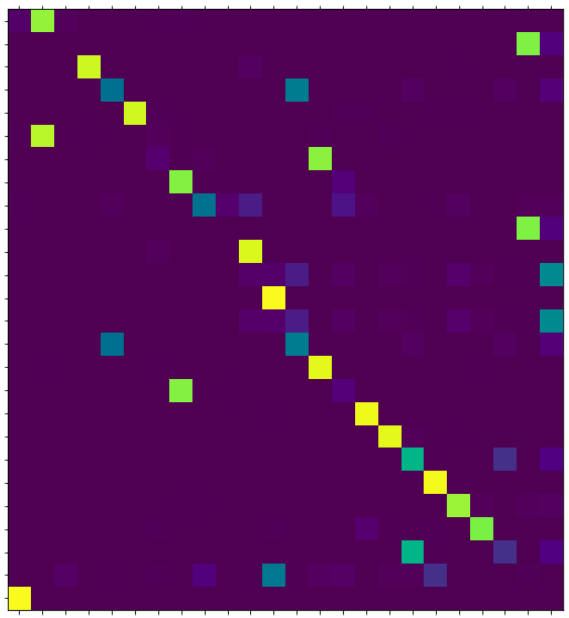
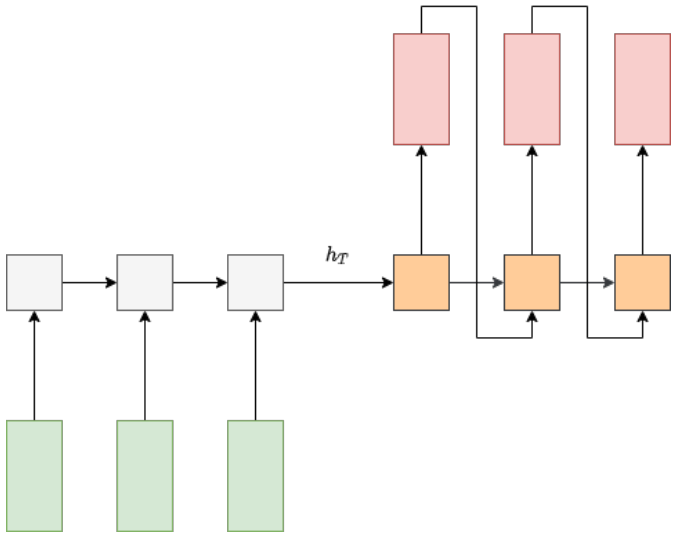
Механизм внимания и трансформер

Виктор Китов
victorkitov.github.io



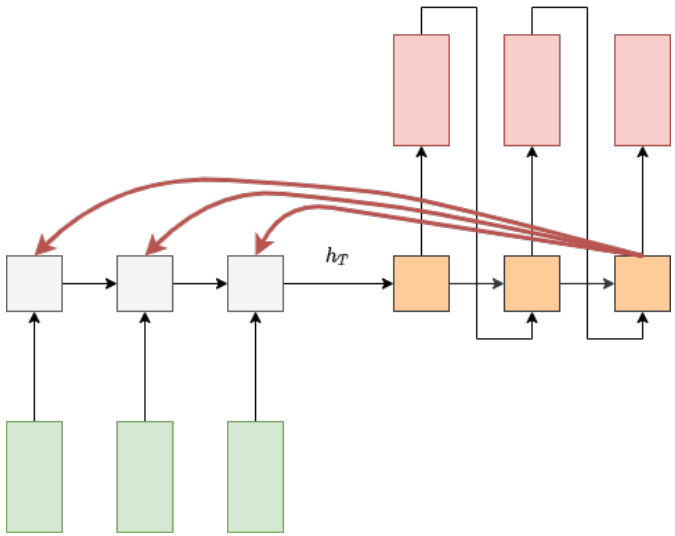
Содержание

Модель seq2seq с вниманием



Проблемы с кодировкой длинной последовательности вектором фиксированной длины.

Модель seq2seq с вниманием[1]



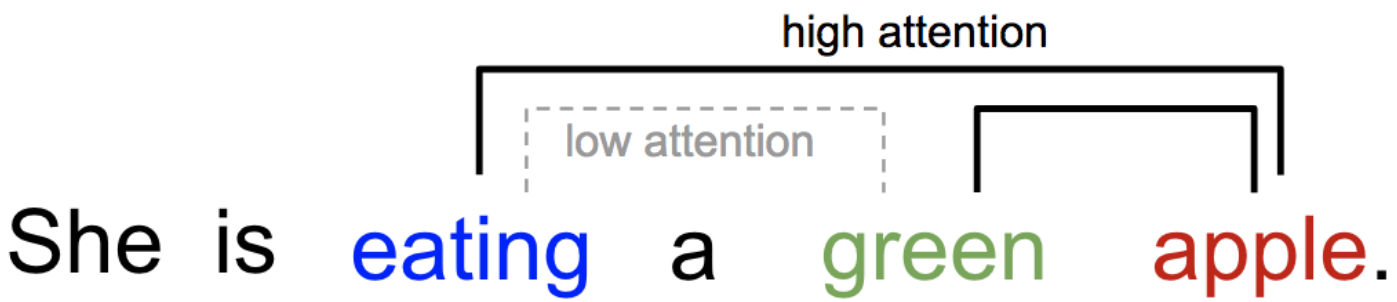
seq2seq со вниманием (attention): в каждый момент состояние учитывает все слова по отдельности из входной последовательности.

[1] https://arxiv.org/pdf/1409.0473.pdf

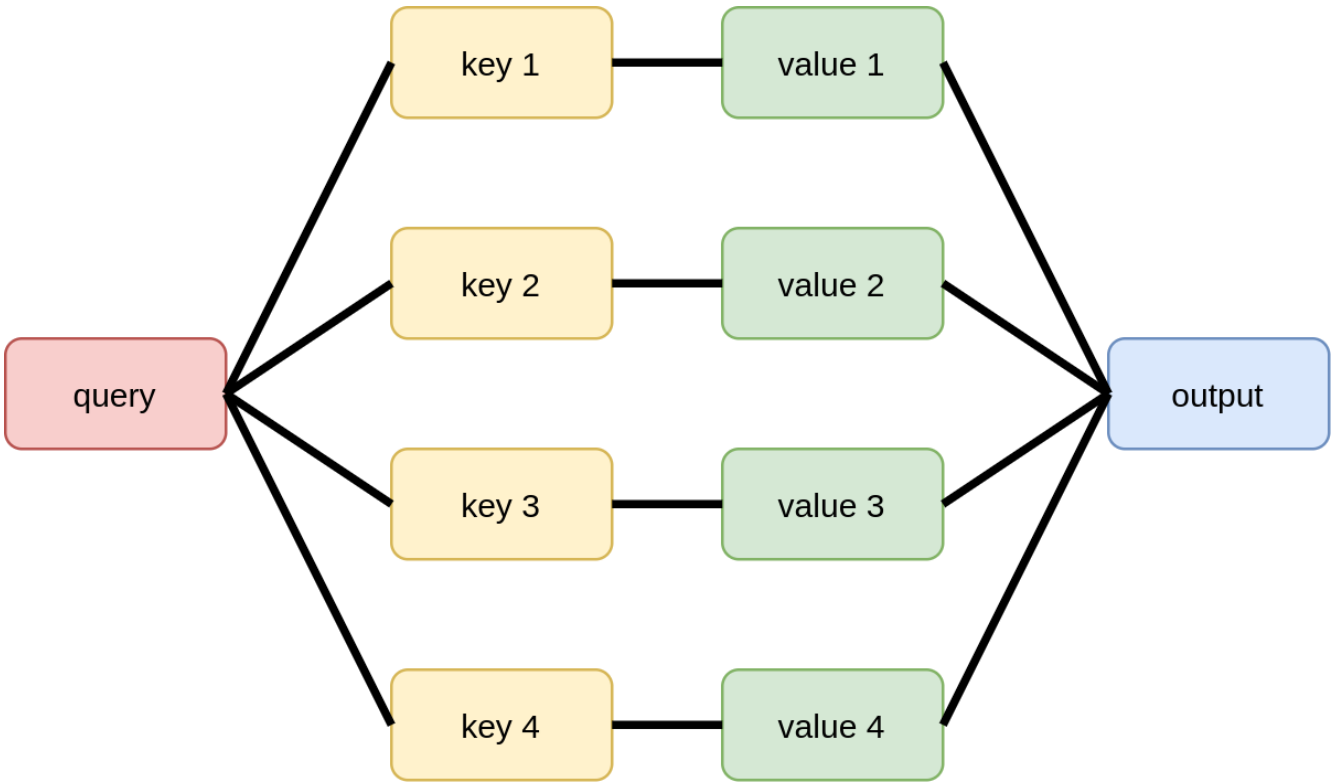
Внимание между словами

При переводе слова важен контекст словоупотребления:

