

ИНТЕЛЛЕКТ ➤

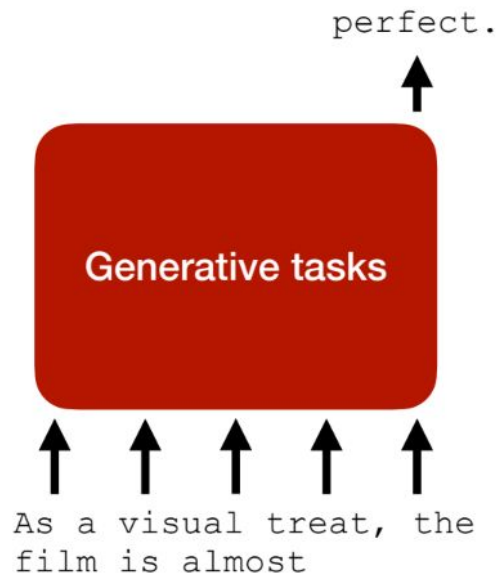
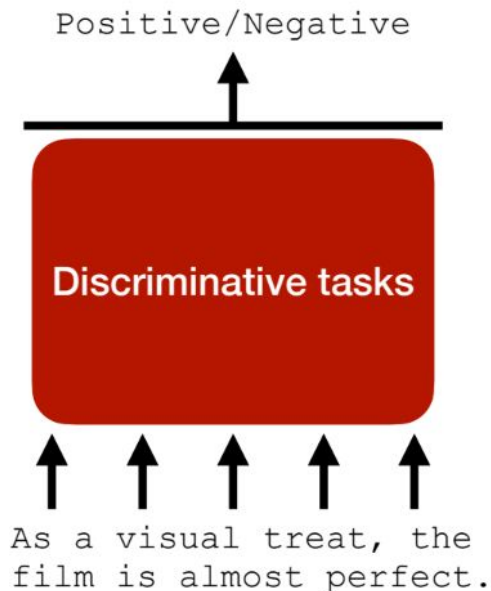
Механизм внимания, архитектура трансформер

к.ф.-м.н. Тихомиров Михаил Михайлович

научный сотрудник
НИВЦ МГУ имени М. В. Ломоносова

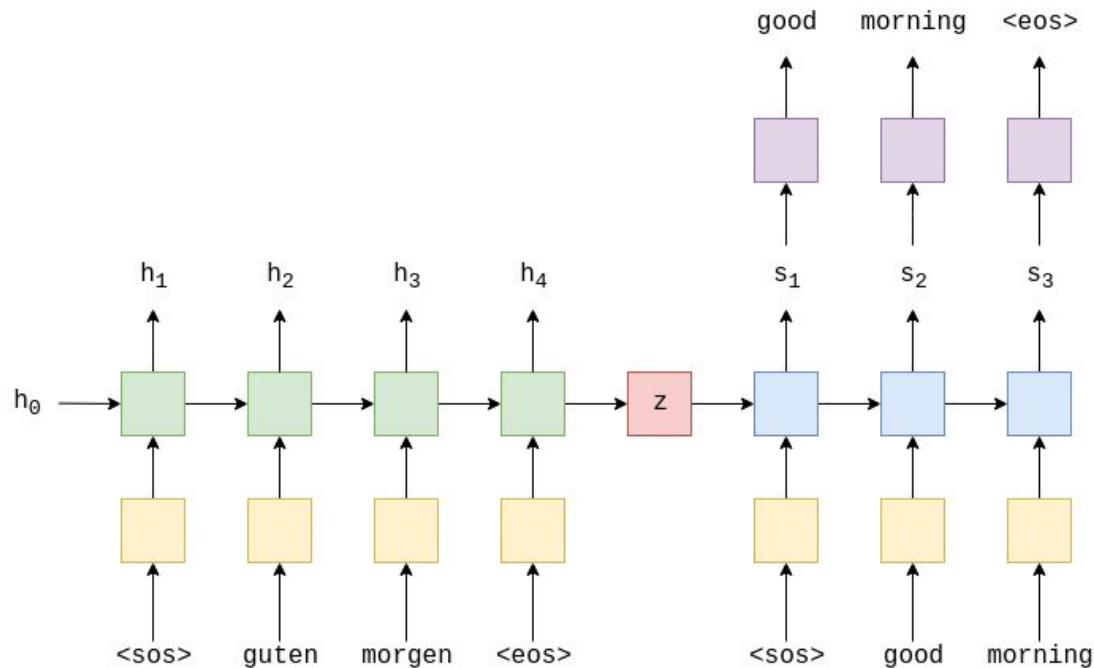
NLP задачи

- Дискриминативные задачи: классификация, NER, и др.
- Генеративные задачи: перевод, аннотирование (summarization), языковое моделирование



Seq2Seq до трансформеров

- Вектор финального состояния должен хранить **всю** информацию из предложения
- По сути является векторным представлением (эмбеддингом) предложения
- Теряет информацию на длинных последовательностях



Механизм внимания (2014)

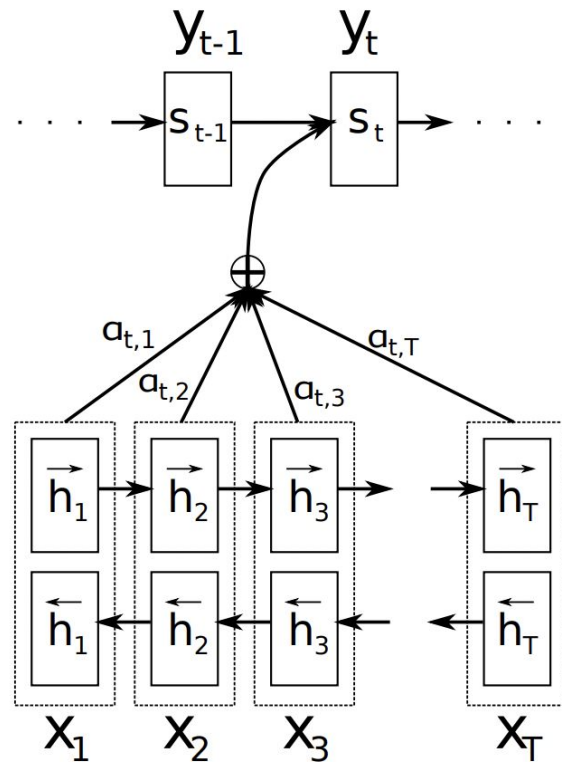
Автокодирующая модель состоит из:

- **Encoder**(text) -> vector:
переводит текст в необходимое векторное представление
- **Decoder**(vector) -> text:
расшифровывает представление в ответ модели

Проблема: в vector помещается только общий контекст

Решение: сохранять векторы для каждого слова и подбирать нужные под каждый шаг decoder

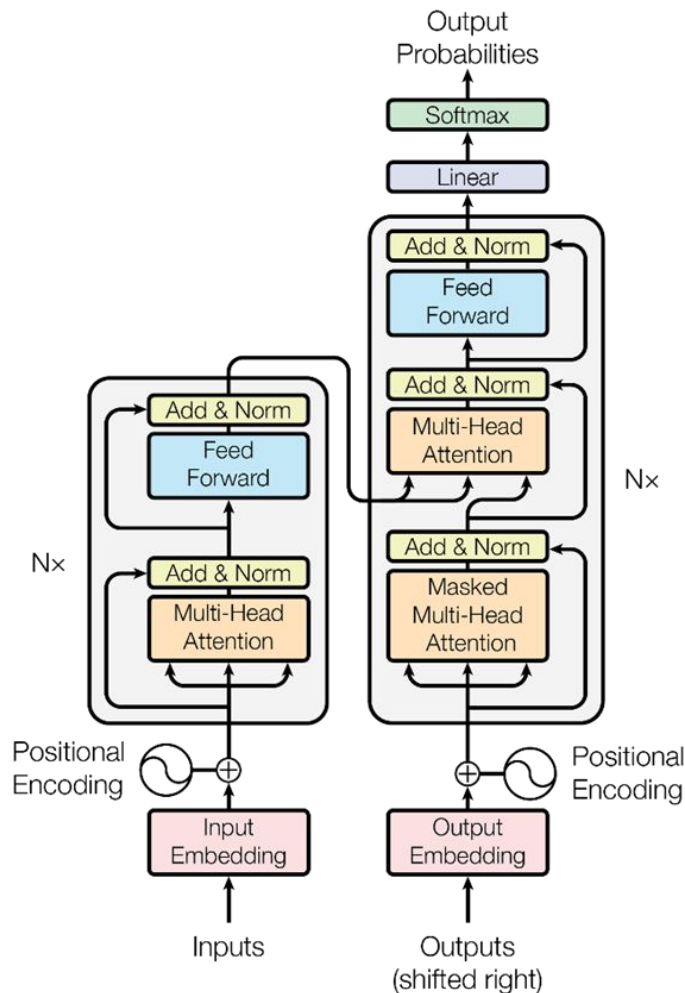
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j.$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$
$$e_{ij} = a(s_{i-1}, h_j)$$



Архитектура трансформер

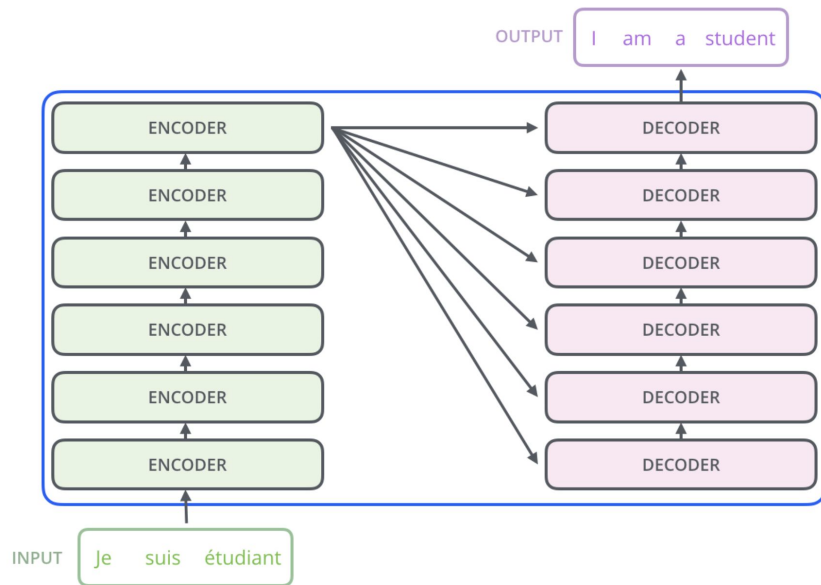
Transformer (2017)

- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



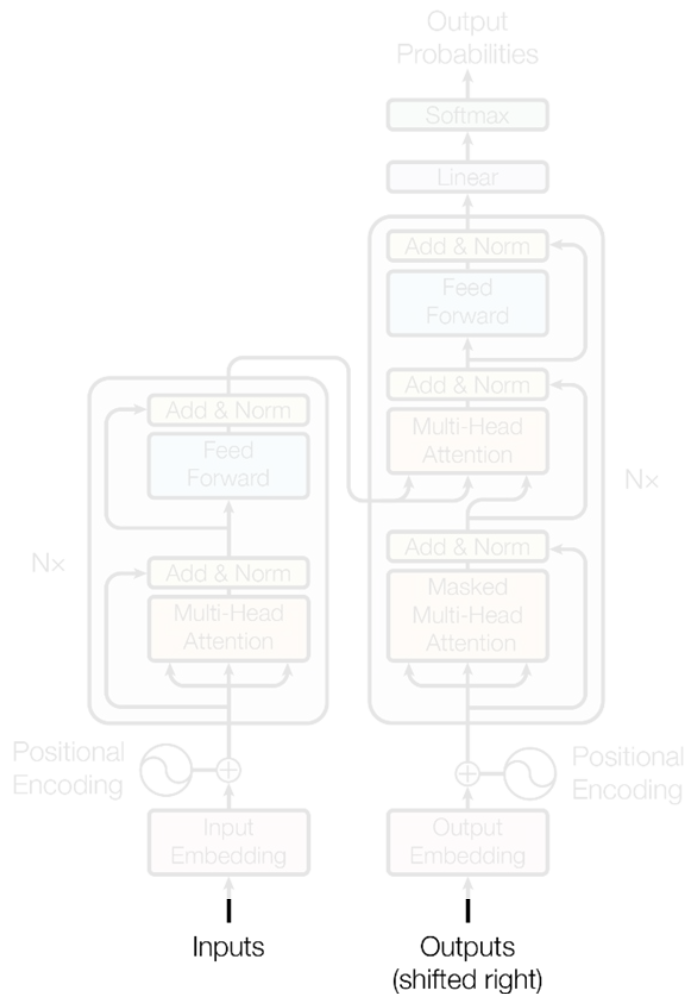
Transformer (2017)

- Исходно **encoder-decoder** архитектура
- Блоки архитектурно эквивалентны** и последовательно преобразует входной вектор в выходной вектор той же размерности
- Сами веса слоев в блоках при этом отличаются



Transformer (2017)

- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



Токенизация текста

Цель - отобразить текст в последовательность id

- Символы?
 - Небольшой словарь, не более 256 токенов!
 - Но длина последовательности будет огромной.
- Каждое слово - токен?
 - Количество различных слов огромное.
 - Если отсекаать по частоте, будут UNK токены
- Решение: subword tokenization
 - Словарь включает в себя как символы, так и би-три-н граммы (символьные), на основе частоты в корпусе.

