САМОВНИМАНИЕ И ТРАНСФОРМЕРЫ

Сергей Николенко

СПбГУ — Санкт-Петербург

09 октября 2025 г.

Random facts:

- 9 октября 768 г. Карломан I и Карл Великий стали королями франков после смерти их отца Пипина Короткого; впрочем, Карломан умер уже в 771 году, и Карл стал единственным королём франков, а затем и императором
- 9 октября 1446 г. Седжон Великий заменил иероглифическое письмо новым алфавитом с 28 буквами — хангылем
- 9 октября 1604 г. европейцы впервые увидели сверхновую SN 1604 (Сверхновую Кеплера), вспыхнувшую в нашей Галактике в созвездии Змееносца
- 9 октября 1780 г. сформировался Великий ураган 1780 года, который стал самым смертоносным ураганом Атлантики за всю историю наблюдения, унеся за 10 дней до 22 тысяч жизней
- 9 октября 1789 г. во Франции были запрещены пытки, а 9 октября 1981 г. Франсуа Миттеран запретил и смертную казнь
- 9 октября 1967 г., на следующий день после пленения, боливийские солдаты вместе с агентами ЦРУ казнили команданте Кубинской революции Эрнесто Че Гевару

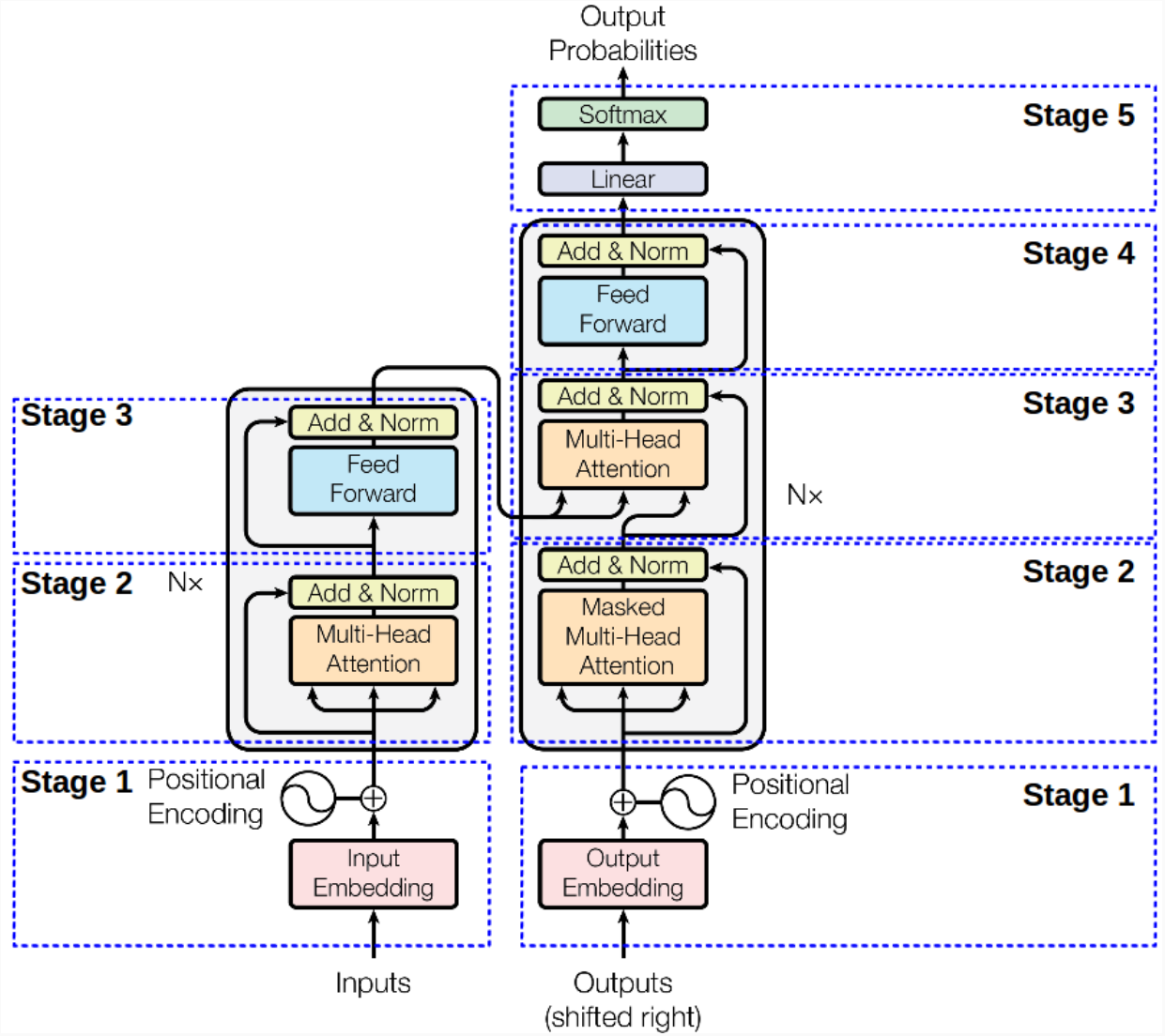
Самовнимание и Transformer

Transformer

- Мы изучали перевод на рекуррентных сетях и дошли до архитектуры Google NMT
- Но в 2017 году оказалось, что всё может быть ещё проще и интереснее
- Google: «Attention is all you need» (Vaswani et al., 2017)
- Основная идея – self-attention; оказывается очень плодотворной для всевозможных seq2seq задач
- Главная мотивация – попробовать всё-таки уйти от кодирования вектором постоянной длины