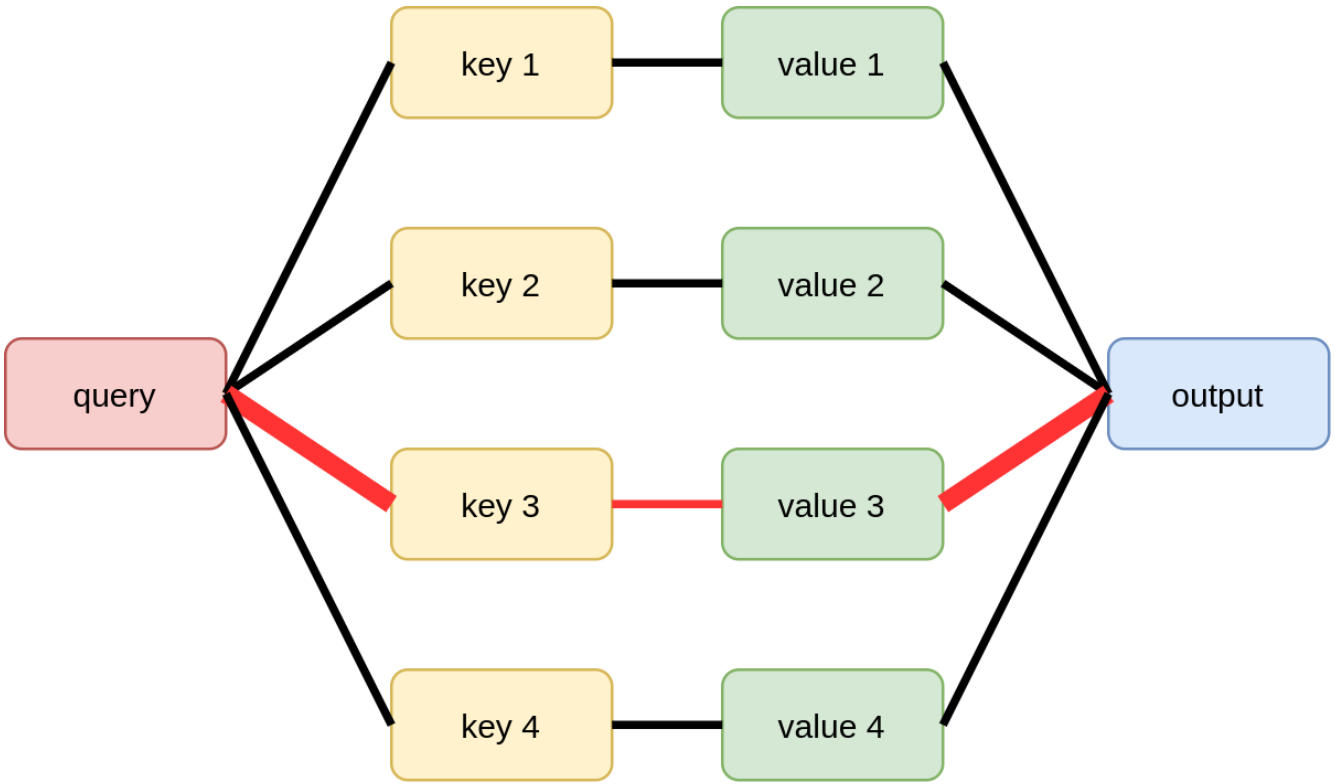
- "она ела зеленое яблоко"
- "она ела зеленые яблоки"
- "она ела зеленую капусту"



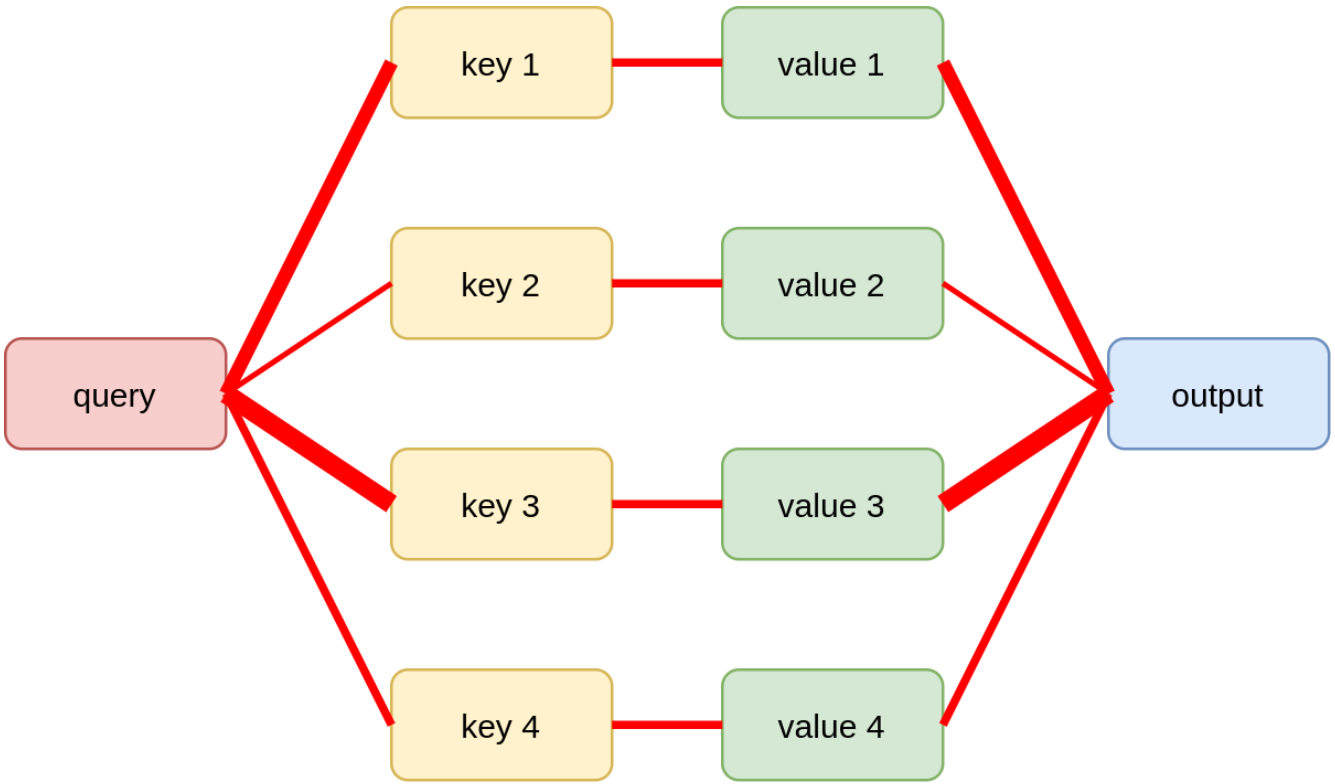
Внимание в БД и нейросетях (soft attention)



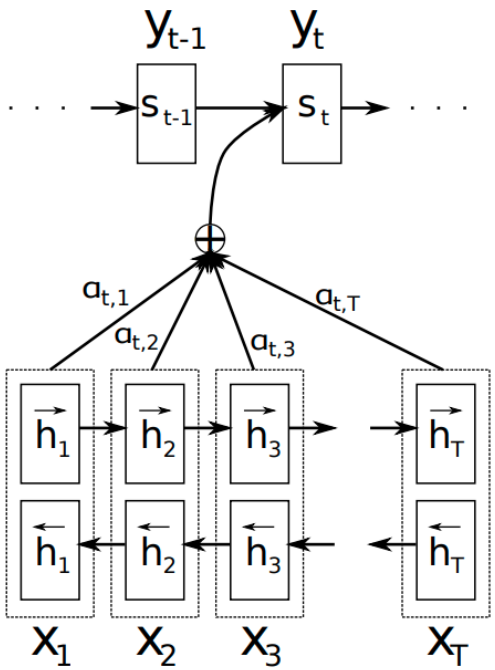
Внимание в БД и нейросетях (soft attention)



Внимание в БД и нейросетях (soft attention)



seq2seq с вниманием[2]