Token count
41

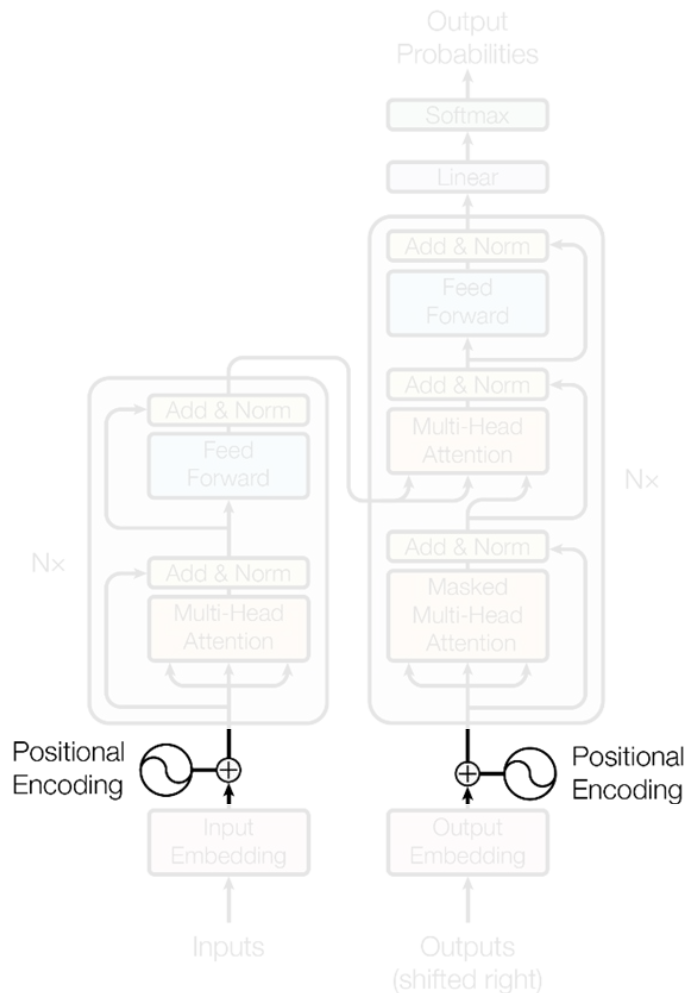
Price per prompt
\$0.000041

Большие языковые модели в вопросно-ответных системах: от трансформера до собственного чат бота

[61432, 17461, 30480, 1532, 46410, 9136, 4655, 90877, 51736, 71239, 61642, 5927, 5927, 29256, 42057, 13999, 12, 13337, 48074, 44786, 93099, 1506, 10693, 25, 20879, 11047, 35682, 2297, 57719, 91883, 57297, 5524, 14082, 20812, 5372, 39900, 17756, 8131, 14391, 13337, 1506]

Transformer (2017)

- Токенизация слов
- **Позиционное кодирование**
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



Позиционные эмбединги

- В отличии от RNN трансформер смотрит на входную последовательность как на мешок слов, нет информации о позиции.
- Решение - позиционные “эмбединги” как добавка к эмбедингам токенов.

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

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

pos - позиция в последовательности, i - координата вектора.

“We also experimented with using learned positional embeddings instead, and found that the two versions produced nearly identical results”

Позиционные эмбединги

$$\mathbf{X}_t + \vec{p}_t = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_{d \times 1}$$
$$\omega_k = \frac{1}{10000^{2k/d}}$$

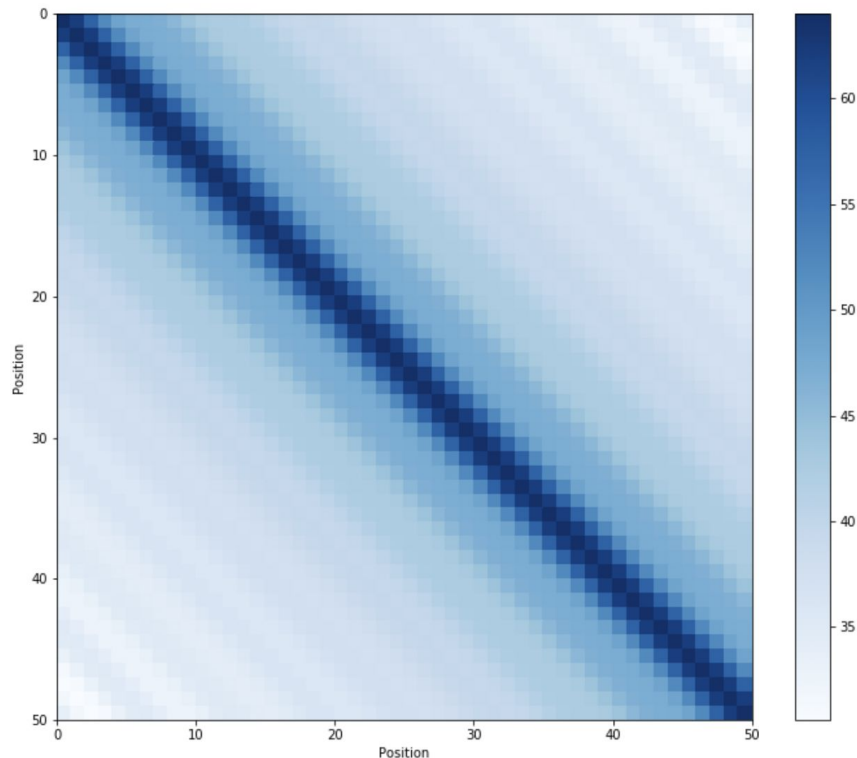
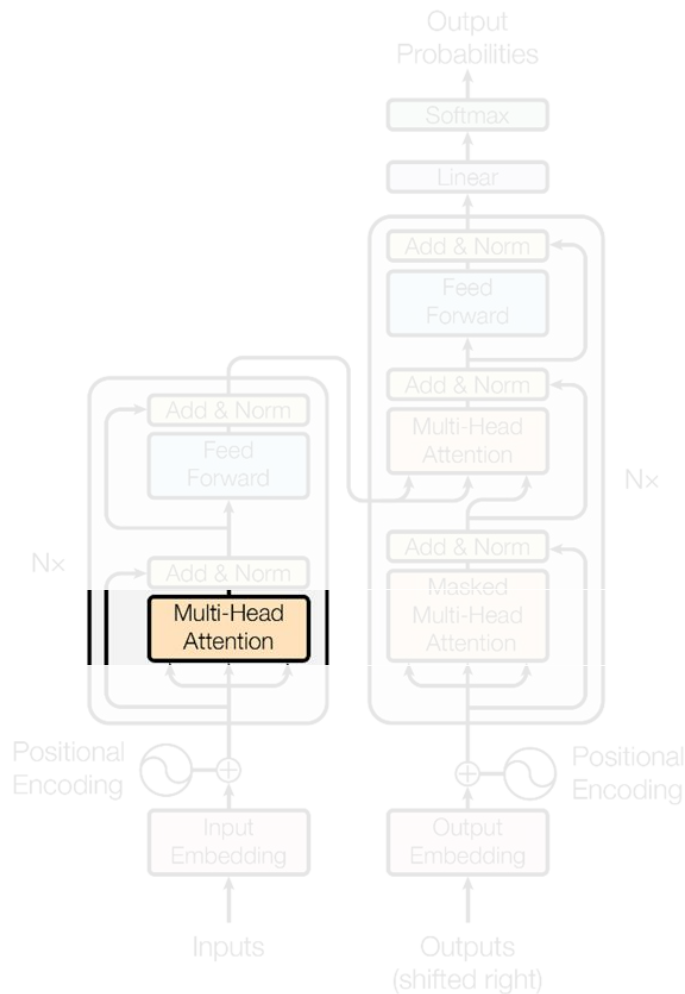


Figure 3 - Dot product of position embeddings for all time-steps

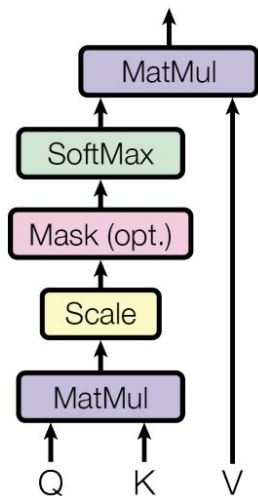
Transformer (2017)

- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)

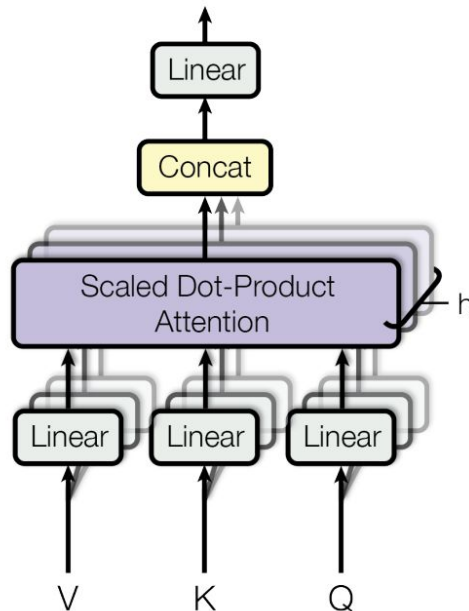


Multi-Head Attention

Scaled Dot-Product Attention



Multi-Head Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Multi-Head Attention: нормировка весов внимания

Зачем делить на корень
из размерности?

