология -> огромное количество словоформ
 - Низкий показатель perplexity
- Для того чтобы уменьшить размер словаря в языковом моделировании есть множество приемов на уровне слов, н-грамм, символов
 - Современный подход – уменьшение размера словаря на разбиение входного слова на сабворды
 - На этапе тренировки и тестирования входная последовательность разбивается на сабворды
- BPE-алгоритм(byte-pair encoding)
 - Начинаем с множества символов(алфавит, цифры и тд) и символа конца слова(#)
 - Используя большой корпус текста находим самые часто встречающиеся вместе буквы – (а, м). Добавляем в словарь сочетание **ар**
 - Создаем словарь до нужного размера(В BERT-base – 30k)
 - Входное предложение жадно токенизуем по словарю

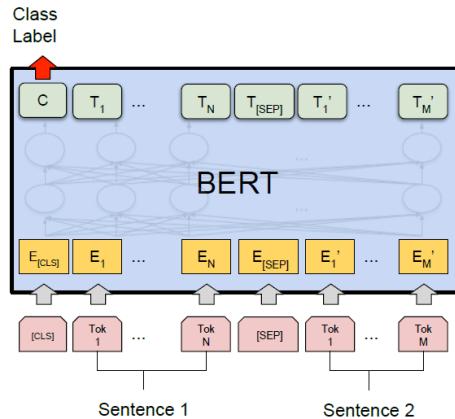
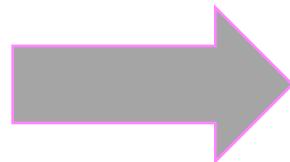
```
#tokenizer example
string_example = 'i have a funny story about dinosours'
print(TOKENIZER.tokenize(string_example))
print(TOKENIZER.encode(string_example))

['i', 'have', 'a', 'funny', 'story', 'about', 'dino', '##so', '##urs']
[101, 1045, 2031, 1037, 6057, 2466, 2055, 22412, 6499, 9236, 102]
```

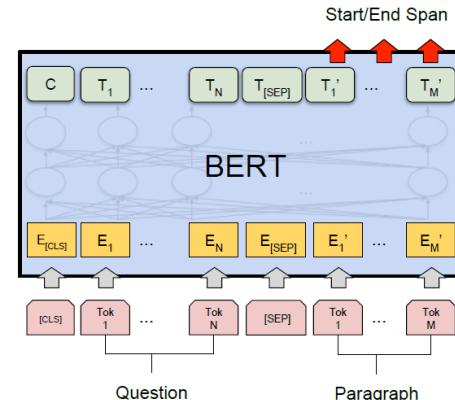
BERT training phases



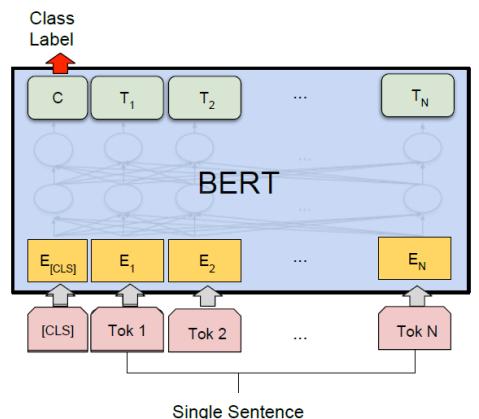
Pretraining



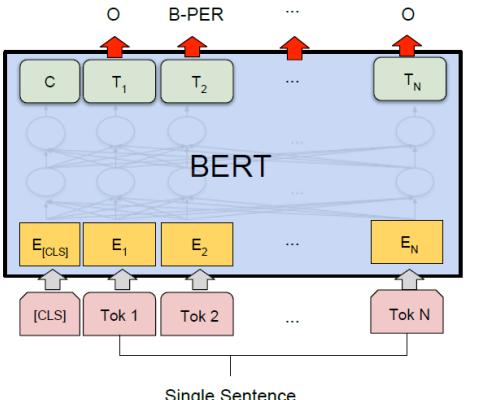
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



