

Трансформеры

Санкт-Петербургский государственный университет
Кафедра статистического моделирования
Семинар «Статистическое и машинное обучение»

Санкт-Петербург, 2025

Проблемы языковых моделей до появления внимания

Модель в идеальном мире

- **Effective** — хорошо справляется с задачей, для которой эта модель используется
- **Efficient** — оптимально потребляет ресурсы с точки зрения памяти, CPU и т.д.

Ограничения RNN-архитектур

- **He Effective** — потеря контекста
 - Вся информация о последовательности содержится в скрытом состоянии, которое обновляется на каждом шаге
 - Длинные зависимости «забываются» из-за механизма обновления состояния
- **He Efficient** — невозможность распараллеливания
 - Вычисления на шаге t зависят от результатов шага $t - 1$

Вывод: Требуется механизм прямого доступа к любой части последовательности

Решение: Механизм *Self-Attention* в архитектуре Transformer

Основные компоненты

- **Keys (K)** — определяет релевантность токена для различных запросов

$$k_m = \beta_k + W_k x_m$$

- **Queries (Q)** — выражает информационные потребности токена в контексте

$$q_n = \beta_q + W_q x_n$$

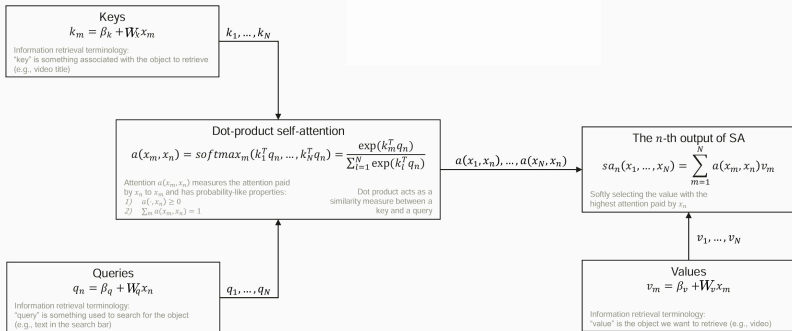
- **Values (V)** — информация, которую мы хотим извлечь

$$v_m = \beta_v + W_v x_m$$

- N — длина контекста (количество входных токенов)
- D — размерность векторного представления токена
- D_q — размерность ключей/запросов/значений
- $x_m \in \mathbb{R}^D$ — векторное представление m -го токена
- $X \in \mathbb{R}^{D \times N}$ — матрица эмбедингов
- $\beta^* \in \mathbb{R}^{D_q}$, $W^* \in \mathbb{R}^{D_q \times D_q}$ — веса линейного слоя для ключей/запросов/значений, общие для всех векторных представлений токенов
- $K, V, Q \in \mathbb{R}^{D_q \times N}$ — матрицы ключей, значений и запросов

Dot-product self-attention

- 1 Каждый токен x_m представляется в виде ключа k_m , значения v_m и запроса q_m
- 2 Для каждой пары (n, m) вычисляется скалярное произведение $k_m^T q_n$
- 3 Применяется softmax для получения вероятностного распределения внимания
- 4 Вычисляется взвешенная сумма значений с весами внимания, тем самым для каждого токена получаем его контекстуализированное представление



Scaled dot-product self-attention

Проблемы:

- Большие значения скалярных произведений приводят к малым градиентам
- Softmax становится нечувствительным к небольшим изменениям входов

Решение — масштабирование:

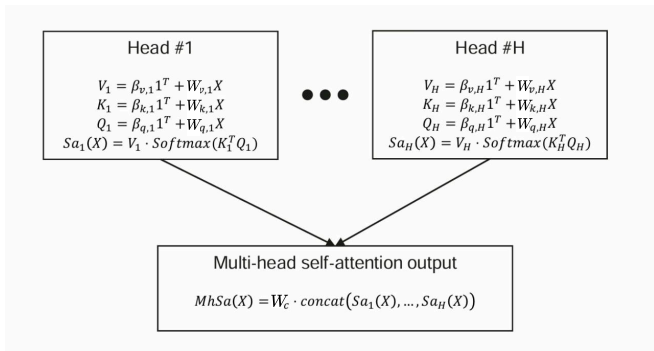
$$Sa(X) = V \cdot \text{Softmax} \left(\frac{K^T Q}{\sqrt{D_q}} \right)$$