->

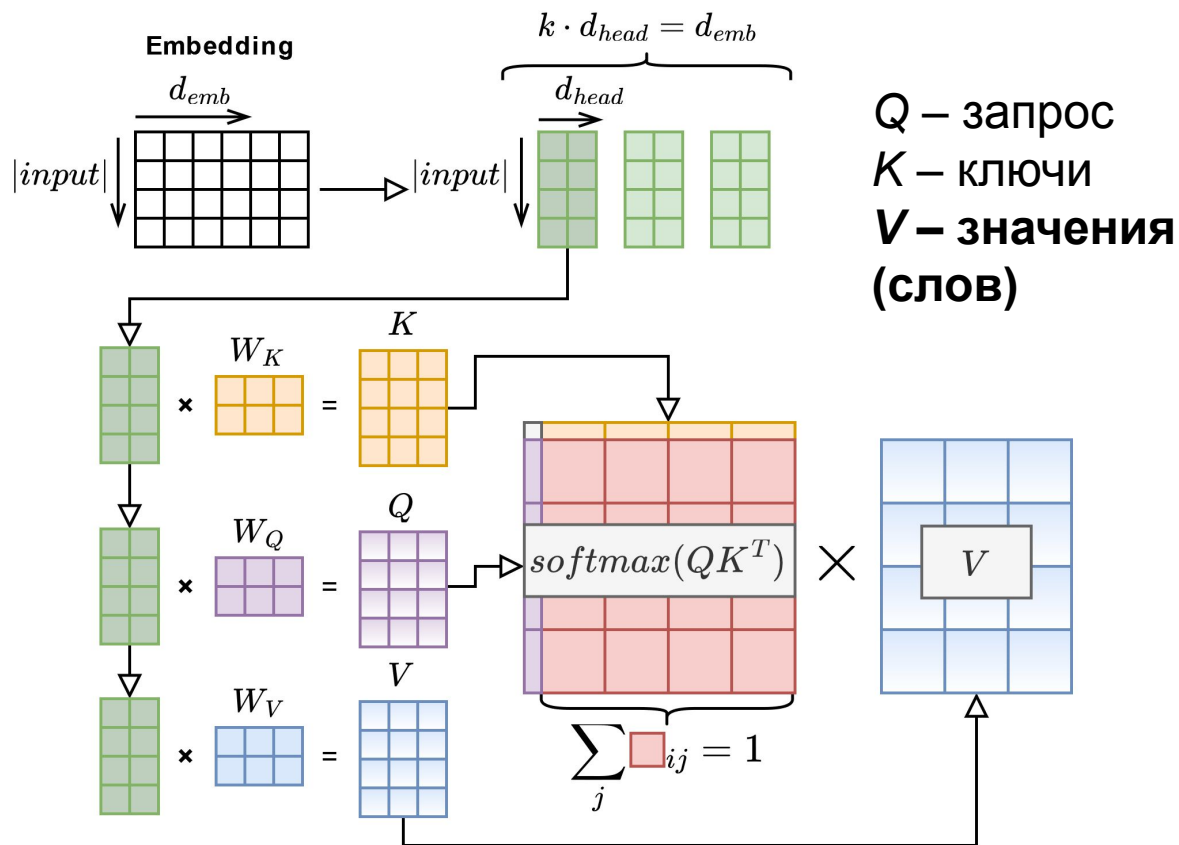
Чтобы веса внимания
имели единичное
стандартное отклонение.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\alpha = q_i k_j^T = \sum_{n=1}^{d_k} q_{in} k_{jn}$$

- 1) Предположим, что Q и K имеют 0 среднее и 1 стандартное отклонение
- 2) Дисперсия суммы = сумма дисперсий -> дисперсия $a = d$, а значит стандартное отклонение это корень из d.

Multi-Head Attention (tensors)



Multi-Head Attention (Pytorch code, LLaMa) [1]

Инициализация:

```
self.q_proj = nn.Linear(self.hidden_size, self.num_heads * self.head_dim, bias=config.attention_bias)
self.k_proj = nn.Linear(self.hidden_size, self.num_key_value_heads * self.head_dim, bias=config.attention_bias)
self.v_proj = nn.Linear(self.hidden_size, self.num_key_value_heads * self.head_dim, bias=config.attention_bias)
self.o_proj = nn.Linear(self.hidden_size, self.hidden_size, bias=config.attention_bias)
```

Расчет матриц Q, K, V:

```
query_states = self.q_proj(hidden_states)
key_states = self.k_proj(hidden_states)
value_states = self.v_proj(hidden_states)
```

“Нарезка” на головы:

```
query_states = query_states.view(bsz, q_len, self.num_heads, self.head_dim).transpose(1, 2)
key_states = key_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)
value_states = value_states.view(bsz, q_len, self.num_key_value_heads, self.head_dim).transpose(1, 2)
```

Multi-Head Attention (Pytorch code, LLaMa) [2]

Расчет весов
внимания и
умножение на V:

```
attn_weights = nn.functional.softmax(attn_weights, dim=-1, dtype=torch.float32).to(query_states.dtype)
attn_weights = nn.functional.dropout(attn_weights, p=self.attention_dropout, training=self.training)
attn_output = torch.matmul(attn_weights, value_states)
```

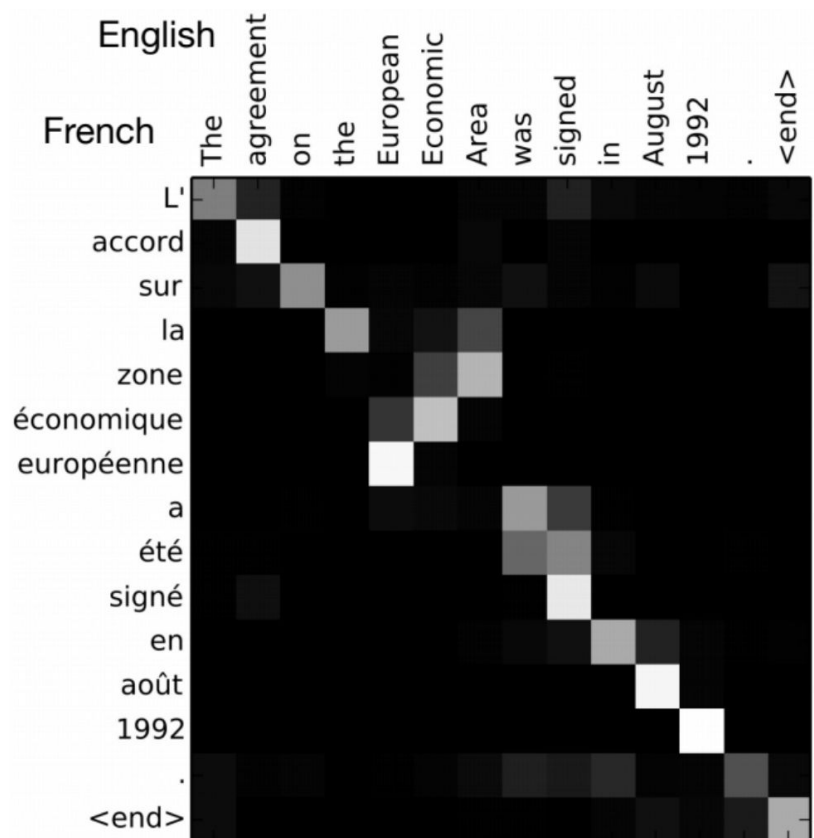
“Конкатенация”
голов:

```
attn_output = attn_output.transpose(1, 2).contiguous()
attn_output = attn_output.reshape(bsz, q_len, self.hidden_size)
```

И еще один
линейный слой на
выходе:

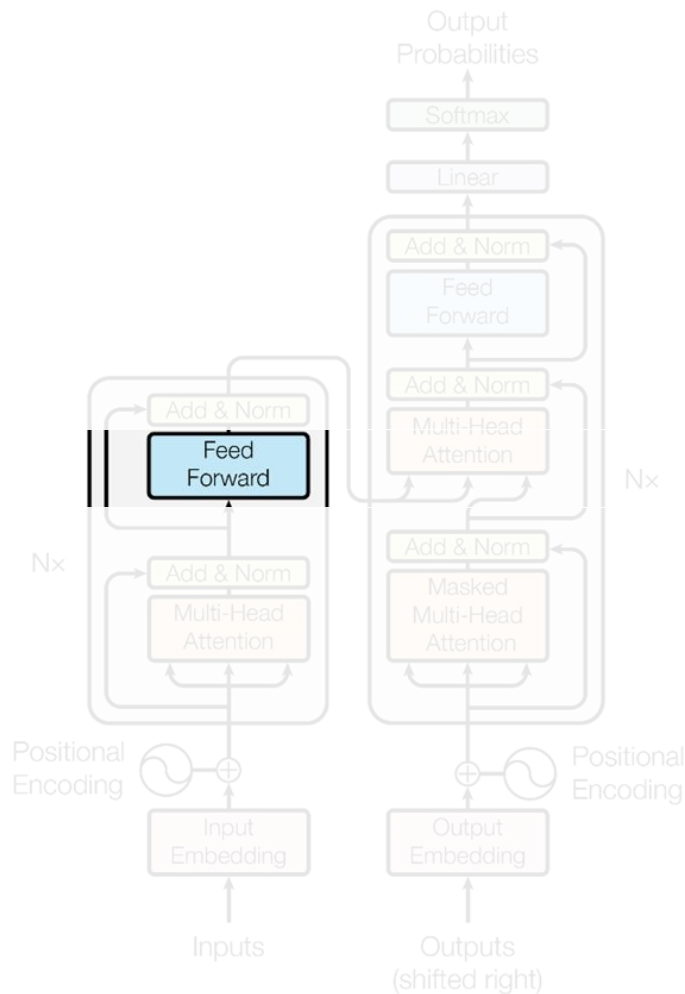
```
attn_output = self.o_proj(attn_output)
```

Визуализация Self Attention



Transformer (2017)

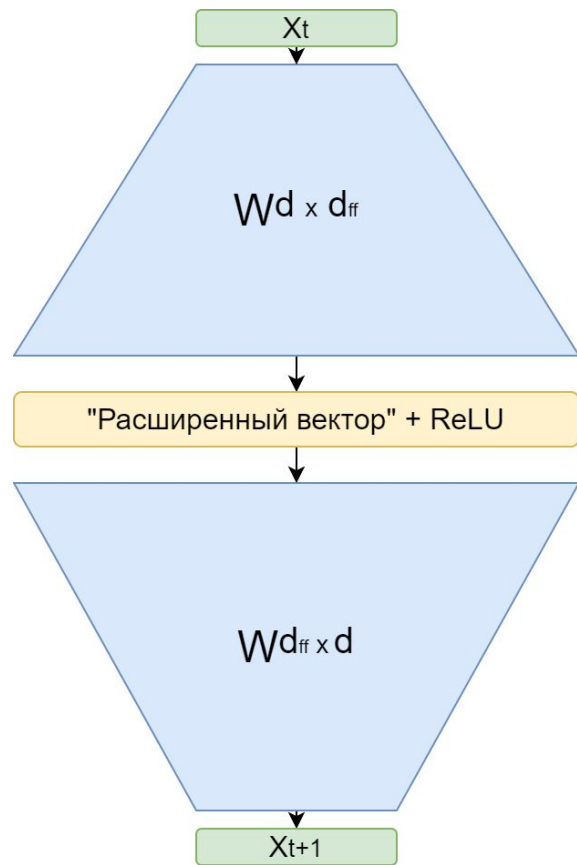
- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



Блок Feed Forward (MLP, FCNN)

- Внутренняя размерность обычно сильно больше hidden_size, например, x4.
- Интуиция: FF являются хранилищем “знаний” моделей (так ли это, не известно).