(c) Question Answering Tasks:
SQuAD v1.1



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

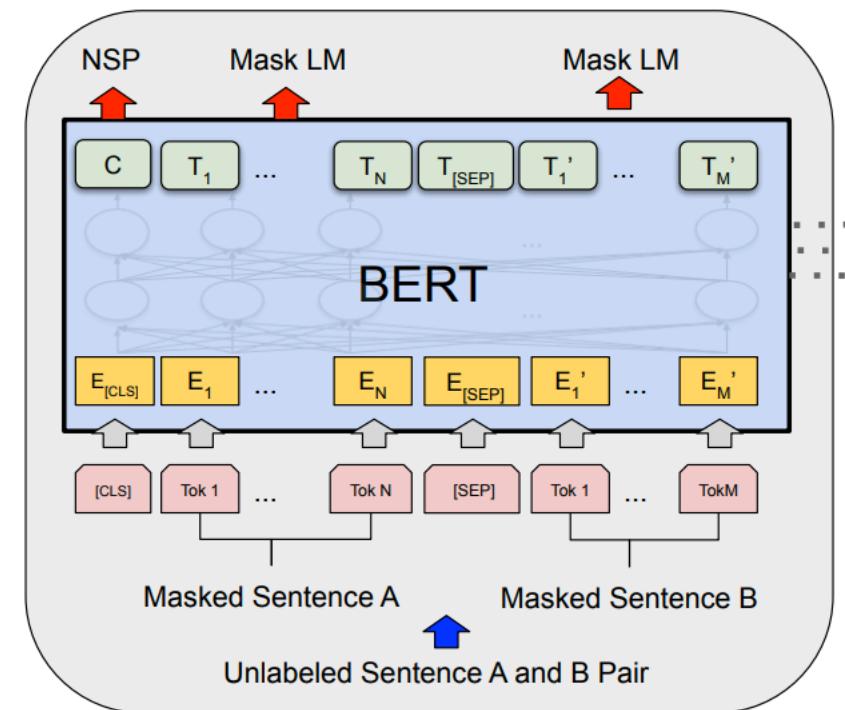


(d) Single Sentence Tagging Tasks:
CoNLL-2003 NER

Finetuning

BERT pre-training

- $BERT \text{ pretrain loss} = L_{NSP} + L_{MLM}$
- **Лосс в обоих случаях – сумма cross-entropy loss на двух задачах**
 - Предсказание некоторого слова в предложении – оценка распределения по всему словарю(masked language modeling)
 - Я пошел в MASK и купил MASK
 - Определение логичности следующего предложения – классификация(next sentence prediction)
 - Я пошел в магазин. Там я купил молоко. – True
 - Я купил собаку. В Москве холодаает. – False



Обучение BERT

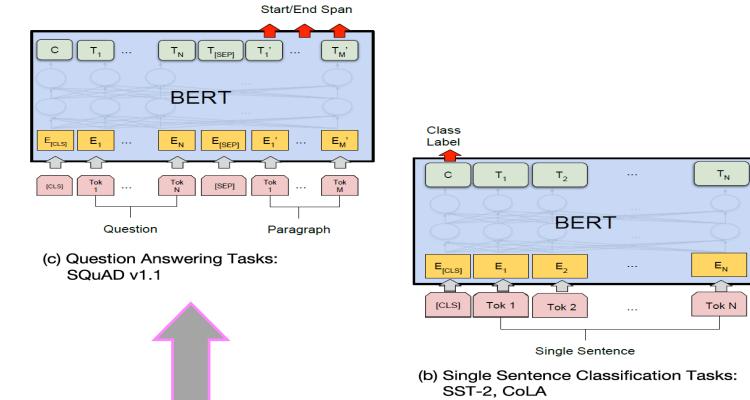
- Обучение с учителем
 - Маскируется 15% токенов в предложении – решается задача MLM
 - Берется либо последующее предложение из текста, либо случайное – решается задача NSP
- Модели BERT
 - BERT-base – 12 трансформеров, hidden dim 768, 12 attention heads, 110 миллионов параметров
 - BERT-large – 24 трансформера, hidden dim 1024, 16 attention heads, 340 миллионов параметров
- Датасеты
 - Wikipedia
 - Корпуса книг
- Обучение
 - BERT pretraining делают на TPU
 - BERT finetuning делают на GPU