Transformer

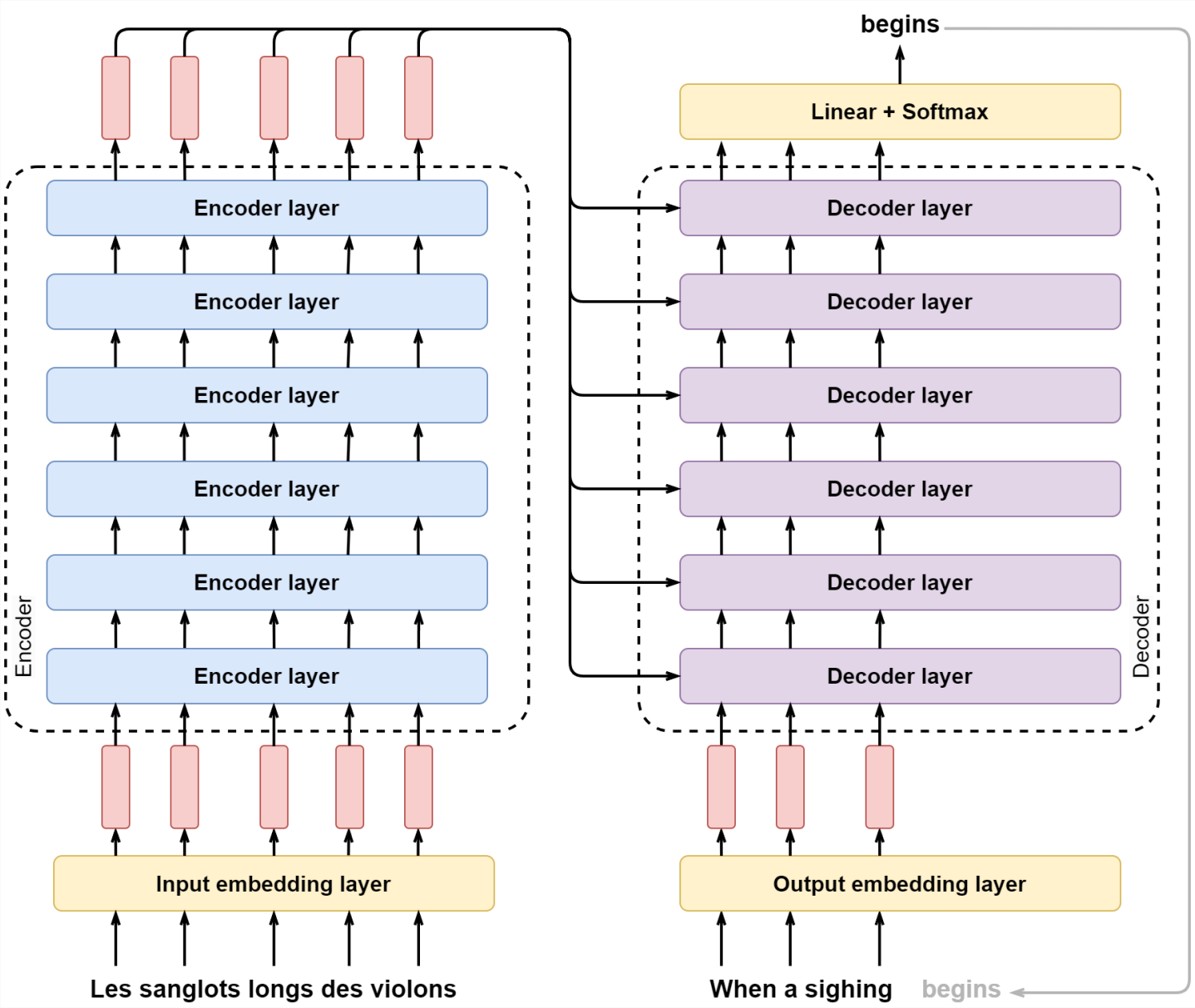
- Общая схема:



- Теперь подробнее...

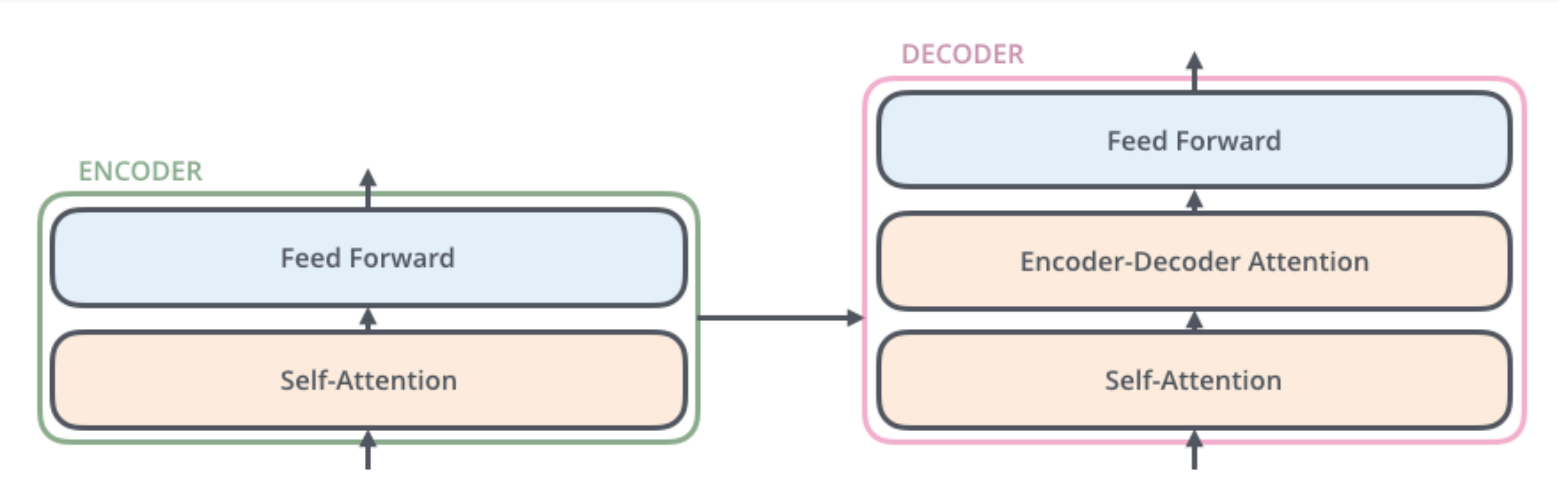
Transformer

- Суть, как и раньше, – encoder-decoder:



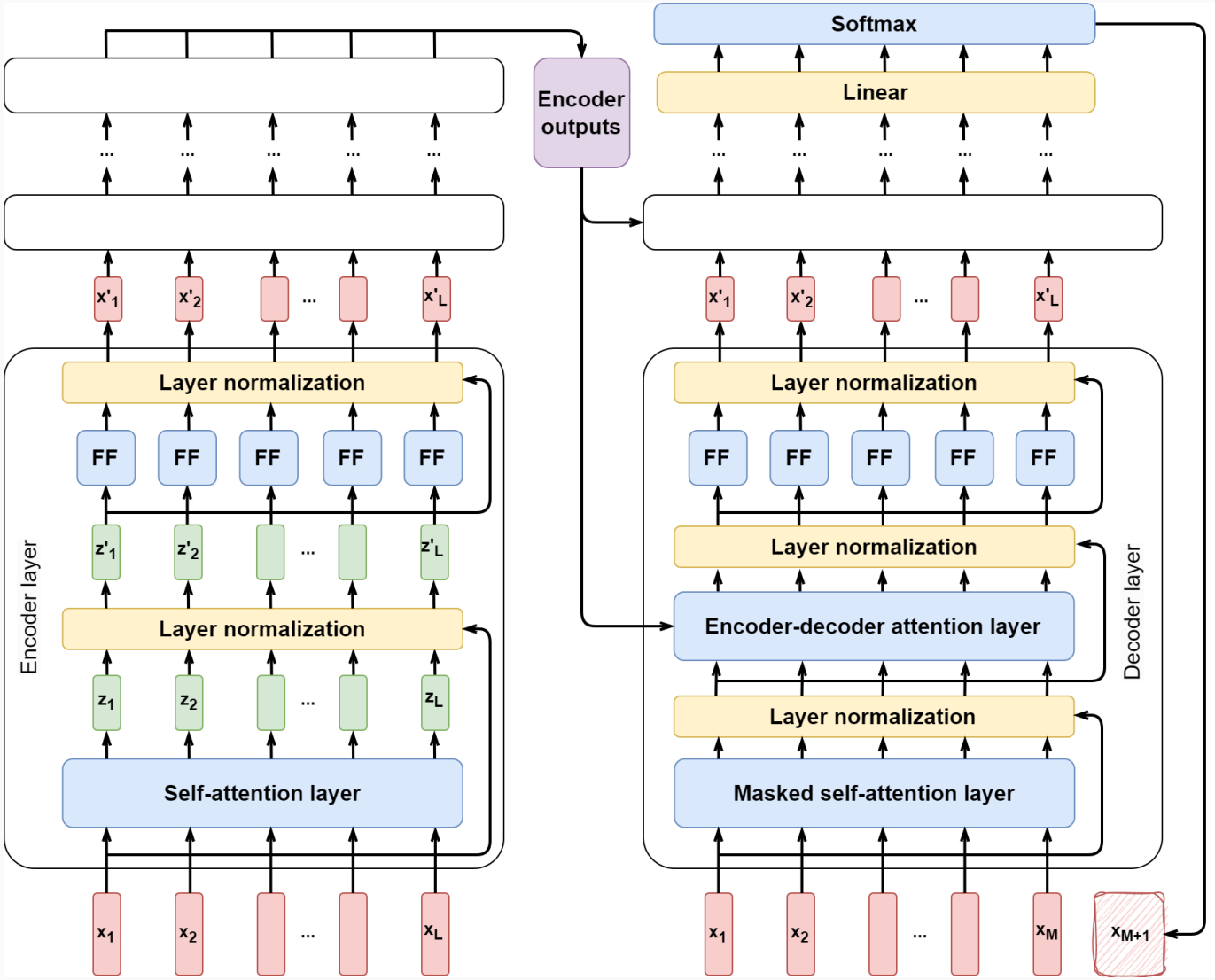
Transformer

- В каждом слое – слой self-attention, а потом feedforward layer, который независимо применяется к каждой позиции входа. У декодера ещё есть attention между ними:



Transformer

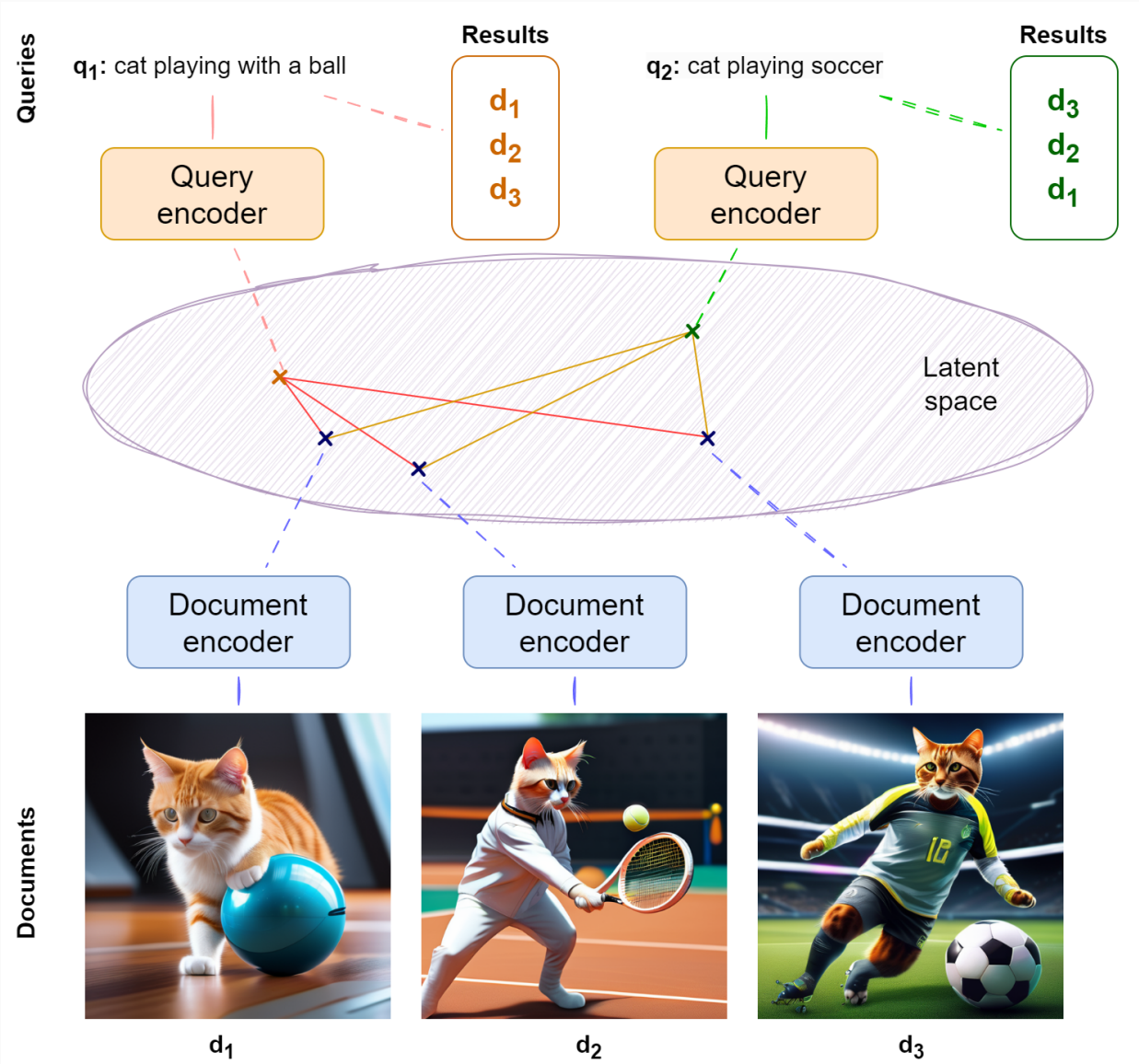
- Слова, естественно, представляются векторами, в feedforward слое всё параллельно:



- Но что же это такое — self-attention?

Self-attention

- Чтобы понять самовнимание (self-attention), начнём с информационного поиска (information retrieval)



Self-attention

- Запросы и документы отображаются в одно и то же латентное пространство (но разными кодировщиками — это могут быть вообще объекты разной природы)
- Запрос идёт через query encoder
- Документы (на иллюстрации это картинки) — через другой encoder, но в то же пространство
- Чтобы найти самые релевантные документы, мы ищем ближайших соседей в латентном пространстве среди документов; часто предполагают, что латентное пространство линейно, и расстояние там — это просто скалярное произведение $\mathrm{dist}(q, d) = \mathrm{Enc}_q(q)^\top \mathrm{Enc}_d(d)$.

Self-attention

- В самовнимании эта интуиция используется очень абстрактно. Слой получает на вход последовательность векторов $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_L$, т.е. матрицу $X \in \mathbb{R}^{d \times L}$.
- Запросы, ключи и документы берутся из самих $\mathbf{x}_i$:
  - умножая $\mathbf{x}_1$ на матрицу весов W^Q, мы получаем вектор *запроса* $\mathbf{q}_1 = W^Q \mathbf{x}_1$; обратите внимание, что размерность q векторов запросов, $\mathbf{q}_i \in \mathbb{R}^q$, может отличаться (обычно меньше) от входной размерности d, $\mathbf{x}_i \in \mathbb{R}^d$;
  - умножая $\mathbf{x}_i$ на матрицу весов W^K, мы получаем векторы *ключей* $\mathbf{k}_i = W^K \mathbf{x}_i$ для $i = 1, \dots, L$; поскольку мы хотим, чтобы запросы и ключи находились в одном латентном пространстве, ключи имеют ту же размерность, что и запросы, $\mathbf{k}_i \in \mathbb{R}^q$, таким образом $W^V \in \mathbb{R}^{q \times d}$;
  - третья матрица весов W^V даёт векторы *значений* $\mathbf{v}_i = W^V \mathbf{x}_i$ для $i = 1, \dots, L$; это те *документы*, которые мы будем «извлекать» с помощью ключей $\mathbf{k}_i$; формально мы имеем другую размерность v для значений, $\mathbf{v}_i \in \mathbb{R}^v$ и $W^V \in \mathbb{R}^{v \times d}$; на практике обычно $v = q$.