Кодировщик - двунапр. RNN

- состояния конкатенируются

- Степень соответствия:

$$e_{tj} = \text{score}\left(s_{t-1}, h_j\right),\ j = 1, 2, ...T$$

- Веса учета состояний:

$$\alpha_{ti} = \exp(e_{ti}) / \sum_{j=1}^{T} \exp(e_{tj}),$$

$$i = 1, 2, ...T$$

- Контекстный вектор:

$$c_t = \sum_j \alpha_{tj} h_j$$

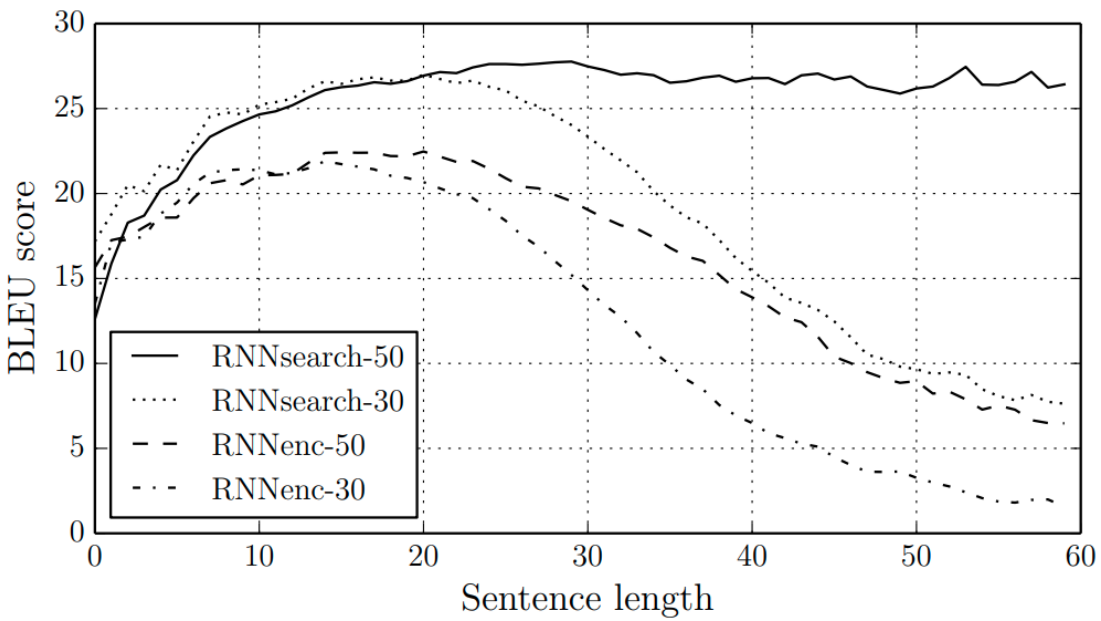
- Пересчёт состояний:

$$s_t = f\left(s_{t-1}, y_{t-1}, c_t\right)$$

[2] https://arxiv.org/pdf/1409.0473.pdf

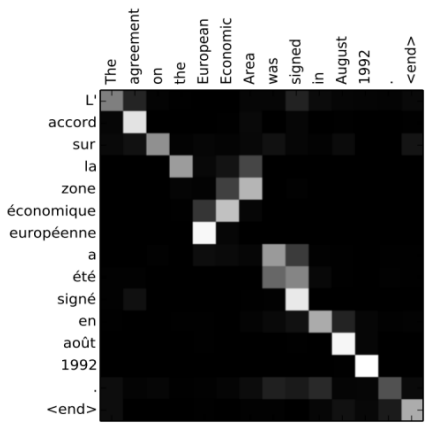
Результаты работы

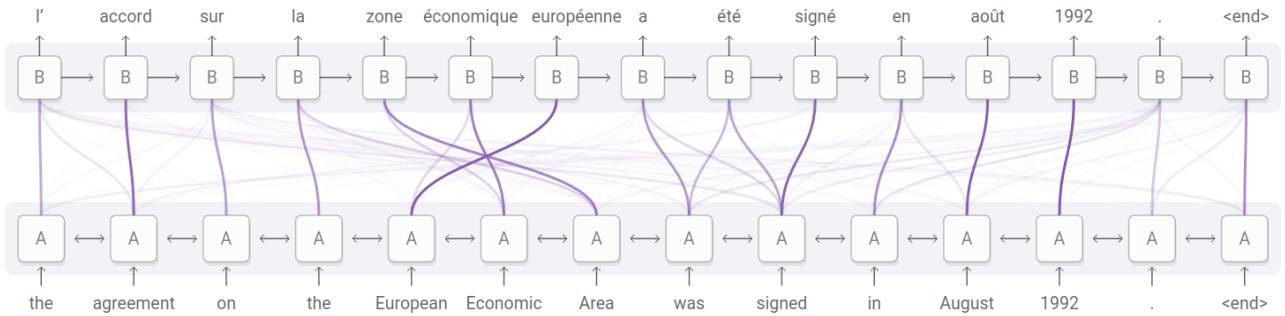
- RNNenc-X - seq2seq
- RNNsearch-X - seq2seq со вниманием
- Обучение: машинный перевод предложений длины до X слов.



Визуализация весов α_{tj}

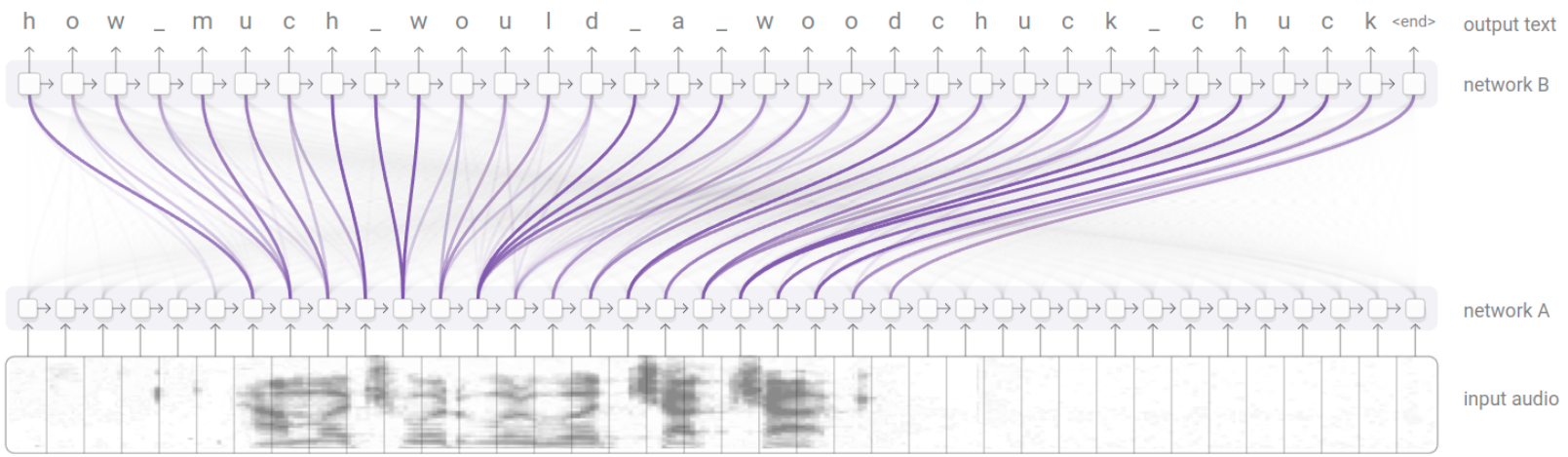
Визуализация весов α_{tj} (перевод английский->французский):



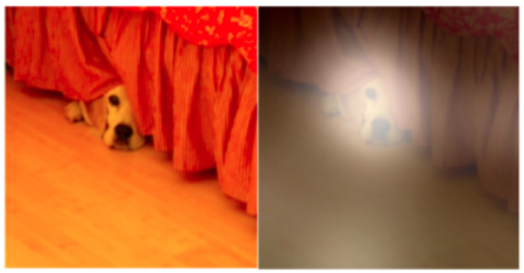


- Матрица близка к диагональной => сеть выучилась.
  - можно явно добавить диагонолизирующий регуляризатор

Визуализация внимания в других задачах[3]




A woman is throwing a frisbee in a park.


A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

[3] Источник иллюстрации.

Варианты функции соответствия

Варианты соответствия $e_{tj} = \text{score}\,(s_{t-1}, h_j)$, $s_{t-1}, h_j \in \mathbb{R}^d$

basic dot-product	$s^T h$
scaled dot-product	$s^T h / \sqrt{d}$
content-based attention	$s^T h / \left(\|s\| \, \|h\|\right)$
additive attention	$w^T \tanh(W_1 s + W_2 h)$
multiplicative attention	$s^T W h$
general (в ориг. статье)	$\text{MLP}(s, h)$

Локальное внимание[4]

- Обычное внимание усредняет по всем состояниям входа.
  - долго работает (например-перевод целого абзаца)

[4] https://arxiv.org/pdf/1508.04025.pdf

Локальное внимание[4]

- Обычное внимание усредняет по всем состояниям входа.
  - долго работает (например-перевод целого абзаца)

- Локальное внимание (local attention):
  - контекст c_t зависит только от $[p_t - D,\ p_t + D]$, веса $\alpha_t \in \mathbb{R}^{2D+1}$
  - $\alpha_{tj} = \text{score}\left(s_{t-1}, h_s\right) \exp\left(-\frac{(j-p_t)^2}{2\sigma^2}\right),\ \sigma = \frac{D}{2}$.