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



Блок Feed Forward (Pytorch code, LLaMa)

Инициализация:

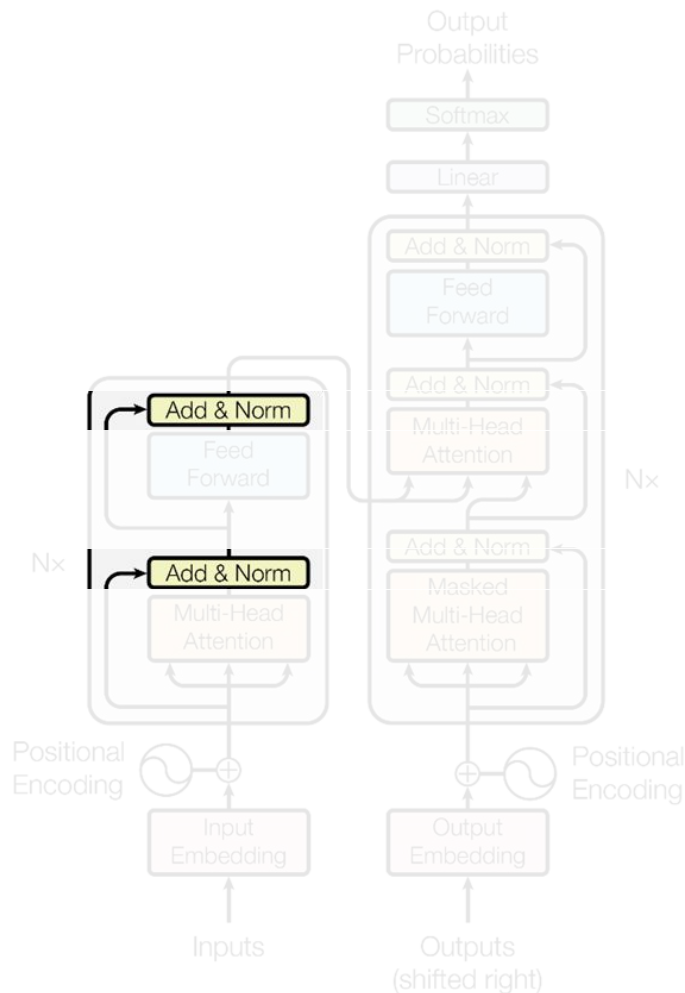
```
self.gate_proj = nn.Linear(self.hidden_size, self.intermediate_size, bias=False)
self.up_proj = nn.Linear(self.hidden_size, self.intermediate_size, bias=False)
self.down_proj = nn.Linear(self.intermediate_size, self.hidden_size, bias=False)
```

Forward:

```
down_proj = self.down_proj(self.act_fn(self.gate_proj(x)) * self.up_proj(x))
```

Transformer (2017)

- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



Layer Norm

- Layer Norm - это метод **нормализации** активации слоя нейронной сети.
- В оригинальной архитектуре “add and norm” шли после блока внимания и после блока FF: **Post-LN**.
- В дальнейшем слой нормализации был перенесен перед и после слоя внимания: **Pre-LN**.

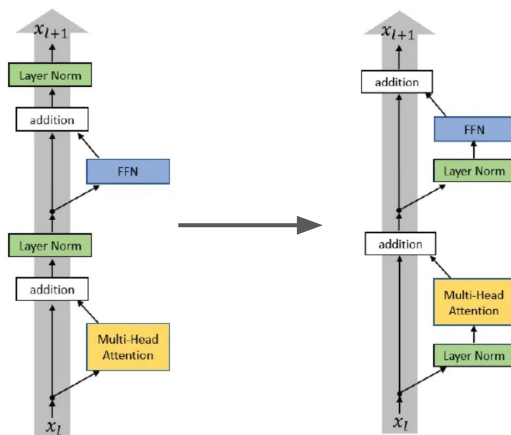
$$\text{LayerNorm}(v) = \gamma \frac{v - \mu}{\sigma} + \beta$$

Среднее:

$$\mu = \frac{1}{d} \sum_{k=1}^d v_k$$

Стандартное отклонение:

$$\sigma^2 = \frac{1}{d} \sum_{k=1}^d (v_k - \mu)^2$$



Layer Norm (Pytorch code, LLaMa)

```
residual = hidden_states
```

LN на входе блока

```
hidden_states = self.input_layernorm(hidden_states)
```

```
# Self Attention
```

Блок внимания

```
hidden_states, self_attn_weights, present_key_value = self.self_attn(  
    hidden_states=hidden_states,  
    attention_mask=attention_mask,  
    position_ids=position_ids,  
    past_key_value=past_key_value,  
    output_attentions=output_attentions,  
    use_cache=use_cache,  
    cache_position=cache_position,  
    **kwargs,  
)
```

```
hidden_states = residual + hidden_states
```

```
# Fully Connected
```

```
residual = hidden_states
```

LN на выходе блока

```
hidden_states = self.post_attention_layernorm(hidden_states)
```

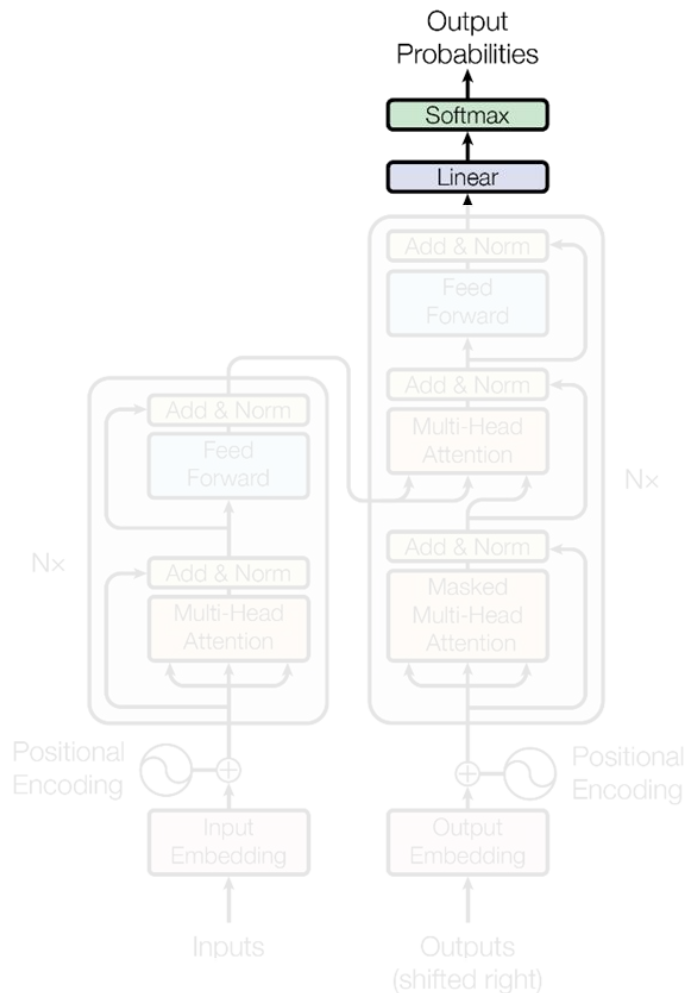
MLP (Feed Forward)

```
hidden_states = self.mlp(hidden_states)
```

```
hidden_states = residual + hidden_states
```

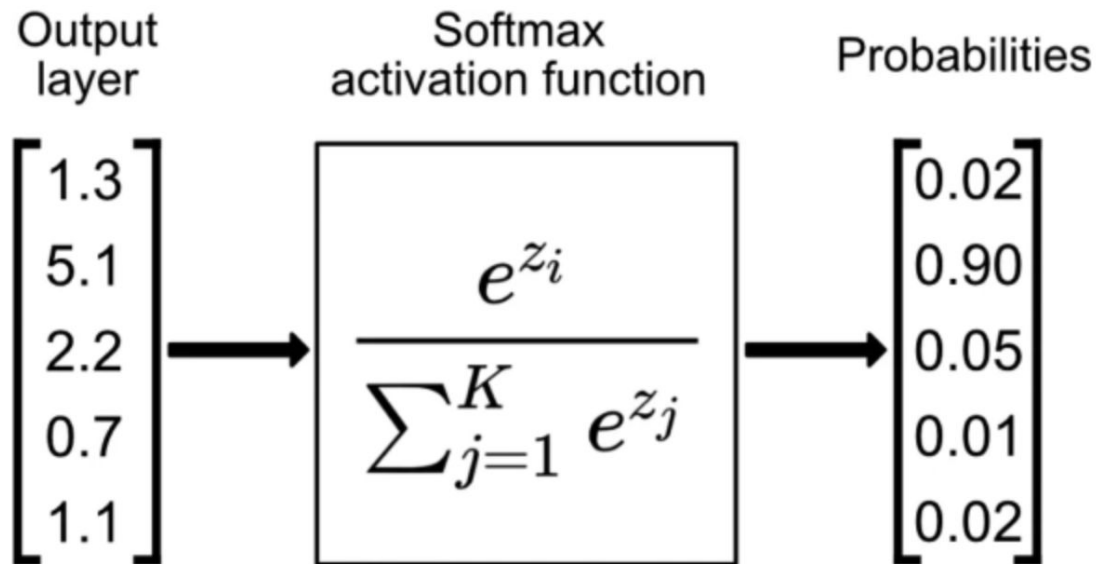
Transformer (2017)

- Токенизация слов
- Позиционное кодирование
- Преобразование векторов через трансформер-блоки
 - Multi-head attention (MHA)
 - Feed-forward (FFN)
 - LayerNorm
- Итоговое предсказание слова линейным слоем (lm head)



Im head

- Линейный слой отображающий вектор, получаемый из трансформера в вектор “логитов”, размерностью в $|V|$.
- Вектор логитов затем преобразуется через softmax для получения “вероятностей” токенов.



Генерация с помощью LLM

- В идеале хотелось бы уметь с помощью LLM генерировать текст, который максимизирует вероятность

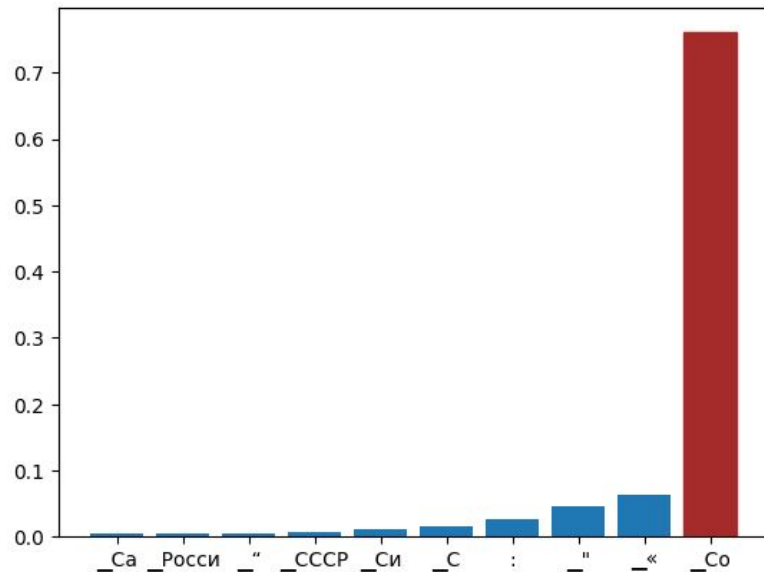
$$y' = \arg \max_y p(y|x) = \arg \max_y \prod_{t=1}^n p(y_t | y_{<t}, x)$$

- Перебрать все существующие цепочки - невозможно.
- Выход: Генерация token за tokenом на основе текущих вероятностей - Sampling:
 - greedy ,
 - top-k,
 - top-p,
 - beam_search

Greedy sampling

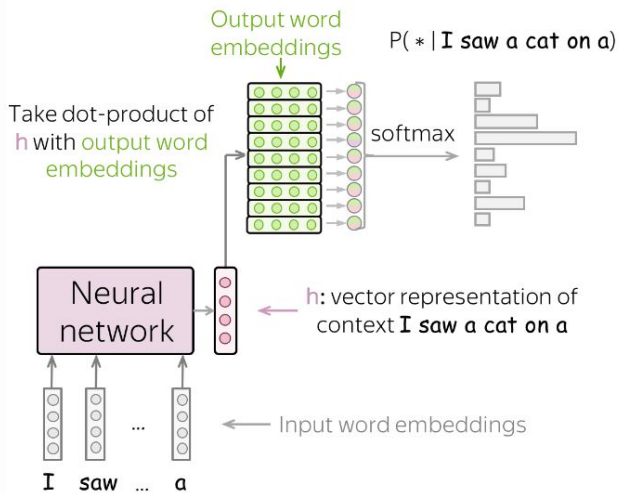
СССР расшифровывается как ?