Self-attention

- Затем мы выполняем поиск, вычисляя оценки внимания как скалярные произведения между запросами и документами
- Нормализуем $\mathbf{q}_i^\top \mathbf{v}_j$, деля на $\sqrt{q}$, и получить оценки через softmax, чтобы преобразовать их в вероятности:

$$\alpha_{ij} = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right)_j,$$

где $K \in \mathbb{R}^{q \times L}$ — это все ключи, объединенные в матрицу, $K = W^K X$.

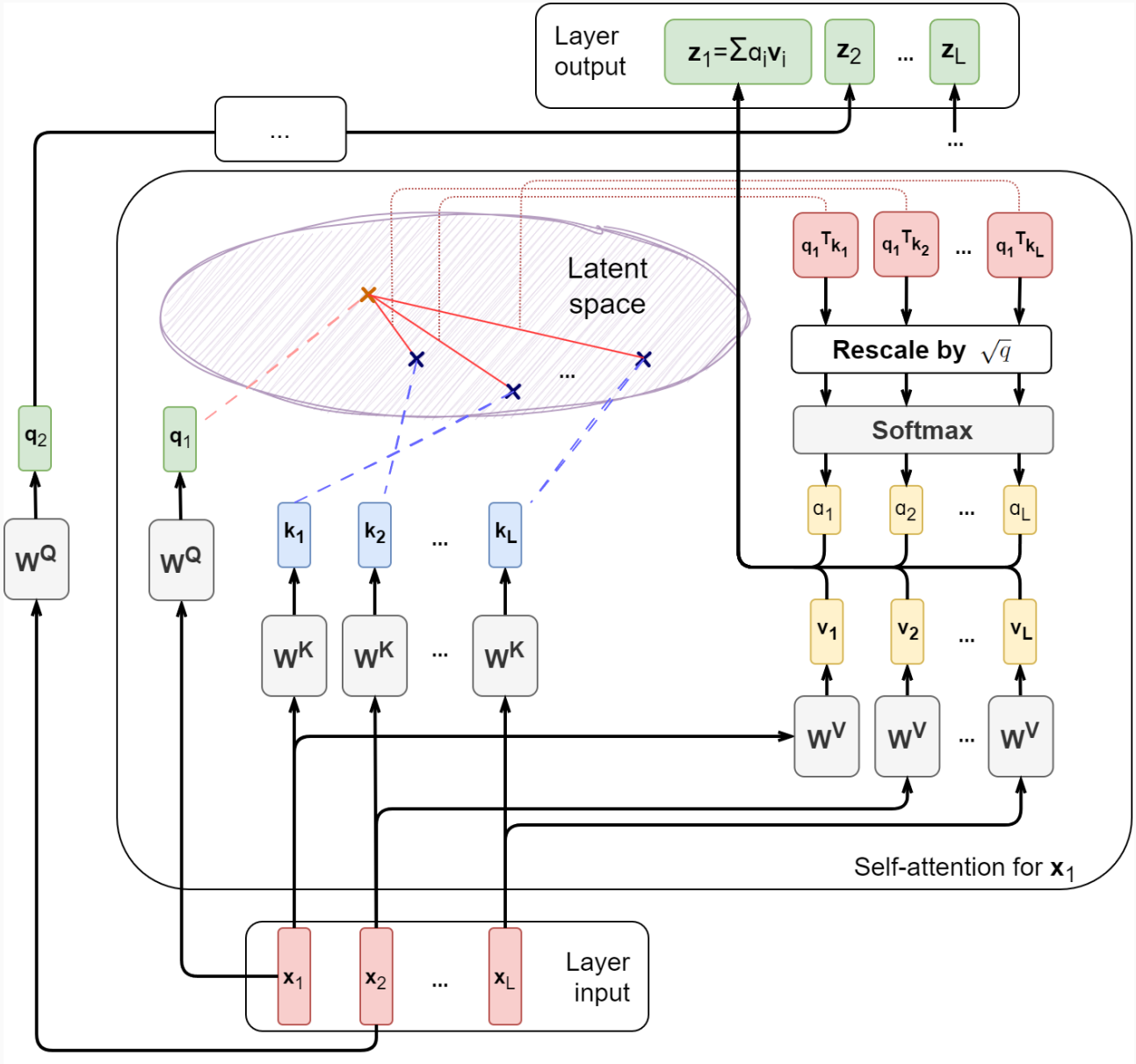
- Затем используем результат в качестве коэффициентов для выпуклой комбинации *значений* $\mathbf{v}_j$:

$$\mathbf{z}_i = \mathrm{softmax}\left(\frac{1}{\sqrt{q}}\mathbf{q}_i K^\top\right) V,$$

где $V \in \mathbb{R}^{q \times L}$ — значения (values), $V = W^V X$.

Self-attention

- Общая структура слоя самовнимания (self-attention):



Self-attention

- И это вся интуиция! Мы можем объединить вычисления каждого $\mathbf{z}_i$ в одну формулу в матричном виде:

$$Z = \mathrm{softmax}\left(\frac{1}{\sqrt{q}} QK^\top\right) V.$$

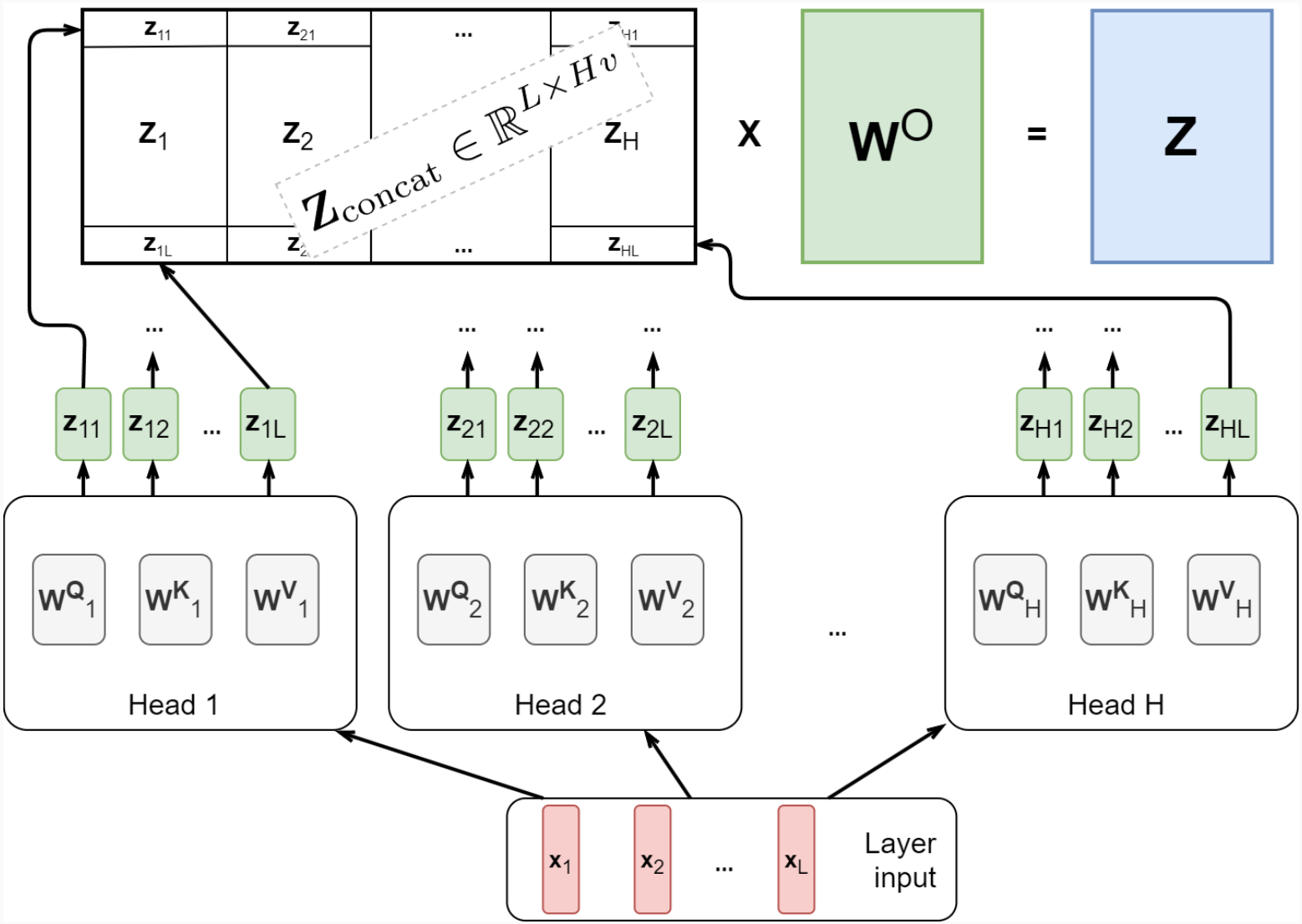
- Но это только один способ «смотреть» на входные векторы; то, что мы сейчас определили, — это не полный слой самовнимания, а только одна *голова* (self-attention head)
- Можно распараллелить эти вычисления вдоль H различных голов, используя разные матрицы весов $W_1^Q, \dots W_H^Q$, $W_1^K, \dots W_H^K$ и $W_1^V, \dots W_H^V$. Это *многоголовое внимание* (multi-head attention)

Self-attention

- После H параллельных голов мы получаем матрицы выходов $Z_1, Z_2, \ldots, Z_H$ размерности $v \times L$, объединяем их все в $Z_{\text{concat}} \in \mathbb{R}^{L \times Hv}$ и добавляем ещё одну матрицу весов $W^O \in \mathbb{R}^{Hv \times d}$, чтобы сжать результат обратно до необходимой размерности:
$$Z = \left(Z_{\text{concat}} W^O\right)^\top.$$
- С помощью W^O слой самовнимания может комбинировать представления, полученные из различных голов внимания; это также важный способ добавить гибкость и выразительность в архитектуру.

Self-attention

- Многоголовое внимание:



- Это самая важная часть, но это ещё не всё.

Декодер

- Декодер включает два дополнительных типа слоёв: *маскированное самовнимание* (masked self-attention) и *encoder-decoder attention*
- Маскированное внимание в основном означает, что, поскольку декодер работает автокоррективно, он не должен смотреть на токены, которые ещё не породил
- Для этого проще всего ввести всю последовательность и замаскировать будущие позиции внутри слоёв самовнимания
- Формально это означает, что мы задаем аргументы softmax как $-\infty$ для будущих токенов, что означает, что их веса внимания всегда будут нулевыми

Декодер

- Encoder-decoder attention — это вариация механизма самовнимания, которая опирается на результаты кодировщика
- Звучит сложно... но на самом деле это почти тривиальное изменение! Мы используем разные векторы в качестве входных данных в том же самовнимании:
  - для создания запросов используем векторы из предыдущего слоя, то есть текущие представления уже порождённых выходных токенов;
  - но для «документов» в задаче «поиска», то есть для векторов ключей и значений, мы используем векторы из декодера
- Неформально это значит, что мы выполняем «поиск» на выходе энкодера с запросами, состоящими из уже порождённых токенов
- Формально все размерности хорошо совпадают: в аргументе softmax есть L элементов для каждого вектора, но число запросов и, следовательно, число выходов совпадает с числом входов