BERT results

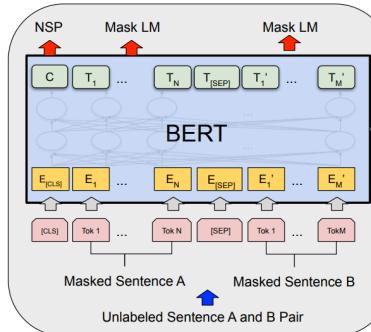
Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX
1	ERNIE Team - Baidu	ERNIE	↗	90.9	74.4	97.8	93.9/91.8	93.0/92.6	75.2/90.9	91.9	91.4	97.3	92.0	95.9	51.7
2	DeBERTa Team - Microsoft	DeBERTa / TuringNLVRv4	↗	90.8	71.5	97.5	94.0/92.0	92.9/92.6	76.2/90.8	91.9	91.6	99.2	93.2	93.2	52.2
3	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6
4	Alibaba DAMO NLP	StructBERT + TAPT	↗	90.6	75.3	97.3	93.9/91.9	93.2/92.7	74.8/91.0	90.9	90.7	97.4	91.2	94.5	49.1
5	PING-AN Omni-SinTic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
6	T5 Team - Google	T5	↗	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
7	Microsoft D365 AI & MSR AI & GATECHMT-DNN-SMART		↗	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
8	Huawei Noah's Ark Lab	NEZHA-Large		89.8	71.7	97.3	93.3/91.0	92.4/91.9	75.2/90.7	91.5	91.3	96.2	90.3	94.5	47.9
9	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	↗	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
10	ELECTRA Team	ELECTRA-Large + Standard Tricks	↗	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
11	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	↗	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1

<https://gluebenchmark.com/leaderboard>

Finetune



Pretrain



Input

Input	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	# ^{ing}	[SEP]
Token Embeddings	$E_{[CLS]}$	E_{my}	E_{dog}	E_{is}	E_{cute}	$E_{[SEP]}$	E_{he}	E_{likes}	E_{play}	$E_{#ing}$	$E_{[SEP]}$
Segment Embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B	E_B
Position Embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

BERT Summary

Процесс обучения

1. Токенизируем BPE алгоритмом текст - T
2. Считаем позиционные эмбеддинги - E_{pos}
3. Считаем с помощью матрицы эмбеддингов эмбеддинги для токенов – E_t
4. Считаем с помощью матрицы эмбеддингов эмбеддинги для сегмента - E_s
5. $\hat{E} = E_{pos} + E_t + E_s$
6. Forward-pass через стек энкодеров
7. 2 выхода
 1. Pooler output – mean/max hidden states pooling
 2. Hidden states –hidden представление для токенов последовательности

Фазы обучения

- Pre-train фаза – обучается на сумме двух ошибок NSP и MLM
- Fine-tune фаза – дообучается на специальной задаче(unfreeze layers)

Датасет

- Большой набор данных – википедия + художественные произведения

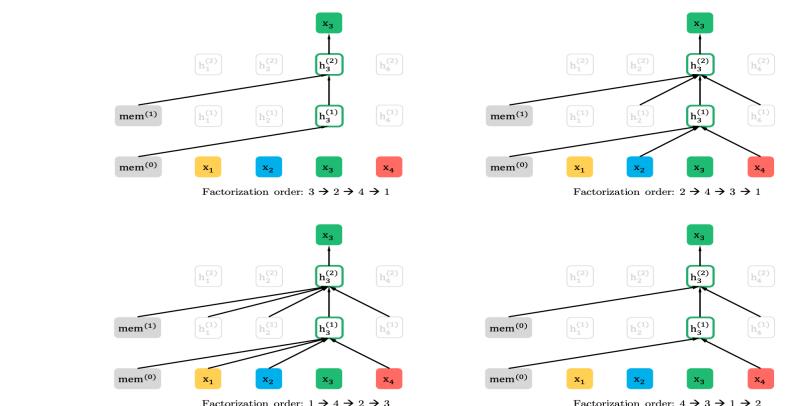
Результат

- Большая языковая модель для задач NLP
- Контекстно зависимые словарные эмбеддинги