- Каждый раз берем токен с максимальной вероятностью.
- Минусы: полное отсутствие разнообразия.
- Альтернатива: sampling в зависимости от вероятности самого токена.

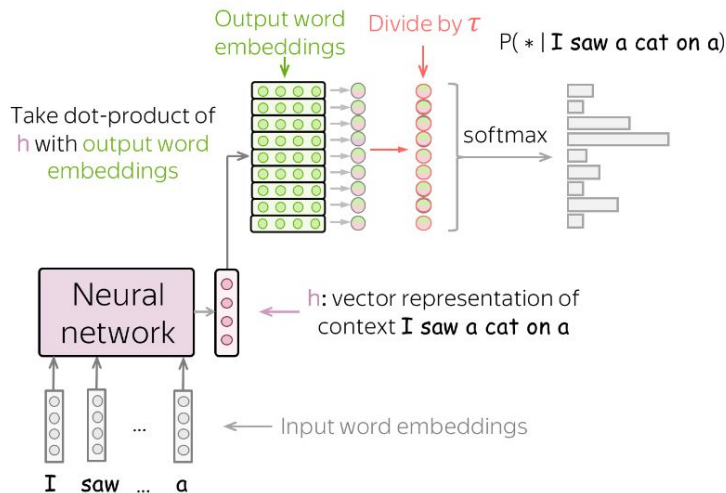


Temperature

Before



After

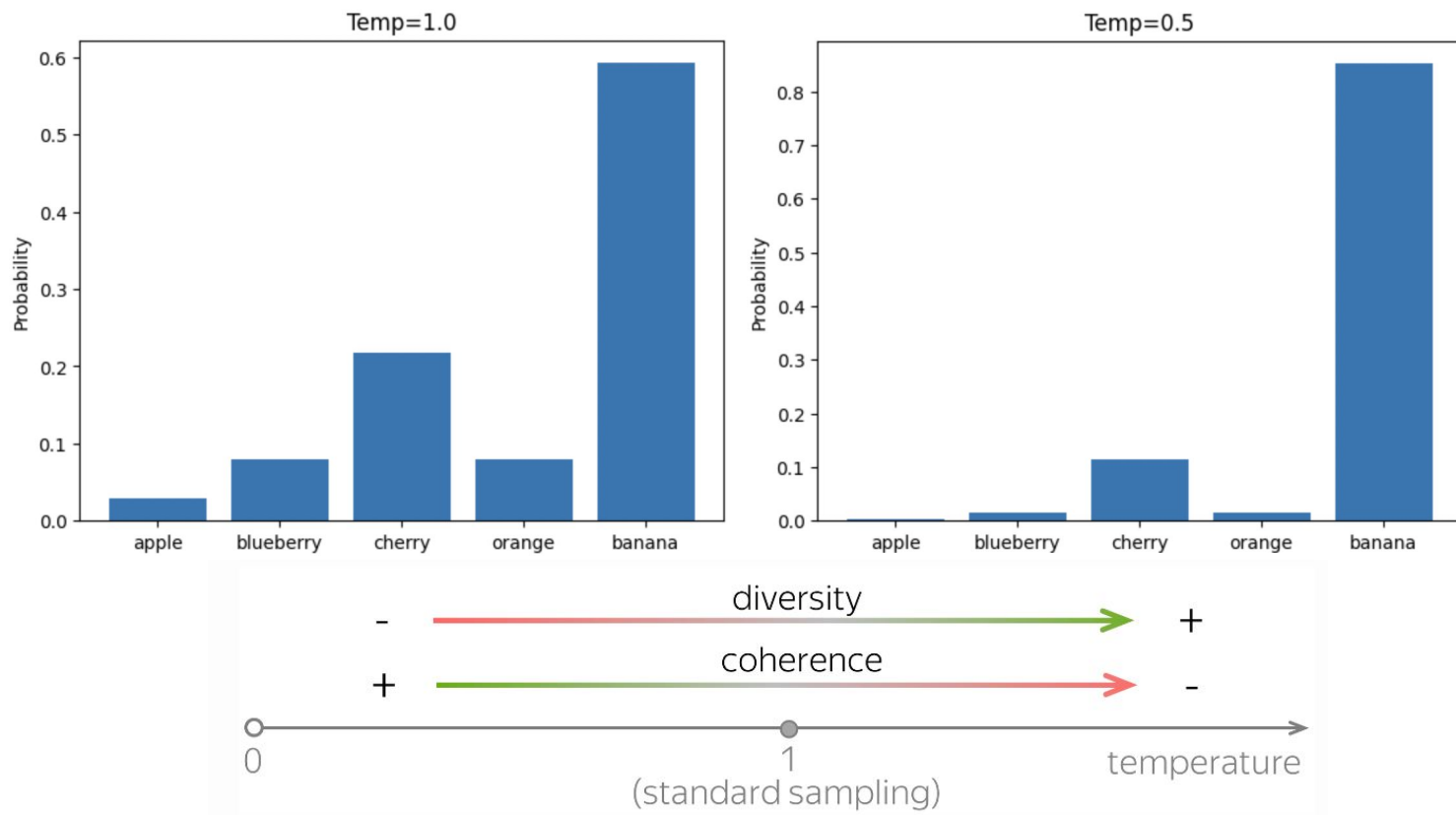


Formally, the computations change as follows:

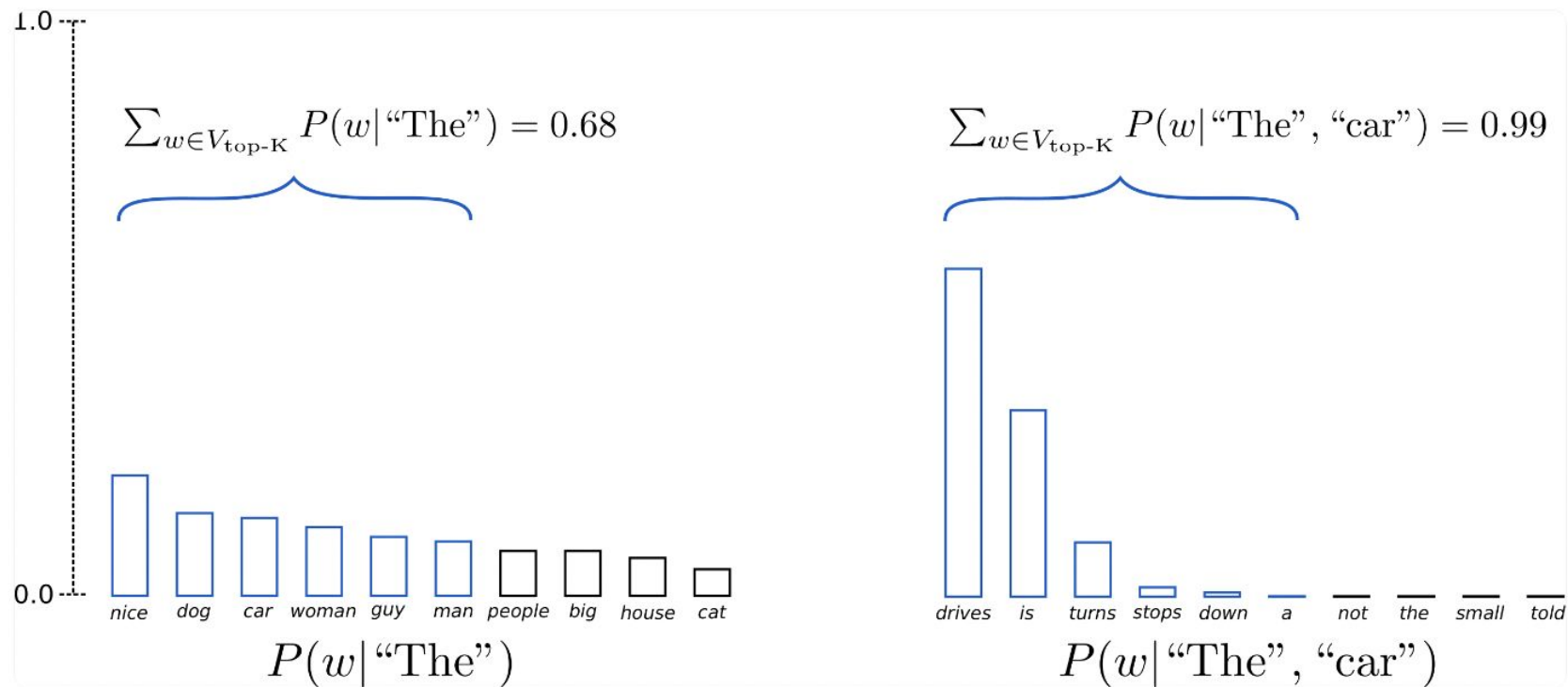
$$\frac{\exp(h^T w)}{\sum_{w_i \in V} \exp(h^T w_i)} \rightarrow \frac{\exp\left(\frac{h^T w}{\tau}\right)}{\sum_{w_i \in V} \exp\left(\frac{h^T w_i}{\tau}\right)}$$

τ - softmax temperature

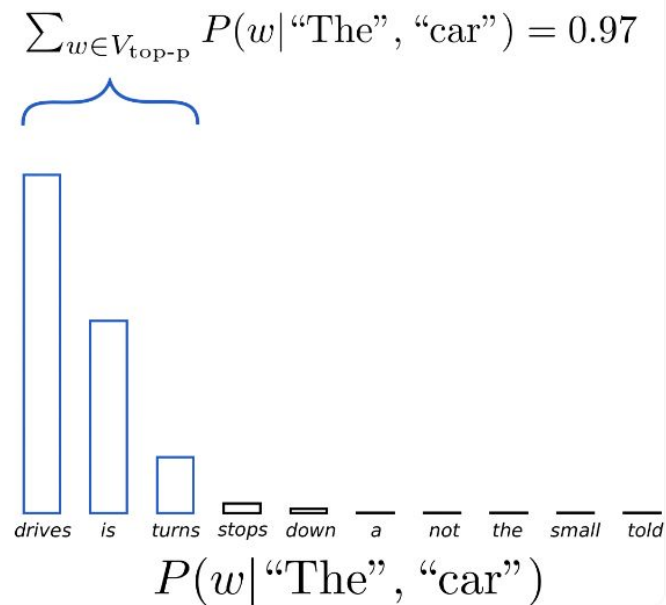
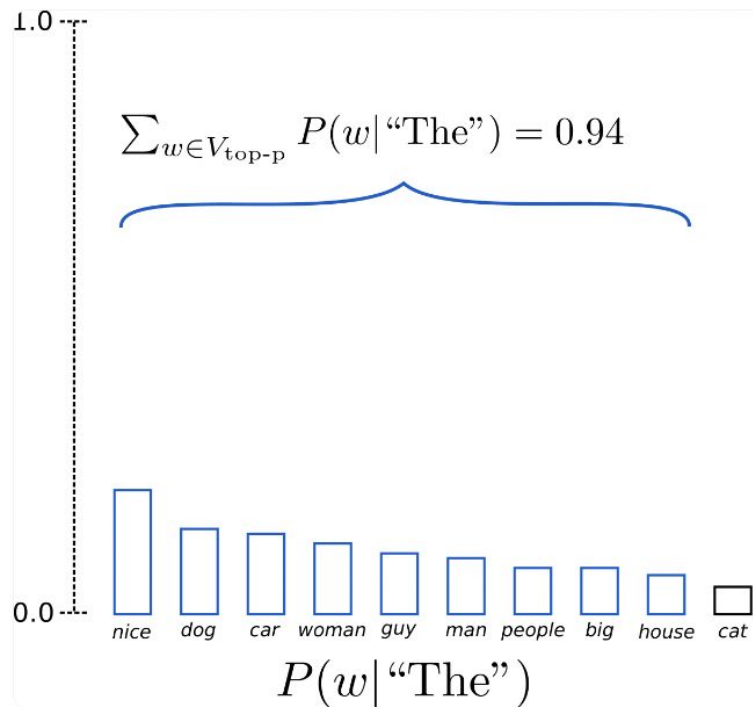
Temperature