Токенизация

- Мы используем вложения (embeddings), но как мы делим текст на токены?
- *Byte-pair encoding*: интересная идея, основанная на кодировании Хаффмана
- Давайте начнем с построения словаря:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4

Токенизация

- Затем разделим его на символы и подсчитаем пары символов:

$$\{\text{cat} : 10, \text{pet} : 12, \text{mat} : 5, \text{rat} : 8, \text{eats} : 4\},$$

$$\{c : 10, a : 27, t : 39, p : 12, e : 16, m : 5, r : 8, s : 4\},$$

$$\{ca : 10, at : 27, pe : 12, et : 12, ma : 5, ra : 8, ea : 4, ts : 4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4

Токенизация

- Теперь возьмем наиболее частую пару («*at*») и перекодируем её в новый символ (Z), который добавляется в словарь:

$$\{\mathrm{cZ}:10, \mathrm{pet}:12, \mathrm{mZ}:5, \mathrm{rZ}:8, \mathrm{eZs}:4\},$$
$$\{c:10, Z:27, t:12, p:12, e:16, m:5, r:8, s:4\},$$
$$\{cZ:10, pe:12, et:12, mZ:5, rZ:8, eZ:4, Zs:4\}.$$

Original words:	cat		pet		mat		rat		eats
Frequencies:	10		12		5		8		4
Characters:	c	a	t	p	e	m	r	s	
Frequencies:	10	27	39	12	16	5	8	4	
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts	
Frequencies:	10	**27**	12	12	5	8	4	4	

Токенизация

- Теперь мы можем выбрать новую наиболее частую пару — в данном случае *pe* или *et* — и заменить её на другой новый символ, например *Y*:

$$\{\mathrm{cZ}:10, \mathrm{pet}:12, \mathrm{mZ}:5, \mathrm{rZ}:8, \mathrm{eZs}:4\},$$
$$\{c:10, Z:27, t:12, p:12, e:16, m:5, r:8, s:4\},$$
$$\{cZ:10, pe:12, et:12, mZ:5, rZ:8, eZ:4, Zs:4\}.$$

Original words:	cat		pet		mat		rat	eats
Frequencies:	10		12		5		8	4
Characters:	c	a	t	p	e	m	r	s
Frequencies:	10	27	39	12	16	5	8	4
Pairs:	ca	**at**	pe	et	ma	ra	ea	ts
Frequencies:	10	**27**	12	12	5	8	4	4

Positional encodings

- В трансформере каждый входной токен может “посмотреть” на любой другой *напрямую*, через любое число других токенов
- Это большое преимущество по сравнению с RNN!
- Но это значит, что у нас фиксированное ограниченное *context window* и, главное, поскольку веса внимания покрывают все токены равномерно, мы *теряем последовательность*: слой self-attention “не знает”, что некоторые токены стоят рядом, а некоторые далеко друг от друга
- Нужно как-то дать понять трансформеру, что у последовательности есть порядок, и это делается через *positional encodings*

Positional encodings

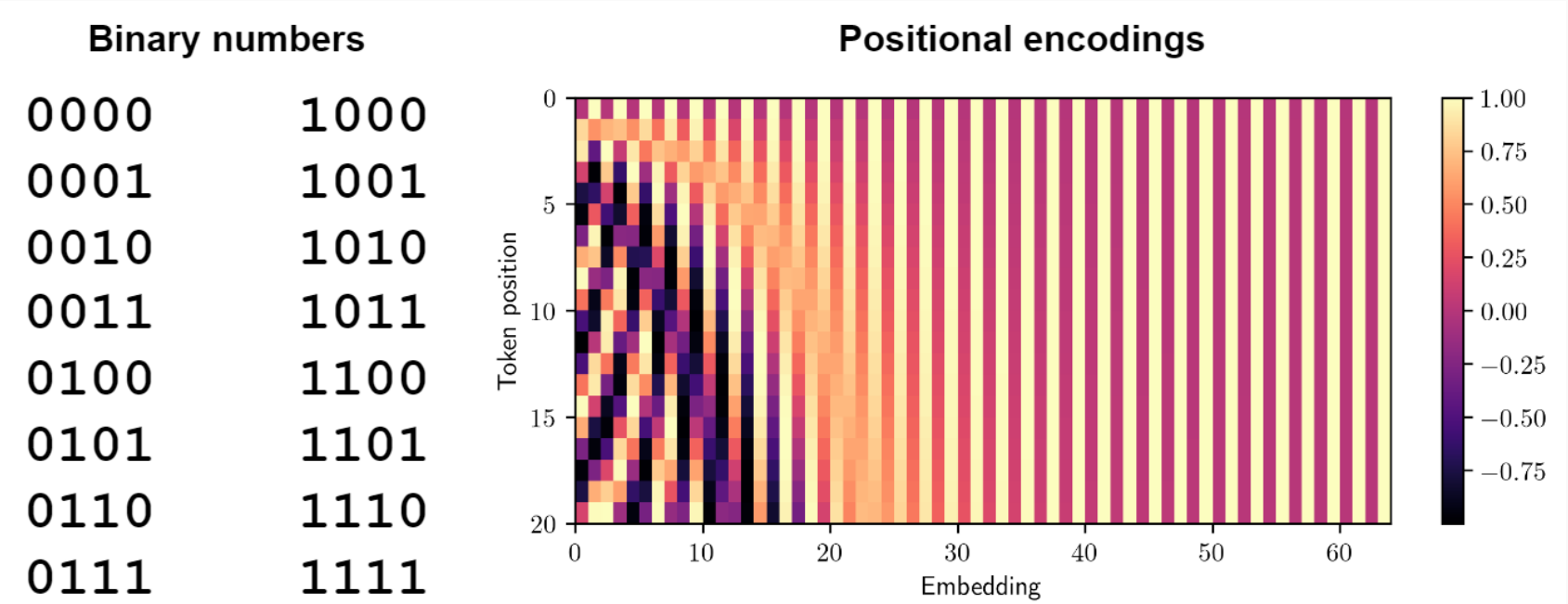
- Positional encodings — векторы, добавляемые ко вложениям, которые отражают порядковый номер токена
- Как его закодировать? Можно просто число увеличивать, но это не так просто:
  - если использовать увеличивающуюся последовательность вроде $1, 2, 3, \ldots$, она не обобщится на последовательности длиннее, чем часто бывало в датасете, да и вообще поведение будет плохо определено для больших значений;
  - если уложить в интервал вроде $[0, 1]$, то positional encoding не будет знать, сколько на самом деле слов между токенами: расстояние от 0 до $\frac{1}{2}$ может быть 1 токен или 100.