[4] https://arxiv.org/pdf/1508.04025.pdf

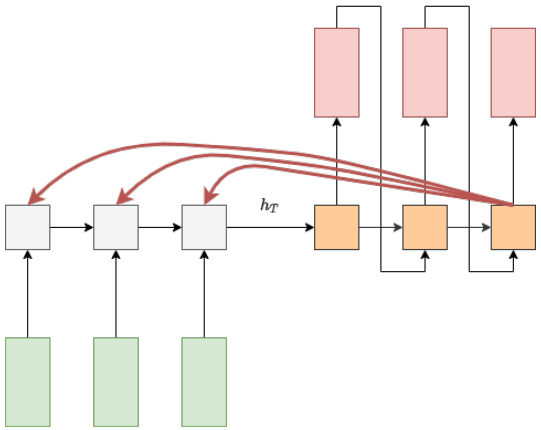
Локальное внимание[4]

- Обычное внимание усредняет по всем состояниям входа.
  - долго работает (например-перевод целого абзаца)
- Локальное внимание (local attention):
  - контекст c_t зависит только от $[p_t - D,\ p_t + D]$, веса $\alpha_t \in \mathbb{R}^{2D+1}$
  - $\alpha_{tj} = \text{score}\left(s_{t-1}, h_s\right) \exp\left(-\frac{(j-p_t)^2}{2\sigma^2}\right),\ \sigma = \frac{D}{2}$.
- Варианты генерации p_t:
  - $p_t = t$ (предполагаем входная и выходная посл-ти выровнены)
  - $p_t = [\text{input length}] \times \sigma\left(v^T \tanh\left(W s_{t-1}\right)\right)$

[4] https://arxiv.org/pdf/1508.04025.pdf

Содержание

Модель seq2seq с вниманием

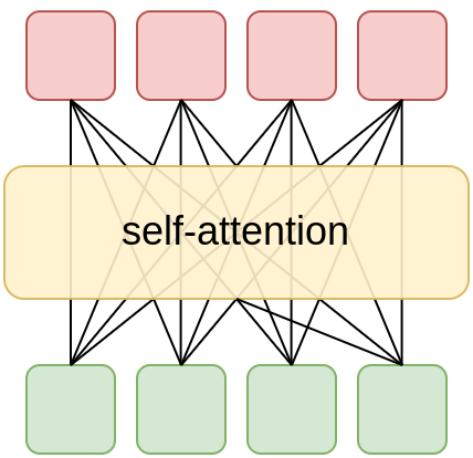


- seq2seq+attention:
  - всю входную посл-ть не нужно запоминать вектором
  - но всё еще нужно запоминать информацию об уже сгенерированной последовательности (state)
- Хотим помнить всю входную и выходную последовательность (к текущему моменту).
- Используем трансформер[5])
  - полностью отказались от рекуррентности - больше параллелизации, быстрее.

[5] https://arxiv.org/pdf/1706.03762.pdf

Модуль самовнимания (self-attention)

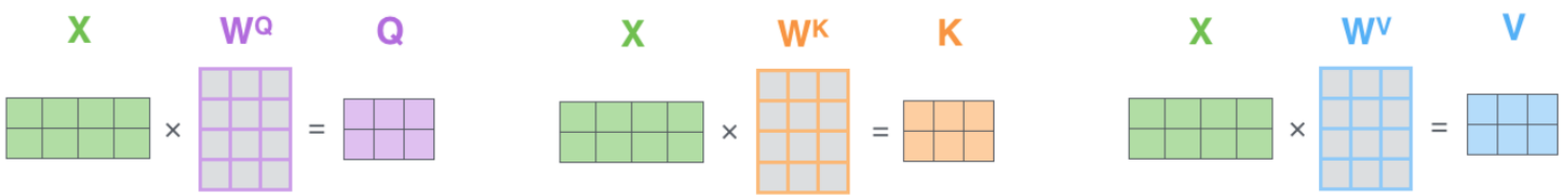
- Transformer - SOTA для машинного перевода и др. задач на последовательностях.
  - вход: эмбеддинги слов, выход: распределения слов.
- Проблема: слова в контексте приобретают другой смысл.
- Решение: модуль self-attention-преобразует s входов в s выходов
  - размерность входов и выходов: D
  - зависимость: каждый от каждого



Модуль самовнимания (self-attention)[6]

- $X_{T\times D}$ - T входов размерности, размерность эмбеддинга $D = 512$.

- Генерируем для каждого входа соответствующие
  - запросы (queries): $Q_{T\times d} = X_{T\times D}W^Q_{D\times d}$
  - ключи (keys): $K_{T\times d} = X_{T\times D}W^K_{D\times d}$
  - значения (values): $V_{T\times \bar{d}} = X_{T\times D}W^V_{D\times \bar{d}}$
- $d, \bar{d} = 64$, т.к. потом конкатенируем 8 раз для разных W^K, W^V, W^Q (много аспектные контексты)



[6] Иллюстрации работы трансформера.

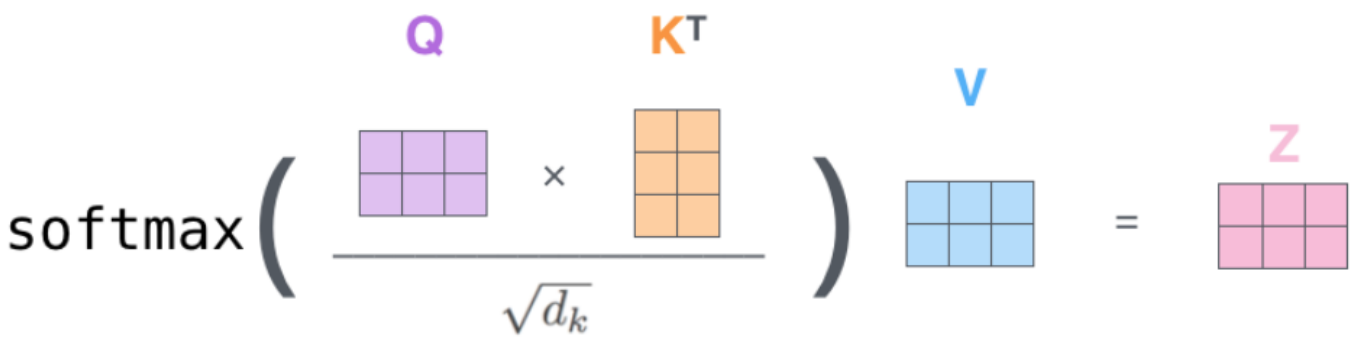
Модуль самовнимания (self-attention)

Выход для одного входа:

$$y_{1\times\bar{d}} = \text{softmax}\left(\frac{1}{\sqrt{d}} q_{1\times d}\left(K^T\right)_{d\times T}\right)_{1\times T} V_{T\times\bar{d}}$$

В матричной форме:

$$Y_{T\times\bar{d}} = \text{softmax}\left(\frac{1}{\sqrt{d}} Q_{T\times d}\left(K^T\right)_{d\times T}\right)_{T\times T} V_{T\times\bar{d}}$$



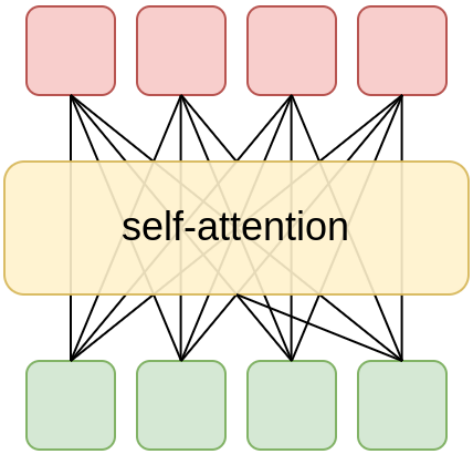
$\frac{1}{\sqrt{d}} q^T k$ из логики, что сумма d случайных величин имеет σ в $\sqrt{d}$ раз больше.

Модуль самовнимания (self-attention)

Одна головка самовнимания:

$$\begin{aligned}
&\text{head}\left(X|W^K,W^V,W^Q\right)_{T\times\bar{d}} \\
&= \text{softmax}\left(\frac{1}{\sqrt{d}}Q_{T\times d}\left(K^T\right)_{d\times T}\right)_{T\times T} V_{T\times\bar{d}} \\
&= \text{softmax}\left(\frac{1}{\sqrt{d}}\left(\underbrace{XW^Q}_{Q}\right)\left(\underbrace{XW^K}_{K}\right)^T\right)\underbrace{XW^V}_{V}
\end{aligned}$$

Self-attention vs. RNN



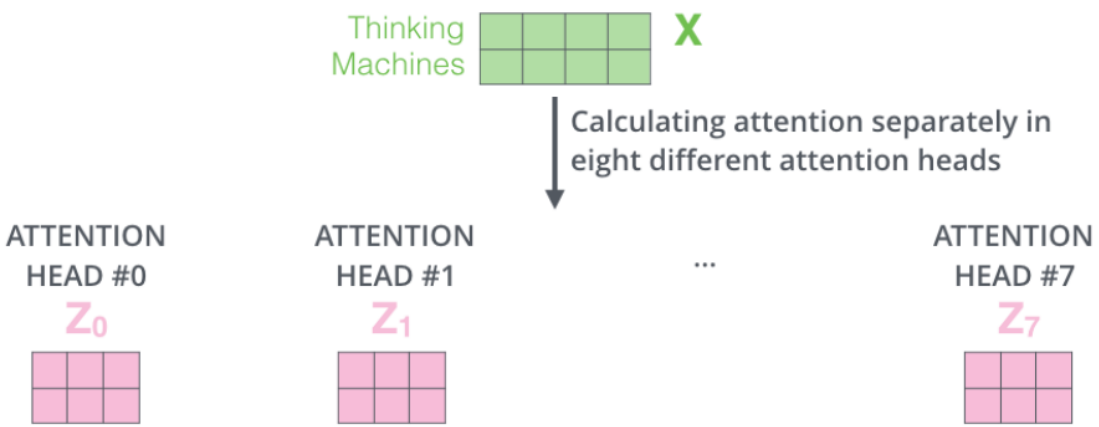
Сложность сканирования последовательности:

- в RNN: $O(T \cdot D^2)$, в self-attention: $O(T^2 \cdot D)$. $(D = 512 > T)$
  - можно еще уменьшить, если для i-го токена учитывать только контекст $[-i - r, ..., i + r]$.
    Средняя длина пути между элементами
  - В RNN: $O(T)$, в self-attention: $O(1)$
    - лучше учитывается контекст

Self-attention параллелизуется, а RNN - нет.