Top-K

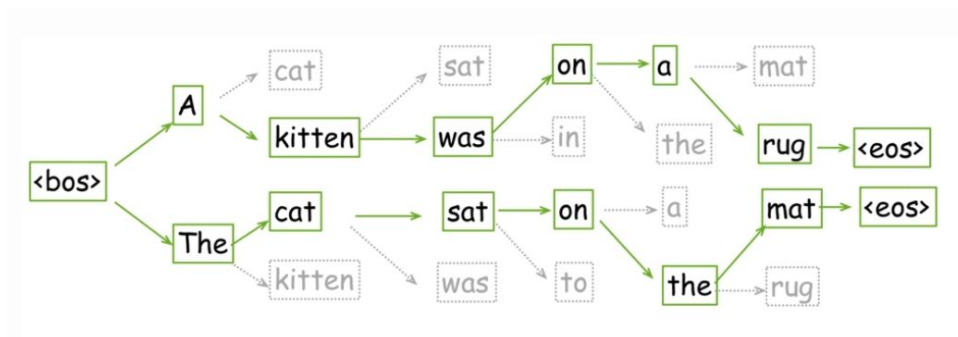


Top-P (nucleus sampling)



Beam search

- Отслеживание нескольких наиболее вероятных гипотез, вместо одной.
- Вес цепочки - сумма logprobs нормированная на длину.



- Раньше активно применялся, сейчас редко из-за накладных расходов (необходимо вести beam_size генераций) и низкого разнообразия ответов.

Архитектура Transformer: начало современного NLP

Transformer: первое впечатление

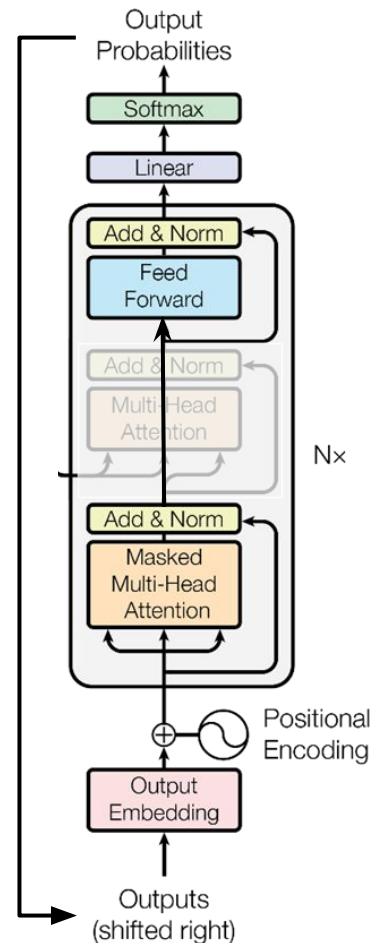
| Model | BLEU | | Training Cost (FLOPs) | |
|---------------------------------|-------------|--------------|---------------------------------------|---------------------|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | 41.29 | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $3.3 \cdot 10^{18}$ | |
| Transformer (big) | 28.4 | 41.8 | $2.3 \cdot 10^{19}$ | |

- Тестирование на задаче перевода,
- Нет существенного “скачка” в качестве.

OpenAI GPT-1 (2018)

- 12 слоев **Transformer decoder** (~117 млн.),
- Обучение в 2 этапа:
 - Предобучение (pre-training) на задаче **моделирования языка**
$$\max_{\Theta} \sum_{0 \leq i \leq n} \log P(w_i | w_{i-1} \dots w_0; \Theta)$$

w - слова последовательности, Θ - параметры модели
 - Дообучение (fine-tuning) на целевые задачи
- Предобучался только на художественной литературе



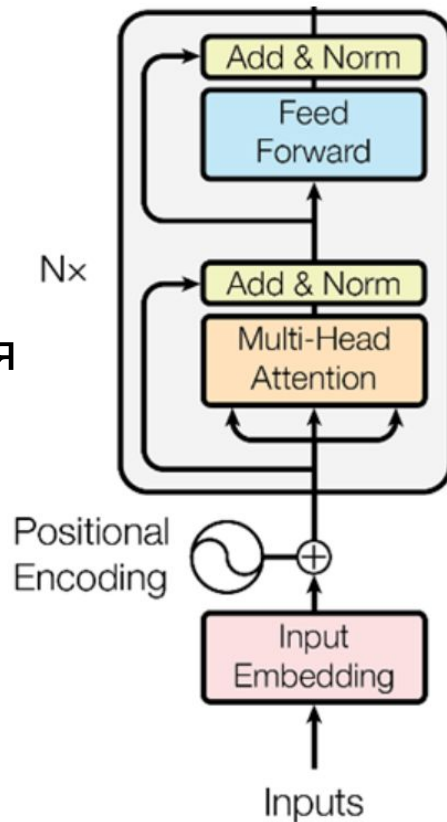
OpenAI GPT-1: оценка качества

| Method | MNLI-m | MNLI-mm | SNLI | SciTail | QNLI | RTE |
|-------------------------------------|-------------|-------------|-------------|-------------|-------------|-------------|
| ESIM + ELMo [44] (5x) | - | - | <u>89.3</u> | - | - | - |
| CAFE [58] (5x) | 80.2 | 79.0 | <u>89.3</u> | - | - | - |
| Stochastic Answer Network [35] (3x) | <u>80.6</u> | <u>80.1</u> | - | - | - | - |
| CAFE [58] | 78.7 | 77.9 | 88.5 | <u>83.3</u> | | |
| GenSen [64] | 71.4 | 71.3 | - | - | <u>82.3</u> | 59.2 |
| Multi-task BiLSTM + Attn [64] | 72.2 | 72.1 | - | - | 82.1 | 61.7 |
| Finetuned Transformer LM (ours) | 82.1 | 81.4 | 89.9 | 88.3 | 88.1 | 56.0 |

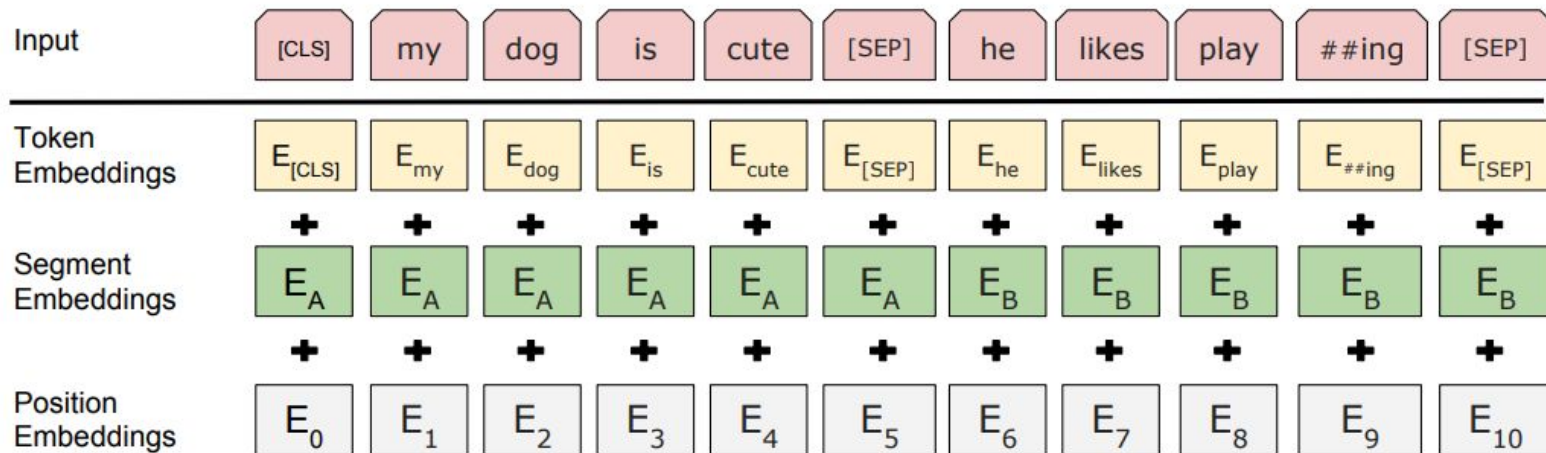
| Method | Story Cloze | RACE-m | RACE-h | RACE |
|---------------------------------|-------------|-------------|-------------|-------------|
| val-LS-skip [55] | 76.5 | - | - | - |
| Hidden Coherence Model [7] | <u>77.6</u> | - | - | - |
| Dynamic Fusion Net [67] (9x) | - | 55.6 | 49.4 | 51.2 |
| BiAttention MRU [59] (9x) | - | <u>60.2</u> | <u>50.3</u> | <u>53.3</u> |
| Finetuned Transformer LM (ours) | 86.5 | 62.9 | 57.4 | 59.0 |

BERT (2018)

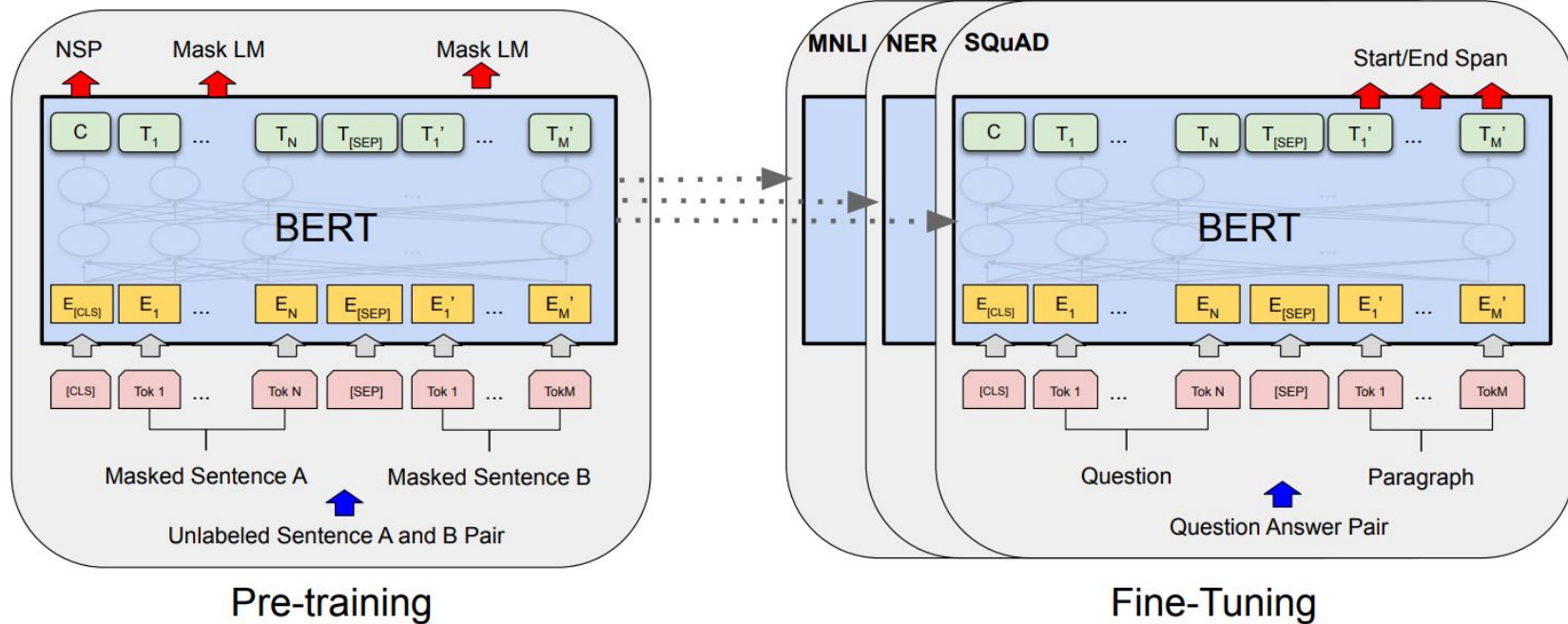
- Base: 12 слоев **Transformer encoder** (~110 млн. параметров),
- Обучение в 2 этапа!
 - **Предобучение (pre-training)** на задачах:
 - маскированного языкового моделирования (**MLM**)
 - предсказания следующего предложения (**NSP**).
 - **Дообучение (fine-tuning)** на целевые задачи
- **3.3 миллиарда** слов English Wikipedia + BooksCorpus.