При условии, что $k_i, q_i \sim \mathcal{N}(0, 1)$ и независимы

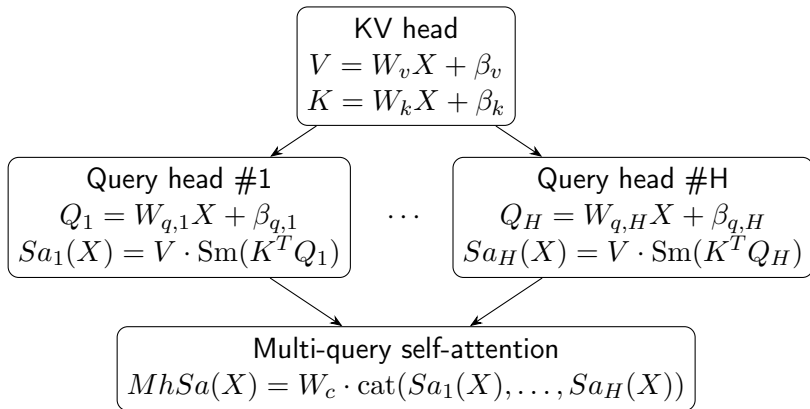
$$\text{Var}[k^T q] = \text{Var} \left[\sum_{i=1}^{D_q} k_i q_i \right] = D_q \implies \text{Var} \left[\frac{k^T q}{\sqrt{D_q}} \right] = 1.$$

Multi-head attention

- Вместо одного слоя внимания применяется несколько параллельных «голов» внимания с разными весами и размерностью D/H
- Каждая голова изучает разные типы зависимостей

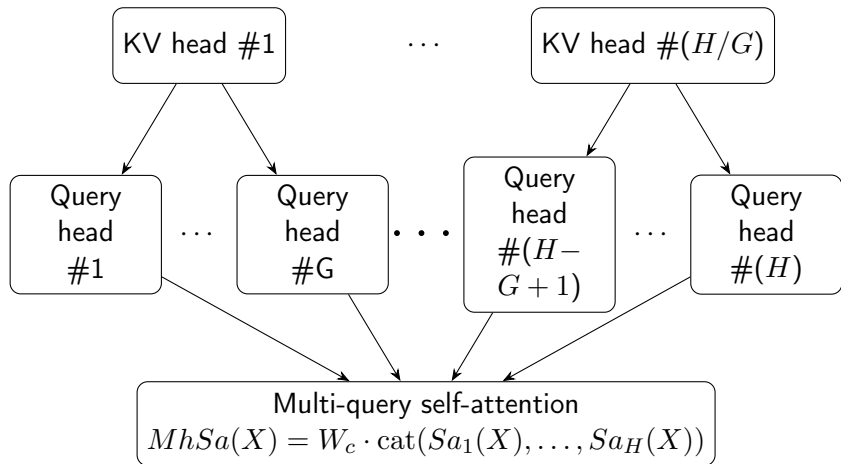


Multi-Query Attention



Sm = Softmax

Grouped-Query Attention



Позиционное Кодирование (Positional encoding)

Применение self-attention теряет информацию о порядке токенов в исходной последовательности

Идея: добавить информацию о порядке на embedding-этапе

- 1 Абсолютная позиция: $X \mapsto X + P$, матрица P может быть фиксирована:

$$P_{i,2j} = \sin\left(\frac{i}{10'000^{2j/D}}\right),$$
$$P_{i,2j+1} = \cos\left(\frac{i}{10'000^{2j/D}}\right),$$

а может быть и обучаемая

- 2 Относительная позиция: на самом деле нас интересует не абсолютное расположение токенов, а относительное друг к другу.

Есть много вариантов...