BERT Family

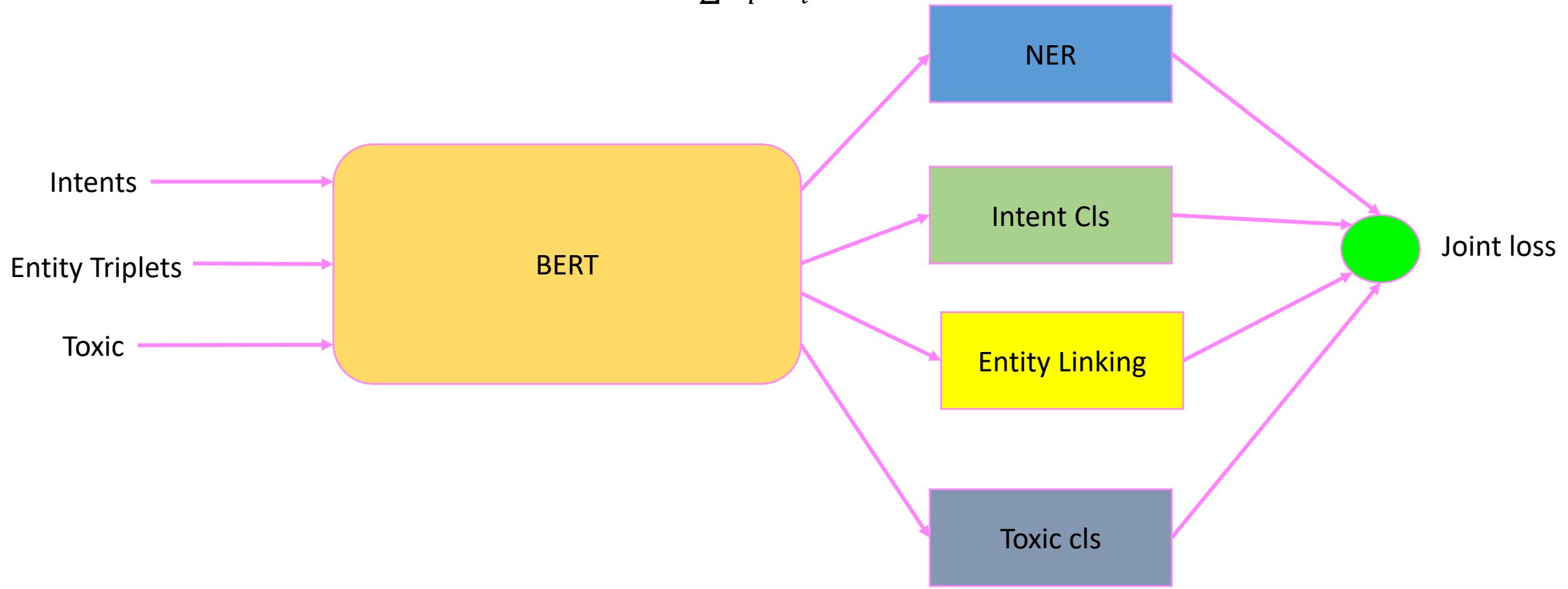
- RoBERTa - <https://arxiv.org/pdf/1907.11692.pdf>
 - Увеличение количества эпох и увеличение количества данных влияет на качество BERT
- ALBERT - <https://arxiv.org/pdf/1909.11942.pdf>
 - Уменьшенная матрица эмбеддингов за счет факторизации
 - Веса всех трансформеров общие
- XLNET - <https://arxiv.org/pdf/1906.08237.pdf>
 - Изменение positional embedding
 - Обучение авторегрессионной модели на предложениях, где слова в предложении случайным образом переставлены
 - Модель учится по возможным сочетаниям других токенов предсказать целевой токен

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4
BERT _{LARGE}						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
XLNet _{LARGE}						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6



Multi-task learning

$$\begin{aligned} \text{Joint loss} &= \sum_{i=0} \text{task loss}_i * \text{alpha}_i \\ \sum \text{alpha}_i &= 1 \end{aligned}$$



Полезные ссылки

- Общее описание BERT(оттуда картинки) -
<http://jalammar.github.io/illustrated-bert/>
- Transformer and Attention реализация -
<https://nlp.seas.harvard.edu/2018/04/03/attention.html>
- Attention is all you need - <https://arxiv.org/abs/1706.03762>
- BERT - <https://arxiv.org/pdf/1810.04805.pdf>