BERT: формирование входа



BERT: pre-training и fine-tuning



BERT: результаты

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

Некоторые практические аспекты работы с LLM

“Типы” LLM

- **Foundational** (базовая) - LLM, которые обучались предсказывать следующее слово без специального дообучения инструкциям
 - На данном этапе приобретаются основные знания модели!
- **Instruct** (инструктивные) - это базовые LLM, которые прошли дополнительный этап обучения инструкциям
 - Дообучение происходит в специальном формате, который фиксируется как `chat_template`
 - Приобретаются навыки отвечать на вопросы пользователя так, чтобы ему понравилось (в частности, решать задачи в zero-shot)

Chat Template

- У разных моделей чат темплейт может быть разным! Всегда смотрите в карточках модели + в токенайзере.
- Формирование промптов из **последовательностей сообщений**
- 3 основных роли: system, user, assistant (названия могут варьироваться)

```
<|im_start|>system
```

```
You are Qwen, created by Alibaba Cloud. You are a helpful assistant.<|im_end|>
```

```
<|im_start|>user
```

```
Что такое LLM?<|im_end|>
```

```
<|im_start|>assistant
```

HuggingFace

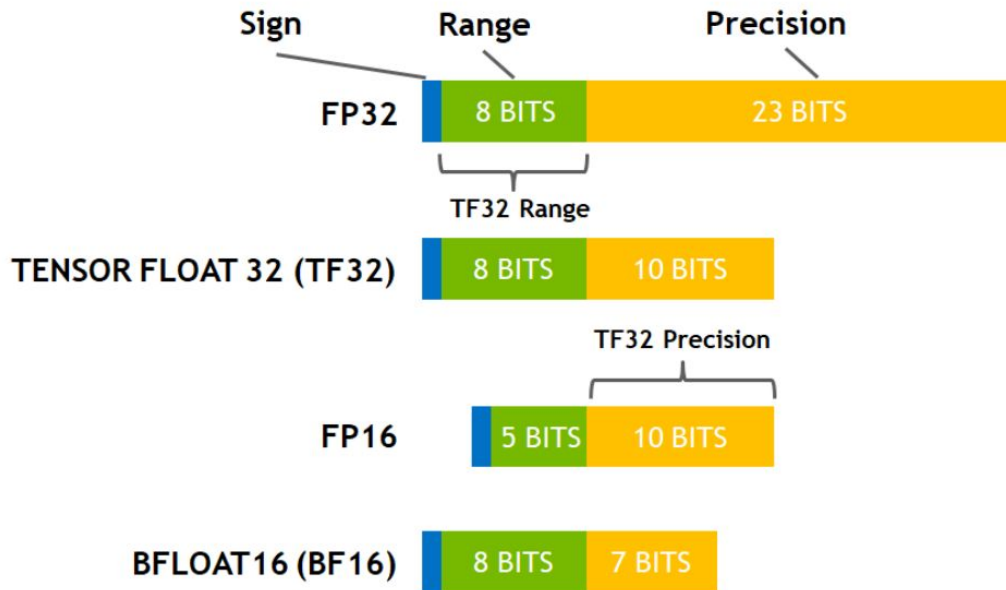
- Огромное сообщество профессионалов
- Платформа для хранения моделей, датасетов и тп
- Все опенсорс релизы LLM можно найти там

Также:

- transformers - одна из основных библиотек для работы с LLM
- datasets - библиотека для работы с датасетами из HF
- peft - библиотека для эффективного дообучения LLM
- ...

Различные типы чисел

- В основном LLM обучают в fp16/bf16
- Некоторые операции / слои все еще могут быть в fp32 (mixed-precision)



Проблемы классического attention

- Квадратичная сложность от длины последовательности как по времени, так и по памяти.
- Как результат, проблемы с длинными последовательностями.

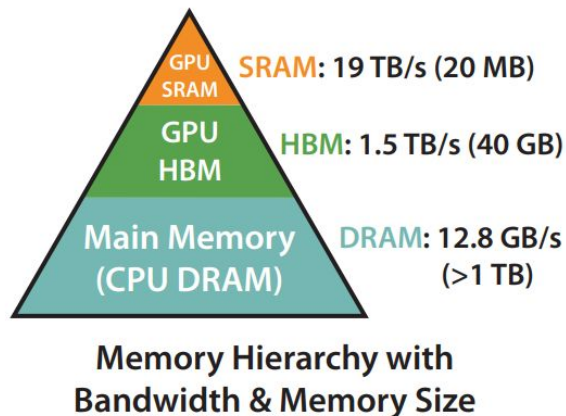
$$X \in \mathbb{R}^{n \times d}$$

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V$$

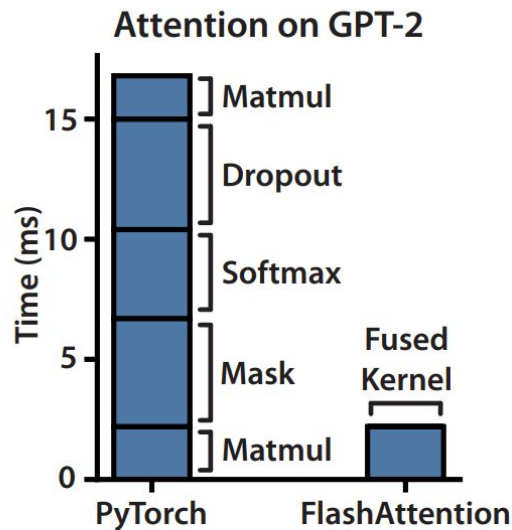
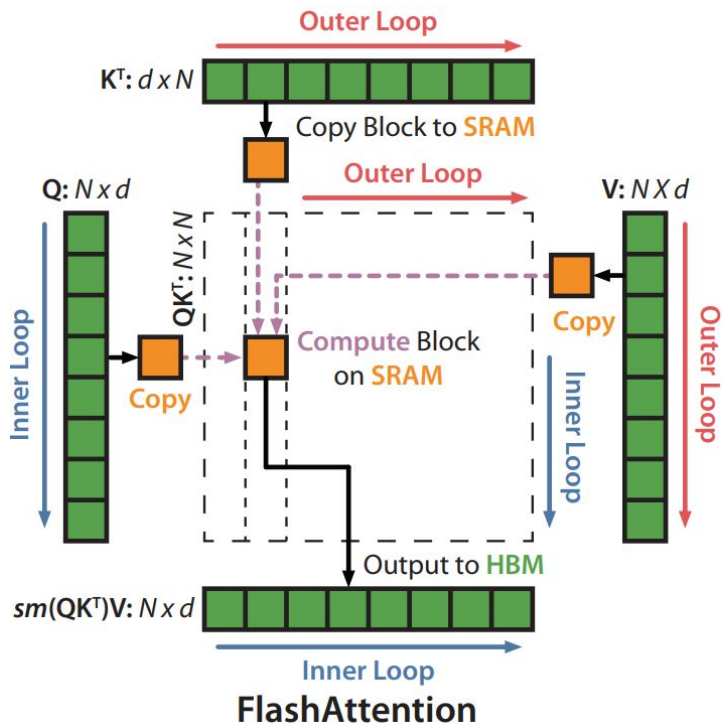
$$SelfAttention(X) = Attention(XW_Q, XW_K, XW_V)$$

flash-attention: идея

- GPU имеет разную память.
- Чтения и записи между ними, а также использование более медленной памяти снижают скорость работы вычислений.



flash-attention: метод



“FlashAttention does not read and write the large $N \times N$ attention matrix to HBM”

flash-attention: результаты

| Models | ListOps | Text | Retrieval | Image | Pathfinder | Avg | Speedup |
|-----------------------------|---------|------|-----------|-------|------------|------|-------------|
| Transformer | 36.0 | 63.6 | 81.6 | 42.3 | 72.7 | 59.3 | - |
| FLASHATTENTION | 37.6 | 63.9 | 81.4 | 43.5 | 72.7 | 59.8 | 2.4× |
| Block-sparse FLASHATTENTION | 37.0 | 63.0 | 81.3 | 43.6 | 73.3 | 59.6 | 2.8× |
| Linformer [84] | 35.6 | 55.9 | 77.7 | 37.8 | 67.6 | 54.9 | 2.5× |
| Linear Attention [50] | 38.8 | 63.2 | 80.7 | 42.6 | 72.5 | 59.6 | 2.3× |
| Performer [12] | 36.8 | 63.6 | 82.2 | 42.1 | 69.9 | 58.9 | 1.8× |
| Local Attention [80] | 36.1 | 60.2 | 76.7 | 40.6 | 66.6 | 56.0 | 1.7× |
| Reformer [51] | 36.5 | 63.8 | 78.5 | 39.6 | 69.4 | 57.6 | 1.3× |
| Smyrf [19] | 36.1 | 64.1 | 79.0 | 39.6 | 70.5 | 57.9 | 1.7× |