Multi-head attention

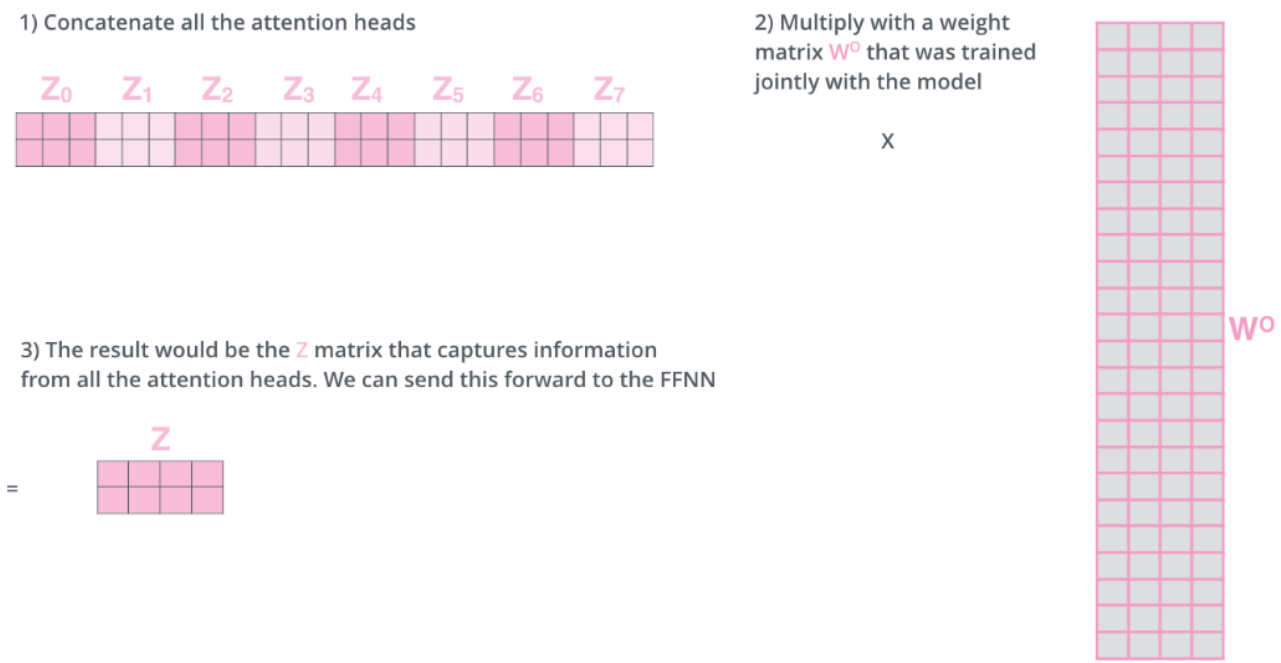
Используется 8 головок (каждая - со своими W^Q, W^K, W^V).



Модуль самовнимания (self-attention)

Итоговый выход $\in \mathbb{R}^{D\times T}$- конкатенация выходов+линейное преобразование:

$$Z = \text{concat}_{T\times 8\bar{d}}\left[\text{head}\left(X|W_n^K, W_n^V, W_n^Q\right)\right]_{n=1}^{8} W^O_{8\bar{d}\times D}$$



Модуль самовнимания (self-attention)

1) This is our input sentence*

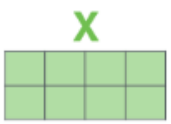
2) We embed each word*

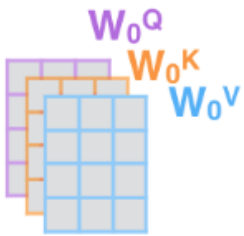
3) Split into 8 heads. We multiply X or R with weight matrices

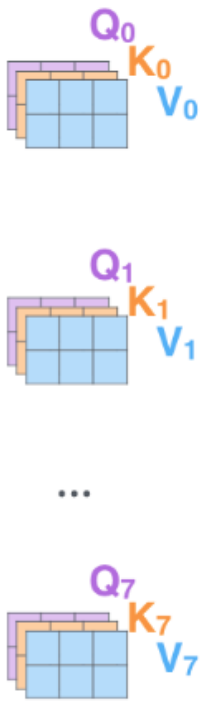
4) Calculate attention using the resulting Q/K/V matrices

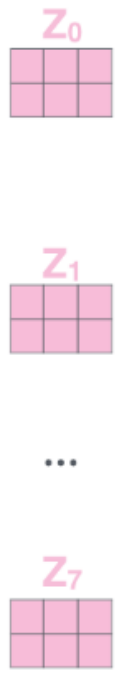
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

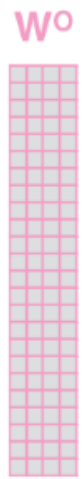
Thinking
Machines

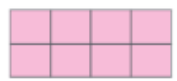












* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one

Пример: на что смотрят блоки самовнимания

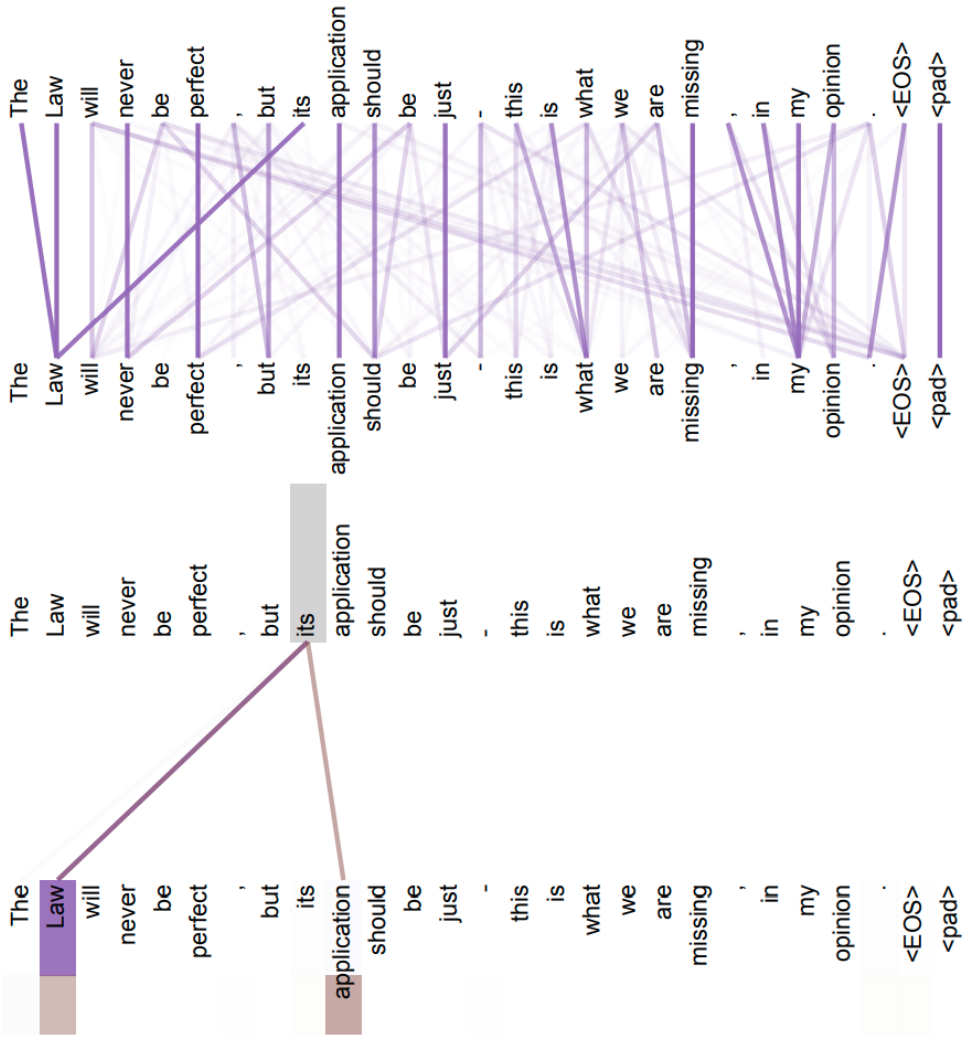


Figure 4: Two attention heads, also in layer 5 of 6, apparently involved in anaphora resolution. Top: Full attentions for head 5. Bottom: Isolated attentions from just the word 'its' for attention heads 5 and 6. Note that the attentions are very sharp for this word.