Positional encodings

- Так что мы используем нечто вроде позиционной системы счисления, только непрерывную:

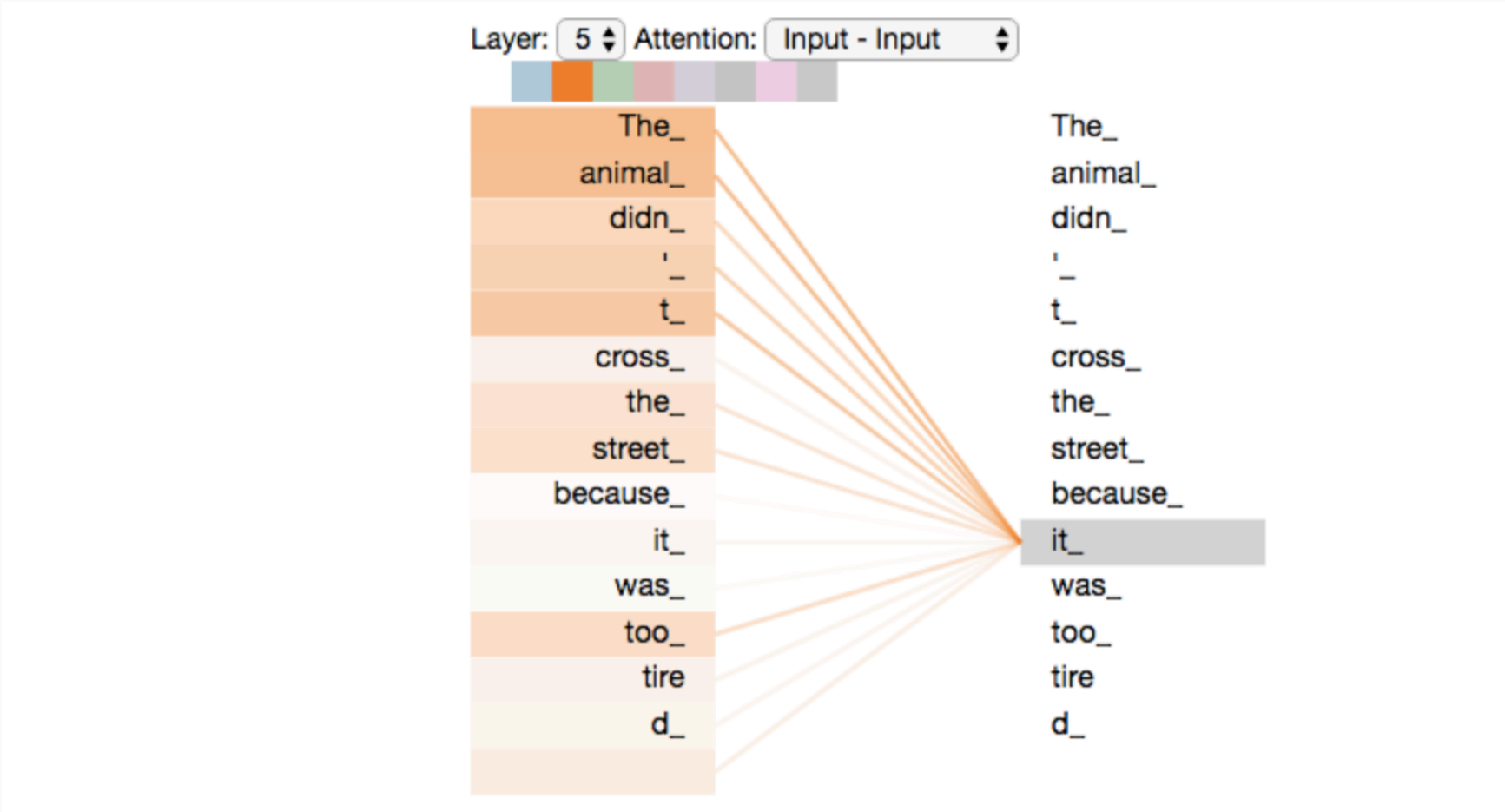
$$\mathrm{PE}\left(\mathrm{pos}, 2i\right) = \sin\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right), \quad \mathrm{PE}\left(\mathrm{pos}, 2i+1\right) = \cos\left(\frac{\mathrm{pos}}{10000^{2i/d}}\right),$$

где pos — позиция токена, d — размерность вложения.



Результаты

- В результате трансформер обучает веса того, как одни слова ”участвуют в обработке” других слов:



Результаты

- Бонус от self-attention – во-первых, вычислительный, во-вторых, сокращает пути между словами, в-третьих, потенциальная интерпретируемость:

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

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

Результаты

- Работает лучше, обучается в сто раз быстрее:

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

Спасибо за внимание!