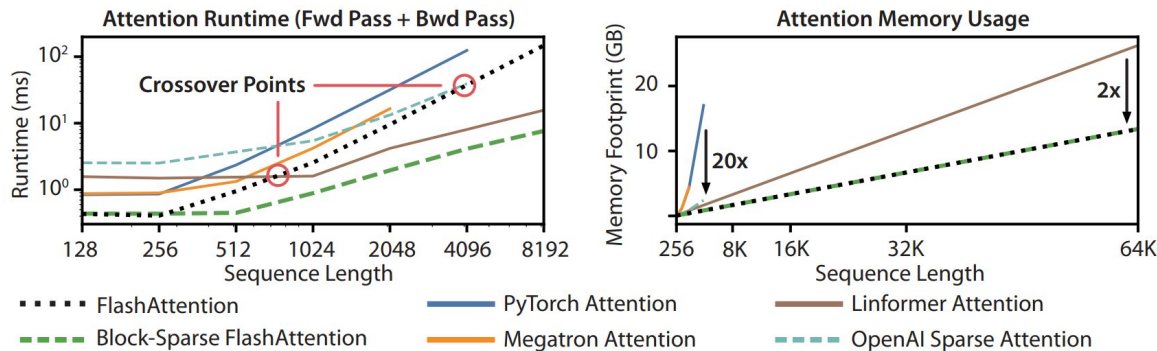
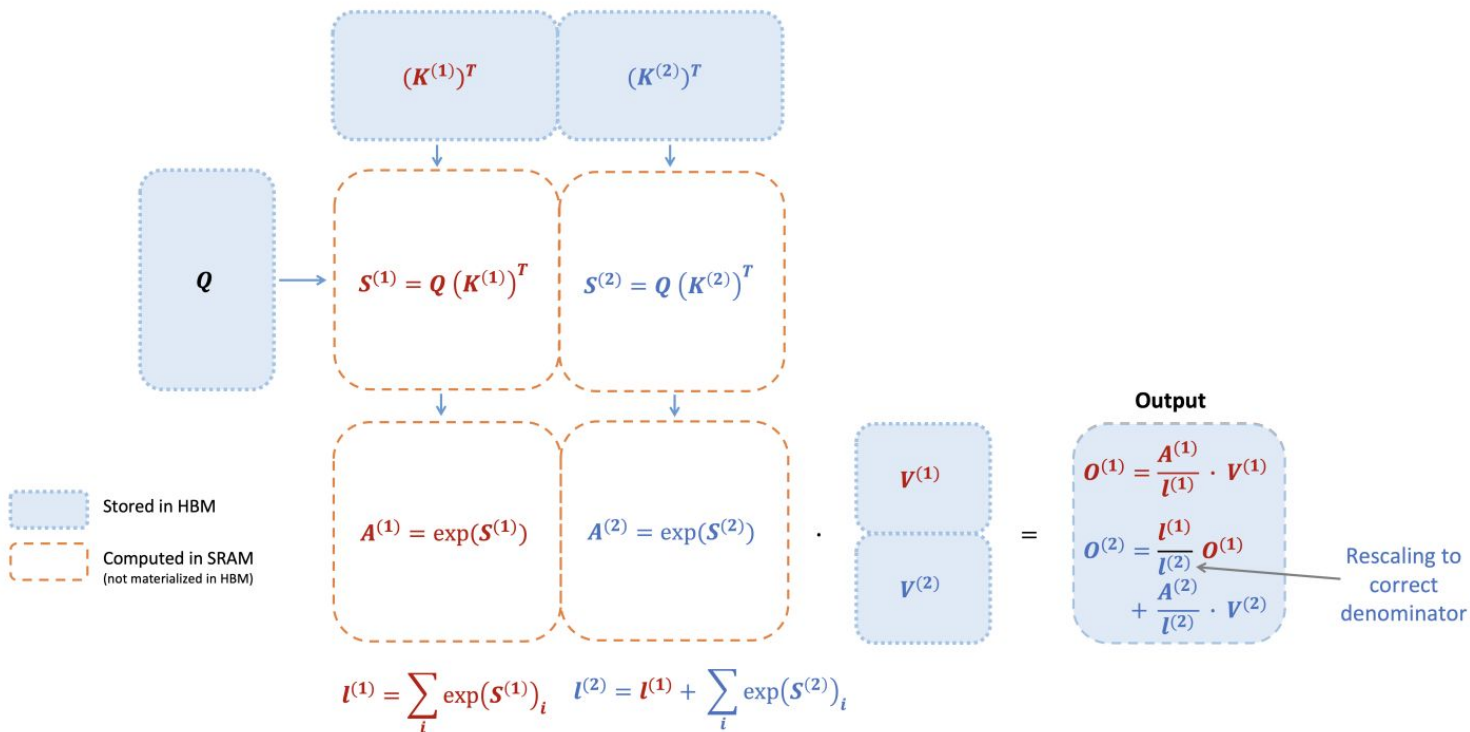


Figure 3: **Left:** runtime of forward pass + backward pass. **Right:** attention memory usage.

flash-attention-2

- Несмотря на то, что FlashAttention в 2-4 раза быстрее стандартной реализации, все равно forward pass достигает только 30-50% от теоретического максимума FLOPs/s на современных GPU.
 - Backward pass еще хуже, 25-35%
- Современные GPU имеют специализированные встроенные средства для перемножения матриц, благодаря которым пропускная способность matmul операции может быть в 16 раз выше, чем для non-matmul.

flash-attention-2: метод



flash-attention-2: результаты A100

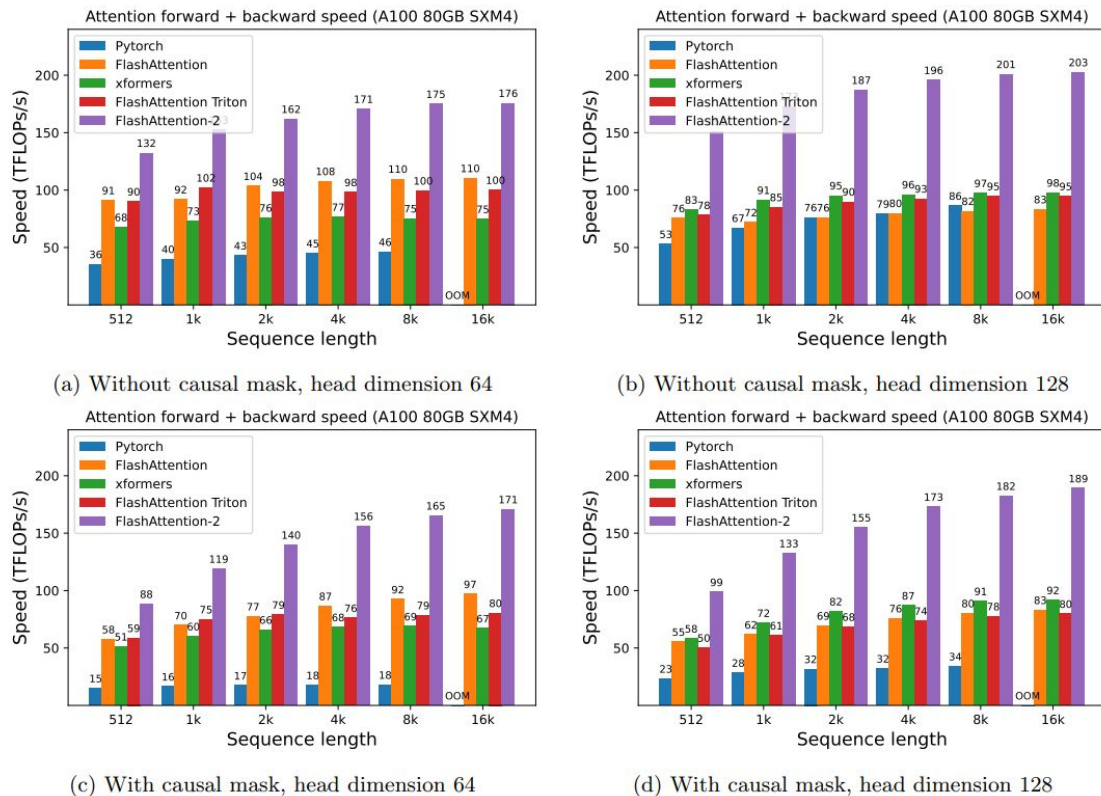
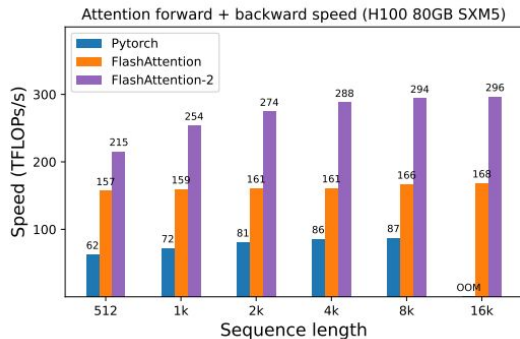
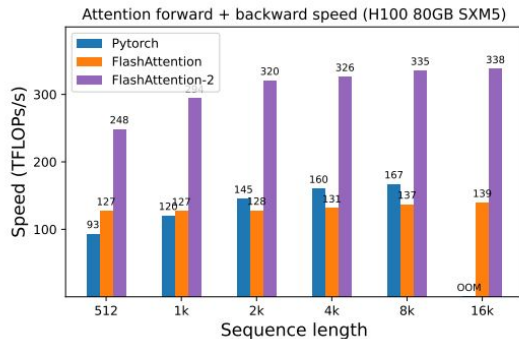


Figure 4: Attention forward + backward speed on A100 GPU

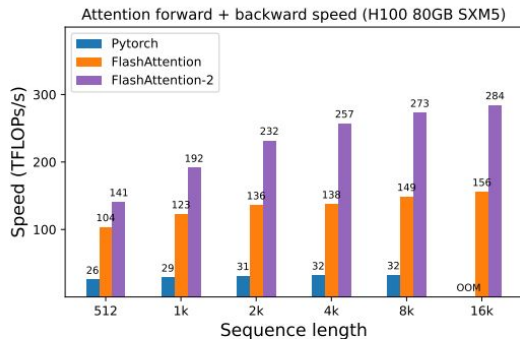
flash-attention-2: результаты H100



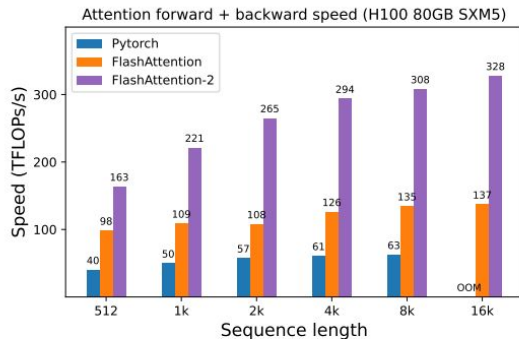
(a) Without causal mask, head dimension 64



(b) Without causal mask, head dimension 128



(c) With causal mask, head dimension 64



(d) With causal mask, head dimension 128

Figure 7: Attention forward + backward speed on H100 GPU

flash-attention 1-2: выводы

- Существенное ускорение расчета attention, как во время обучения, так и во время инференса.
- Нет потери в качестве, так как по сути расчет attention не поменялся.
- Стала линейная зависимость по памяти.
- Применяется как есть на существующие уже обученные модели.
- Для FlashAttention-2 требуются современные ускорители.

FlashAttention-2 currently supports:

1. Ampere, Ada, or Hopper GPUs (e.g., A100, RTX 3090, RTX 4090, H100). Support for Turing GPUs (T4, RTX 2080) is coming soon, please use FlashAttention 1.x for Turing GPUs for now.
2. Datatype fp16 and bf16 (bf16 requires Ampere, Ada, or Hopper GPUs).
3. All head dimensions up to 256. Head dim > 192 backward requires A100/A800 or H100/H800.

Квантизация

- Квантизация – еще один способ “уменьшить” модель, в частности для “инфера”,
- Суть в **преобразовании весов модели** из типа float32, float16 в **int8**, а иногда и в **int4**,
- Преобразование делается не просто “напрямую”, а более хитрыми способами. Существуют реализации, совместимые с huggingface transformers,
 - **load_in_8bit, load_in_4bit** флаги (не лучшая квантизация),
- Закономерный результат квантизации – **падение качества**,
- Можно совмещать с обычным **LoRa**,
 - Замороженную модель в 8bit/4bit, но обучаемые веса в float16,
- **QLoRa** - более эффективный способ обучения модели с квантизацией.

Домашнее задание 1

- Зайти на google colab / kaggle (или на своем железе, если есть)
- Создать нотбук с загрузкой и работой с LLM по аналогии с показанным в лекции
- Придумать 10 не самых простых вопросов к LLM и сравнить 2 модели на выбор:
 - Например, qwen-2.5-7B-Instruct и Phi-4-mini-instruct
- Сравнить вручную: лучше model1, лучше model2, одинаковы.
- Прислать нотбук + отчет (pdf)

Задание: оценивание и сроки

- Срок 1 неделя: до 9 марта 23:59.
- Присылать на tikhomirov.mm@gmail.com
 - Название письма: Основы LLM: Задание 1, запуск LLM.
 - В письме Ваше полное ФИО, группа, решение и краткий отчет по нему в PDF.
- Оценка по шкале “-/-+/-+/-++”.
 - ++ за те решения, которые особо мне понравятся чем-либо.