Пример: на что смотрят блоки самовнимания

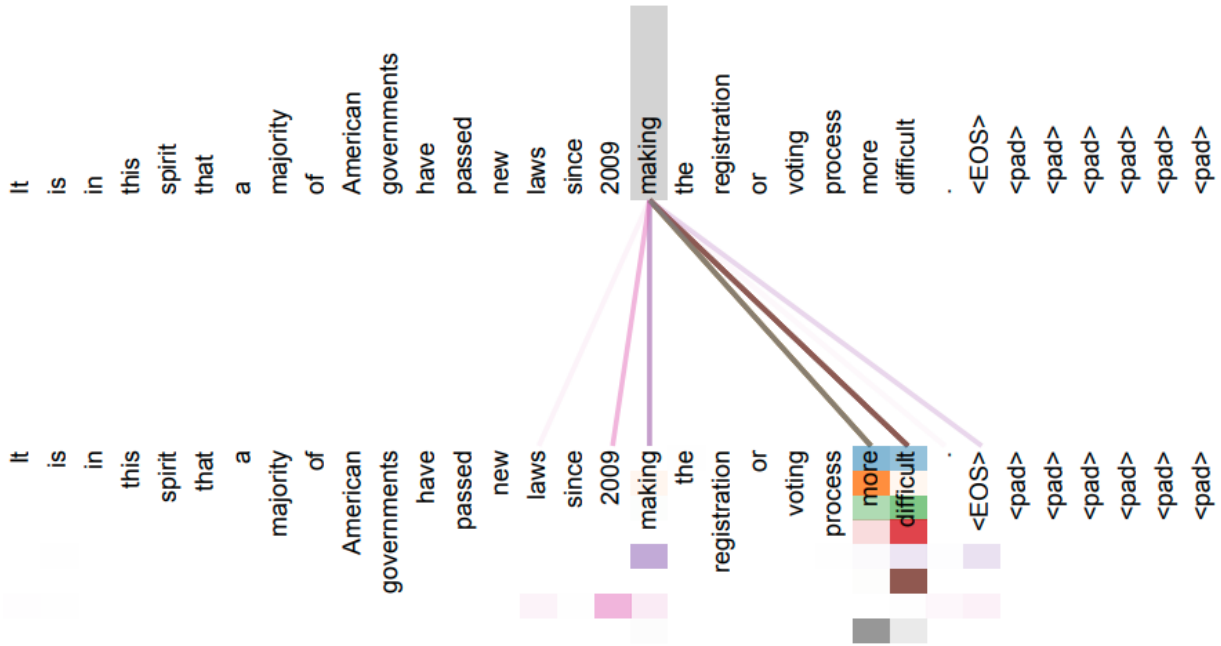
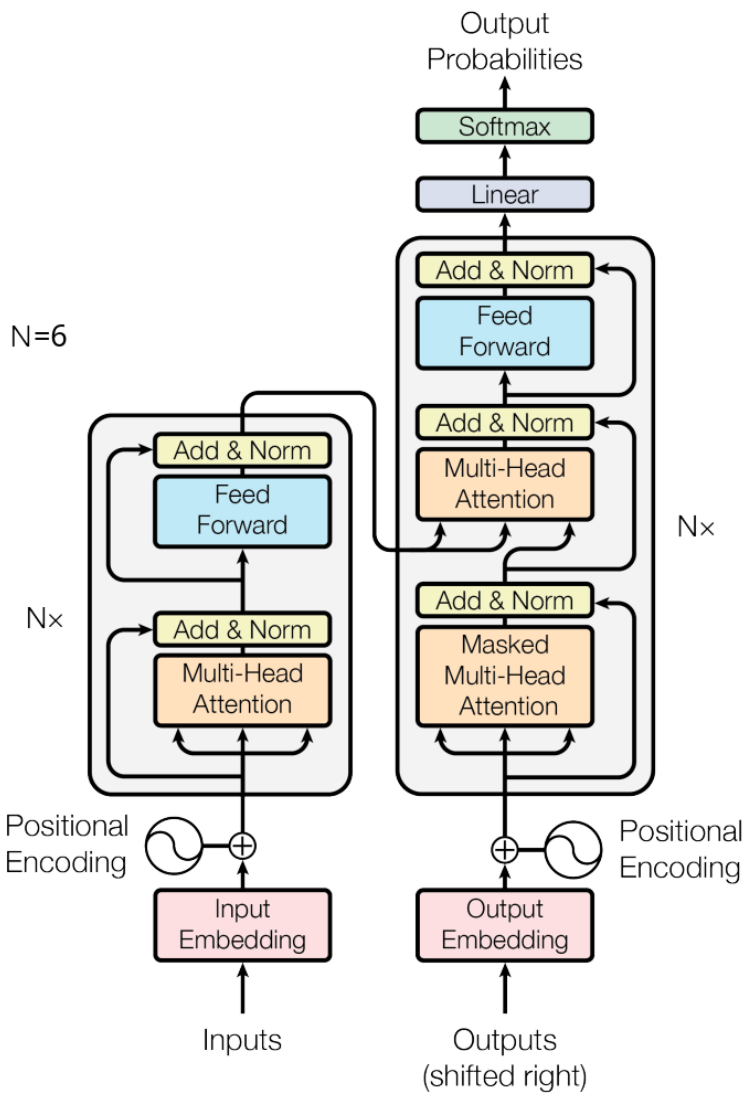


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word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

- <pad> на входе - для выравнивания посл-тей минибатча.
- Внимание маскировалось, чтобы не смотреть на <pad>.

Трансформер: вся модель



- Feed Forward: сеть с одинаковыми весами, применяемая к каждому элементу

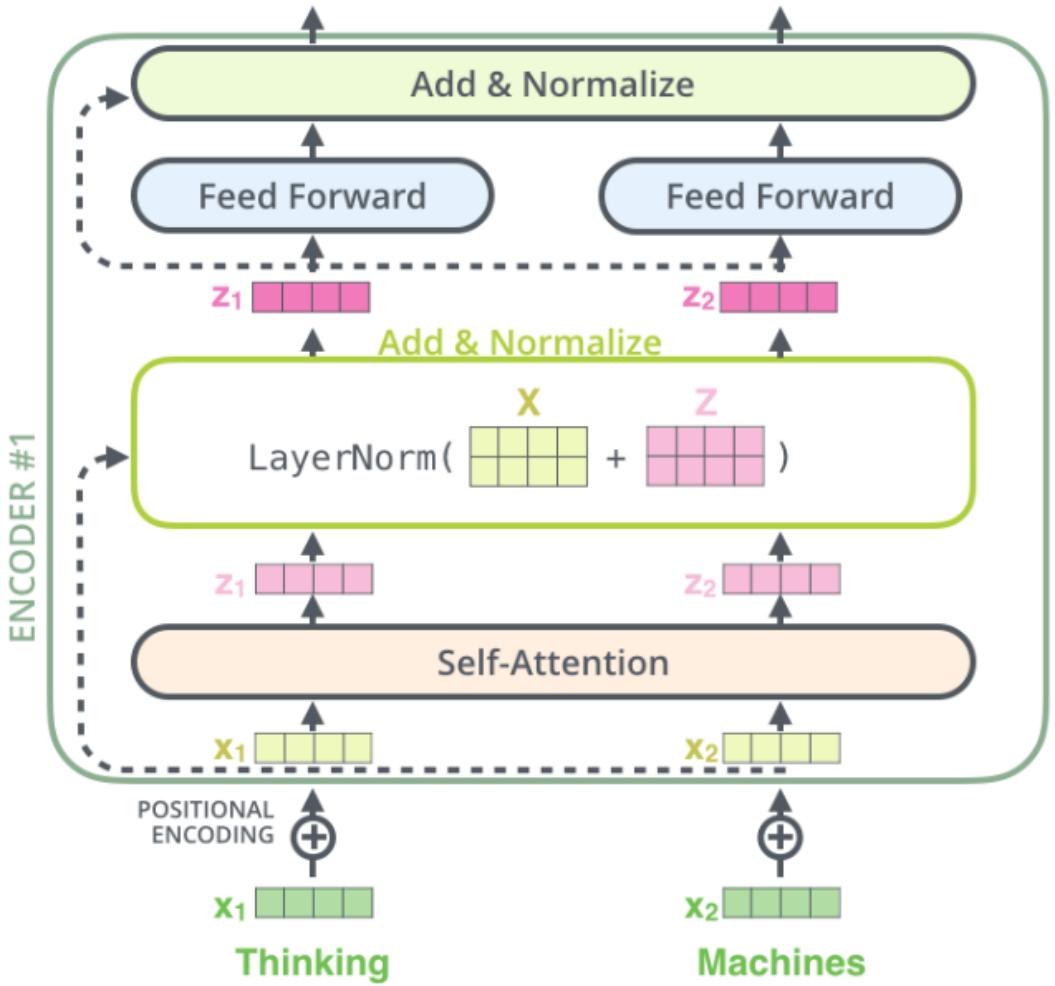
$$out = ReLU\left(xW_1 + b_1\right)W_2 + b_2$$

- Эмбеддинги позиций прибавляются к эмбеддингам слов.
- Блоки повторяются $N = 6$ раз.
- Add&Norm:

$$LayerNorm\left(x + SubLayer\left(x\right)\right)$$

- Кодировщик работает сразу.
- Декодировщик-в авторегрессионном режиме много раз до <EOS>.