- Rotary Position Embedding (RoPE)

$$k_m^\top q_n \mapsto \langle k_m, q_n \rangle = g(x_m, x_n, n - m),$$

$$g(x_m, x_n, n - m) = x_n^\top W_q \mathbf{R}_{\Theta, n-m}^d W_k x_m,$$

$$\Theta = (\theta_i)_{i=1}^{D/2}, \quad \theta_i = 10'000^{-2(i-1)/D}$$

$$\mathbf{R}_{\Theta, m}^D = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{D/2} & -\sin m\theta_{D/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{D/2} & \cos m\theta_{D/2} \end{pmatrix}$$

Модификации: NTK-aware RoPE, YaRN...

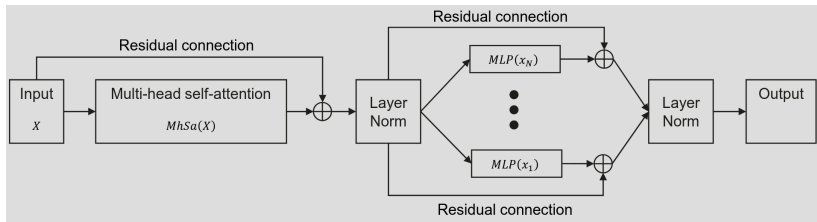
- Attention with Linear Biases (ALiBi)

$$\begin{array}{ccccc}
 q_1 \cdot k_1 & & & & \\
 q_2 \cdot k_1 & q_2 \cdot k_2 & & & \\
 q_3 \cdot k_1 & q_3 \cdot k_2 & q_3 \cdot k_3 & & \\
 q_4 \cdot k_1 & q_4 \cdot k_2 & q_4 \cdot k_3 & q_4 \cdot k_4 & \\
 q_5 \cdot k_1 & q_5 \cdot k_2 & q_5 \cdot k_3 & q_5 \cdot k_4 & q_5 \cdot k_5
 \end{array}
 +
 \begin{array}{ccccc}
 0 & & & & \\
 -1 & 0 & & & \\
 -2 & -1 & 0 & & \\
 -3 & -2 & -1 & 0 & \\
 -4 & -3 & -2 & -1 & 0
 \end{array}
 \cdot m$$

m — нетренируемый гиперпараметр, свой для каждой головы в MHA блоке

Трансформерный слой

Схема одного слоя трансформера (в изначальном виде):



В матричной форме:

$$X_{\text{in}} \mapsto X_{\text{Sa}} = X_{\text{in}} + \text{MhSa}(X_{\text{in}})$$

$$X_{\text{Sa}} \mapsto X_{\text{LN}} = \text{LayerNorm}(X_{\text{Sa}})$$

$$x_i^{\text{LN}} \mapsto x_i^{\text{MLP}} = x_i^{\text{LN}} + \text{MLP}(x_i^{\text{LN}}), \quad i = 1, 2, \dots, N$$

$$X_{\text{out}} = \text{LayerNorm}(X_{\text{MLP}})$$

Layer Normalization

На входе — набор векторов x_i (напр. эмбединги)

Для каждого x_i независимо проводится следующая нормализация:

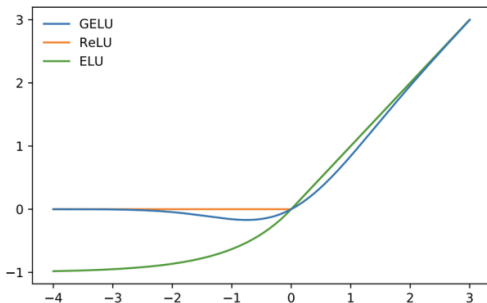
$$\mu_i = \bar{x}_i, \quad \sigma_i^2 = \widehat{D}(x_i)$$
$$x_i \mapsto y = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}} \mapsto z = \gamma_i y + \beta_i$$