Визуализация первого блока кодировщика



Детали

- Нормализация слоя (LayerNorm) $[x_1, ...x_D]$:

$$x := \alpha \frac{x - \mu}{\sigma} + \beta, \quad \mu = \frac{1}{D}\sum_{i=1}^{D} x_i, \quad \sigma = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(x_i - \mu)^2}$$

  - α, β - выучиваемые параметры
  - работает независимо для каждого объекта (здесь-токена)
  - обучение и применение не различаются

Детали

- Нормализация слоя (LayerNorm) $[x_1, ...x_D]$:

$$x := \alpha \frac{x - \mu}{\sigma} + \beta, \quad \mu = \frac{1}{D}\sum_{i=1}^{D} x_i, \quad \sigma = \sqrt{\frac{1}{D}\sum_{i=1}^{D}(x_i - \mu)^2}$$

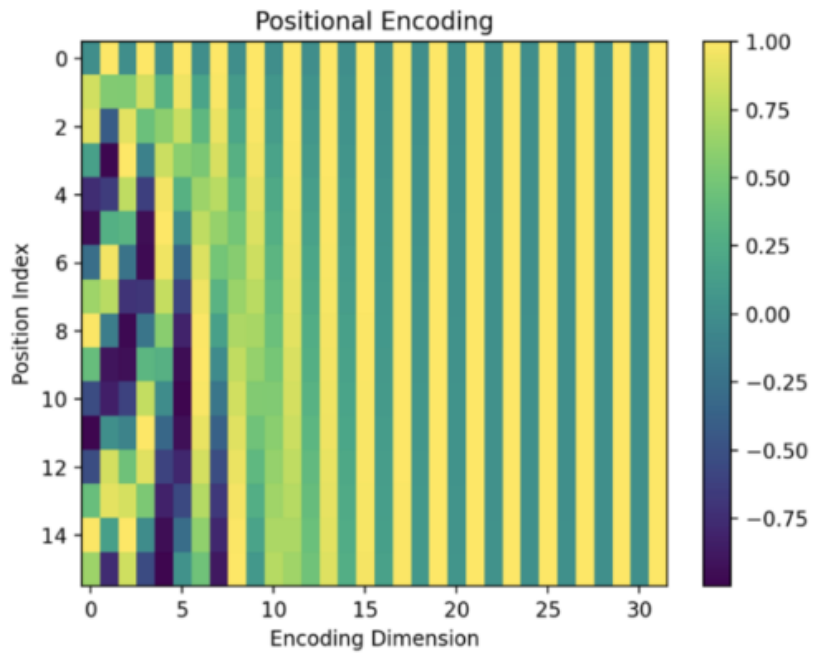
  - α, β - выучиваемые параметры
  - работает независимо для каждого объекта (здесь-токена)
  - обучение и применение не различаются
- Positional embedding: кодирует расположение слов.
  - pos - позиция слова, i - индекс D-мерного эмбеддинга

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/D}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/D}\right)$$

  - Периоды $[2\pi - 10000 \cdot 2\pi]$.

Позиционное кодирование

Позиционный эмбеддинг $\in \mathbb{R}^{32}$:

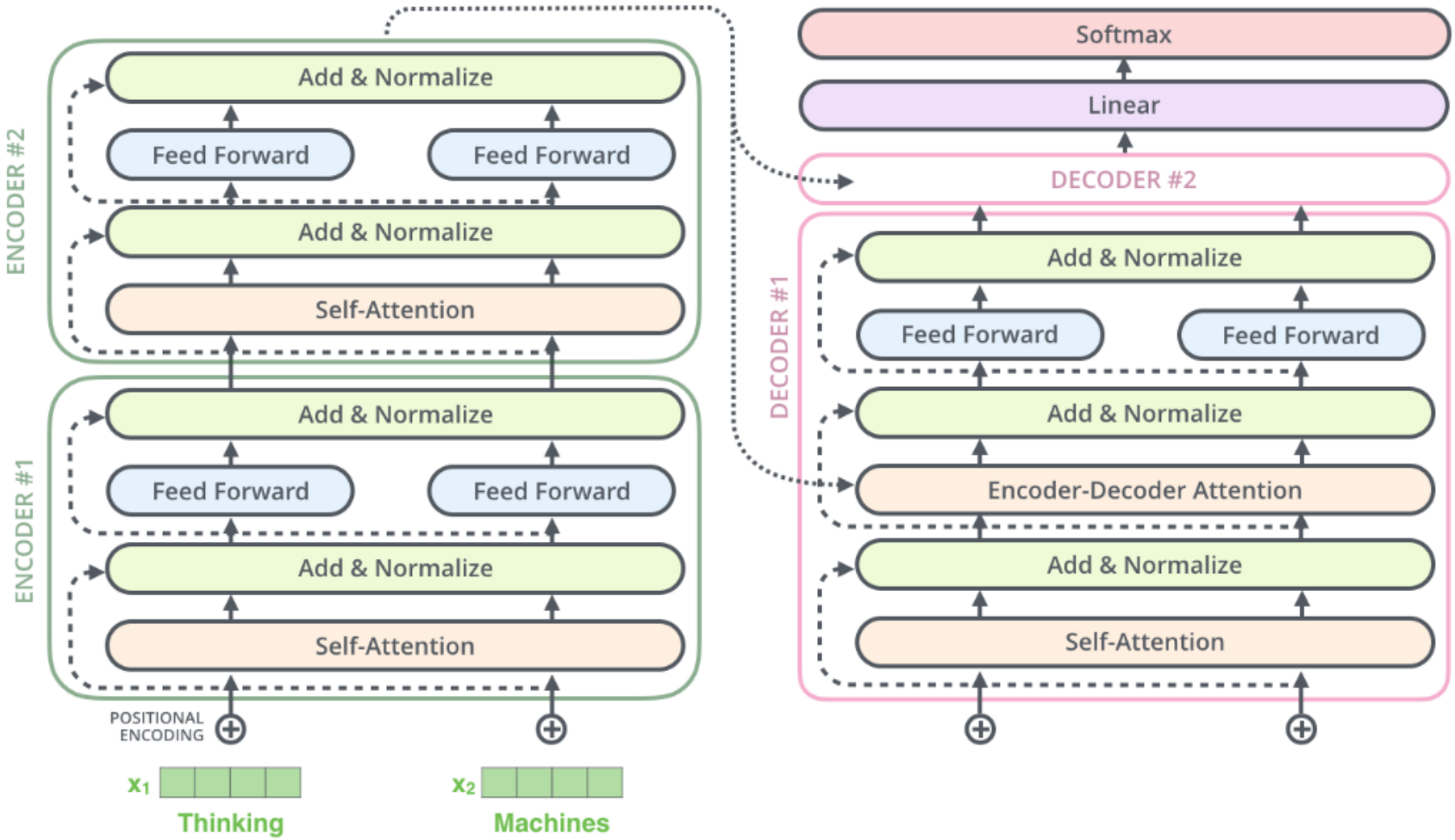


Аналогия:

$$0 \to (0,0,0)$$
$$1 \to (1,0,0)$$
$$2 \to (0,1,0)$$
$$3 \to (1,1,0)$$
$$4 \to (0,0,1)$$
$$5 \to (1,0,1)$$
$$6 \to (0,1,1)$$
$$7 \to (1,1,1)$$

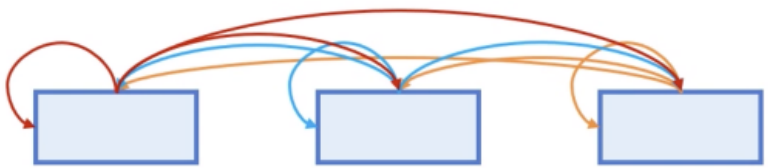
Полосы - sin/cos.

Схема двухуровневого трансформера



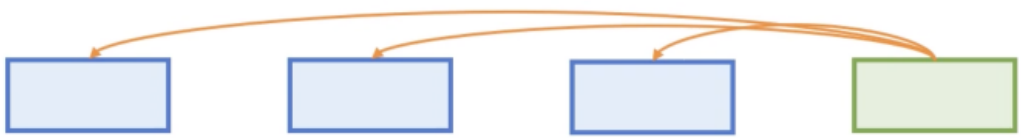
Виды внимания

Encoder Self-Attention:



Masked Decoder Self-Attention:

Encoder-Decoder Attention:

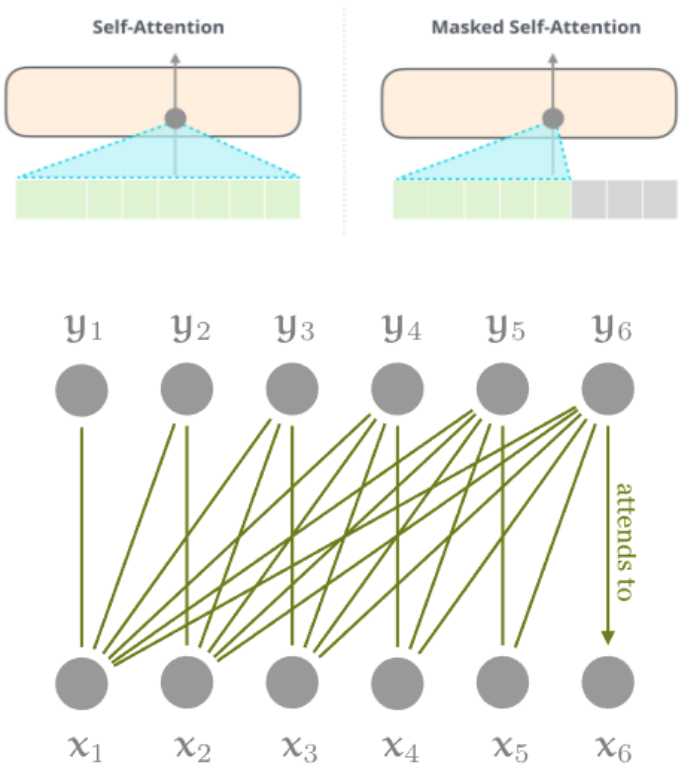


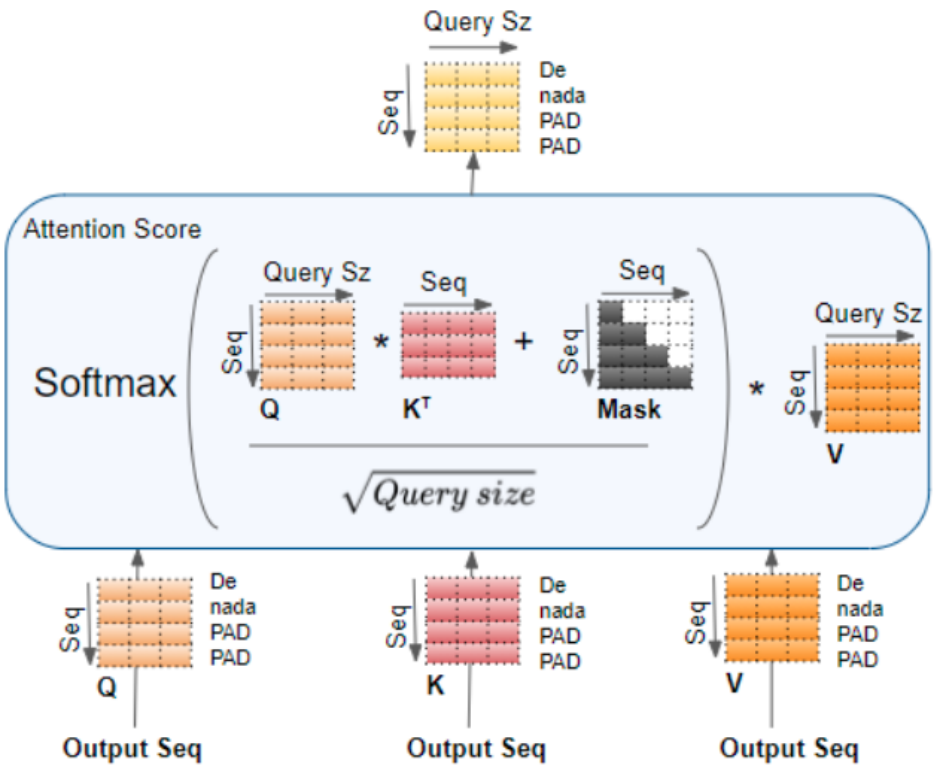
Виды внимания

- в кодировщике Q,K,V считаются:
  - в первом блоке: по эмбеддингам слов+позиций
  - в последующих блоках: по выходам кодировщика пред. блока
- в декодировщике:
  - masked multi-head attention:
    - в первом блоке: по эмбеддингам предсказанных слов+позиций (маскированным)
    - в последующих блоках: по выходам декодировщика (маскированным)
  - encoder-decoder attention: Q-по выходам декодировщика, K,V - по финальным выходам кодировщика

Маскирование[7]

- Masked multi-head attention декодировщика: элемент i не должен смотреть на $i+1, i+2, \ldots$ (их еще нет).
- Прибавляем $-\infty$ к соотв. аргументам SoftMax:





[7] Источник иллюстрации.

Особенности настройки

- Использовался dropout:
  - в residual-блоках:

$$LayerNorm\left(x + DropOut \odot SubLayer\left(x\right)\right)$$

  - в суммам эмбеддингов в кодировщике и декодировщике.
  - $p_{drop} = 0.1$
- Сглаживались метки классов (слов).
- Пример кода на PyTorch с комментариями.

Содержание

Только кодировщик

- Модели, использующие только кодировщик трансформера:
  - BERT, DistilBERT, RoBERTa, ALBERT, ELECTRA, ...
- Обучаются на задачах
  - на токенах:
    - Masked Language Modeling, Replaced Token Detection.
  - на предложениях:
    - Next Sentence Prediction, Sentence Order Prediction
- Выдают эмбеддинги токенов и эмбеддинг всего предложения (для [CLS]-токена)
- На эмбеддингах настраиваем модель под свою задачи (трансферное обучение)
  - на токенах: token-level задачу
  - на [CLS]: sentence-level задачу

Только декодировщик

- Модели, использующие только кодировщик трансформера:
  - GPT, LLaMA, Gemini, Mistral, Qwen, ...
- Обучаются на задаче Next Token Prediction.
  - энциклопедии, книги, статьи, новости, коды программ, ...
- Дообучаются, используя
  - Instruction Tuning (учатся отвечать на вопрос)
  - Dialogue Fine-tuning (учатся поддерживать контекст беседы)
  - Обучение с подкреплением от пользователей/верифицирующей языковой модели
    - получен в итоге ответ или нет?
- Поддерживают диалог (за счёт включения его в контекст).

Структурные улучшения

- Токенайзер BPE (byte pair encoding)
- Позиционные эмбеддинги - Rotary position embeddings.
- $\downarrow$ вычислений, используя смесь экспертов (mixture of experts).
- Уточнение информации через поисковые запросы к базе данных
  - называется RAG (Retrieval Augmented Generation)
  - по вопросу ищутся релевантные документы, которые вставляются в контекст диалога перед ответом.
- Возможность запускать внешние сервисы через API (агенты)
  - Модель пишет вызов API, результаты которого добавляются в контекст.

Локализация моделей под задачу

При локализации предобученной модели под нашу задачу существуют следующие подходы:

- без дообучения:
  - **Prompt Engineering**: автодополнение запроса под задачу
- с дообучением:
  - **Prompt Tuning**: добавление ~20 токенов в каждый запрос, эмбеддинги которых учатся под задачу (веса остальной модели зафиксированы).
    - *Используется, когда мало данных (<1000 примеров) или ограничения по памяти.*
  - **LoRA** (low rank adaptation)
    - *Используется, когда больше данных и память позволяет.*

LoRA

$$W' = W + BA$$

- где:
  - $W \in \mathbb{R}^{d_{\text{in}} \times d_{\text{out}}}$ — замороженные предобученные веса
  - $B \in \mathbb{R}^{d_{\text{in}} \times r}$, $A \in \mathbb{R}^{r \times d_{\text{out}}}$ — обучаемые низкоранговые матрицы
- $r \ll \min(d_{\text{in}}, d_{\text{out}})$ — ранг LoRA, контролирует емкость адаптации
- Применение (в порядке убывания значимости):
  - Запросы (что ищем), значение (что находим), агрегация голов внимания (что извлекаем), ключи (как ищем), FC-слои (знания модели).

Ускорение генерации

- Ускорение авторегрессионной генерации:
  - **KV-кэширование** (KV caching): кэширование ключей и значений ранее сгенерированных токенов
    - старые токены зафиксированы, поэтому перевычислять их нет смысла.
    - но они понадобятся при генерации следующего токена.
  - **Спекулятивное декодирование** (speculative decoding):
    - вместе с базовой моделью используется драфт-модель (быстрая, но неточная)
    - драфт-модель генерирует продолжение текста
    - основная модель проверяет корректность продоолжения за 1 проход
    - генерация продолжается с 1го термина, на котором была допущена ошибка

Заключение

- Рекуррентная сеть
  - теряет информацию о длинных посл-тях из-за $\mathbf{h} \in \mathbb{R}^K$
  - медленно обучается из-за рекуррентности
- Трансформер:
  - помнит каждый входной и выходной токен
  - параллелизм:
    - кодировщик - всегда
    - декодировщик - на этапе обучения (генерация авторегрессионная)
- У трансформера $\uparrow$ требования на производительность и память:
  - считаем связи каждый с каждым
  - помним Q,K,V всех токенов
- Развитие: признаки для $\forall$ NLP-задачи, чат-боты, RAG-системы, агенты.