Параметры γ_i, β_i обучаемые

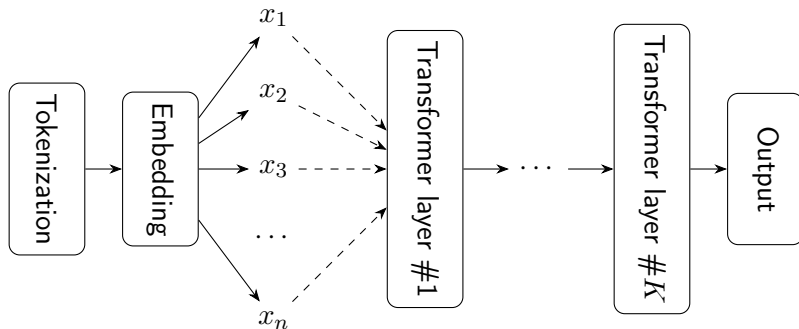
Позже было показано, что применение слоя LayerNorm перед attention- и MLP-блоками даёт более стабильные результаты

Multi-Layer Perceptron (MLP)

- Блок с обычной полносвязной сетью
- Размер входа и выхода должен совпадать (для Residual Connection)
- Размеры скрытых слоёв обычно в несколько (2-4) раз больше размеров входов
- Обычно используется всего 1 скрытый слой (Feed Forward)
- Активация обычно $\text{GELU}(x) = x\Phi(x)$



Трансформерная модель-энкодер



Применение: классификация текстов (выход — вероятности классов), угадывание замаскированных токенов в предложении (выход — вероятности токенов, пример — BERT)

- Величина словаря: $M = 30k$ токенов
- Длина контекста: $N = 512$ токенов
- Размерность эмбединга: $D = 1024$
- Размерности K, V, Q : 64
- Количество голов в МНА: $H = 16$
- Количество трансформерных слоёв: $K = 24$
- Размерность MLP: $D_{\text{MLP}} = 4096$
- Общее количество параметров: $340 \cdot 10^6$

Трансформерная модель-декодер

Та же структура, что и у энкодера, но таргеты — следующий токен в последовательности

Так как задача — предсказывать следующий токен, то нельзя заглядывать вперёд (авторегрессионная модель, вероятность каждого токена должна зависеть только от предыдущих)

Но в self-attention для каждого токена учитываются и все последующие токены

Решение: зафиксировать $a(x_m, x_n) = 0$ для $m < n$

То же самое:

$$\text{MaskedSa}(X) = V \cdot \text{Softmax}(K^T Q + M),$$

$$M(m, n) = \begin{cases} 0, & m \geq n \\ -\infty, & m < n \end{cases}$$

Для GPT-3:

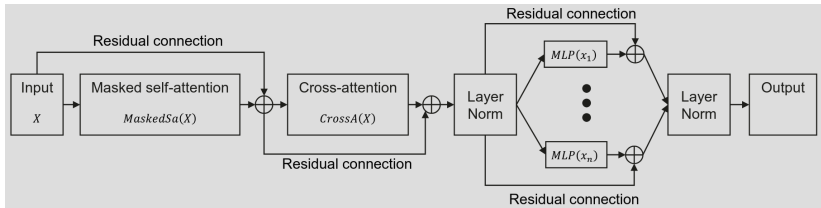
- Величина словаря: неизвестно
- Длина контекста: $N = 2048$ токенов
- Размерность эмбединга: $D = 12288$
- Размерности K, V, Q : 128
- Количество голов в МНА: $H = 96$
- Количество трансформерных слоёв: $K = 96$
- Размерность MLP: неизвестно
- Общее количество параметров: $175 \cdot 10^9$

Трансформенная модель Seq2Seq (энкодер-декодер)

Эмбеддинг входной последовательность подаётся вход в сеть энкодер-трансформеров

Эмбеддинг целевой (target) последовательности подаётся на вход в сеть модифицированных декодер-трансформеров (появляется cross-attention (о нём позже))

Схема обновлённого слоя декодер-трансформера:



Для подсчёта ключей и значений используются выходы энкодер-трансформера

$$K = W_k X_{\text{enc}} + \beta_k$$

$$V = W_v X_{\text{enc}} + \beta_v$$

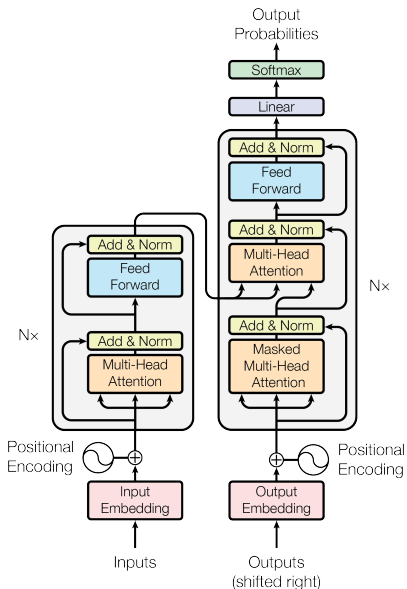
Для запросов используются (преобразованные) входы декодер-трансформера

$$Q = W_q X_{\text{dec}} + \beta_q$$

В остальном всё то же, что и для обычного Self-Attention

$$\text{CA}(X) = V \cdot \text{Softmax}(K^T Q)$$

Итоговая Схема Трансформер-Модели

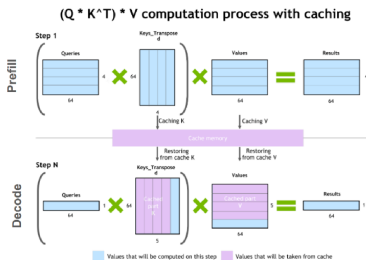


Проблема квадратичности Self-Attention

Честный подсчёт SA имеет временную трудоёмкость $O(N^2)$, где N — количество токенов в последовательности

Варианты оптимизации:

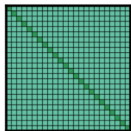
- Параллелизм и кэширование матриц K и V



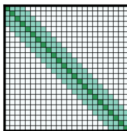
- Использование приближений с меньшей трудоёмкостью

Sliding Window Attention (Longformer)

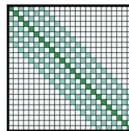
Считаем SA по некоторому окну...



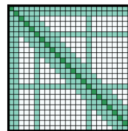
(a) Full n^2 attention



(b) Sliding window attention



(c) Dilated sliding window



(d) Global+sliding window

Эффективность зависит от окна

Переобозначим Q, K, V — входные эмбединги, тогда SA считается как:

$$\text{Sa}(QW_i^Q, KW_i^K, VW_i^V) = \underbrace{\text{Softmax} \left[\frac{QW_i^Q (KW_i^K)^T}{\sqrt{d_k}} \right]}_P VW_i^V,$$

где W_i^* — обучаемые матрицы

Можно доказать, что матрица P хорошо приближается меньшими рангами

Но низкоранговое приближение P считается долго
Зато быстро можно спроектировать K и V

$$\begin{aligned} & \text{Sa}(QW_i^Q, E_iKW_i^K, F_iVW_i^V) \\ &= \underbrace{\text{Softmax}\left(\frac{QW_i^Q(E_iKW_i^K)^T}{\sqrt{d_k}}\right)}_{\bar{P}} \cdot F_iVW_i^V \end{aligned}$$

Linear Attention — Linear Transformer

Self-attention we considered is actually a particular implementation of the self-attention idea which could be called softmax attention:

$$Sa(X) = V \cdot \text{Softmax}(K^T Q)$$

We can generalize it to an arbitrary similarity function akin to Nadaraya-Watson regression:

$$Sa_n(X) = \frac{\sum_{m=1}^N \text{sim}(q_n, k_m) v_m}{\sum_{m=1}^N \text{sim}(q_n, k_m)}$$

Here $\text{sim}(q, k) = \exp(q^T k / \sqrt{D_Q})$ would correspond to the softmax attention.

Now we can kernelize this attention, i.e. choose some kernel $k(x, y)$ which implies a feature map $\phi(x)$ and turns the similarity function into

$$Sa_n(X) = \frac{\sum_{m=1}^N \phi(q_n)^T \phi(k_m) v_m}{\sum_{m=1}^N \phi(q_n)^T \phi(k_m)}$$

which can be a bit simplified from the computational viewpoint:

$$Sa_n(X) = \frac{\phi(q_n)^T \sum_{m=1}^N \phi(k_m) v_m^T}{\phi(q_n)^T \sum_{m=1}^N \phi(k_m)}$$

Note that highlighted terms can be computed only once and then used for all the queries.