

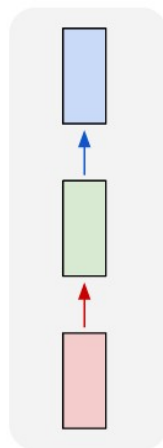


# Механизм внимания. Трансформеры

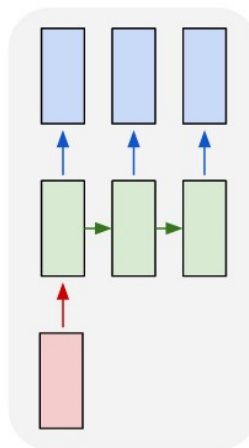
# RNN tasks

Какие виды задач можно решать?

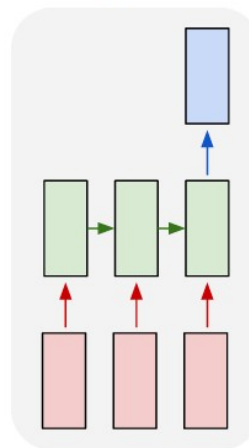
one to one



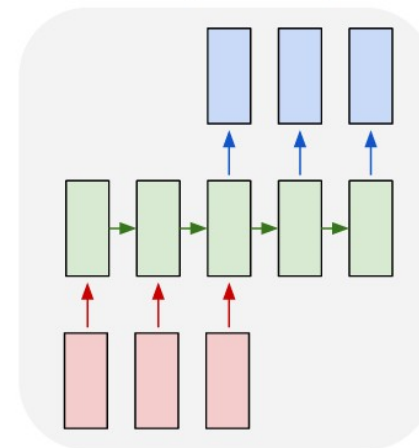
one to many



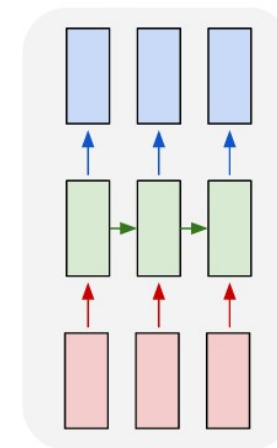
many to one



many to many



many to many



**One to one: стандартные задачи для MLP**

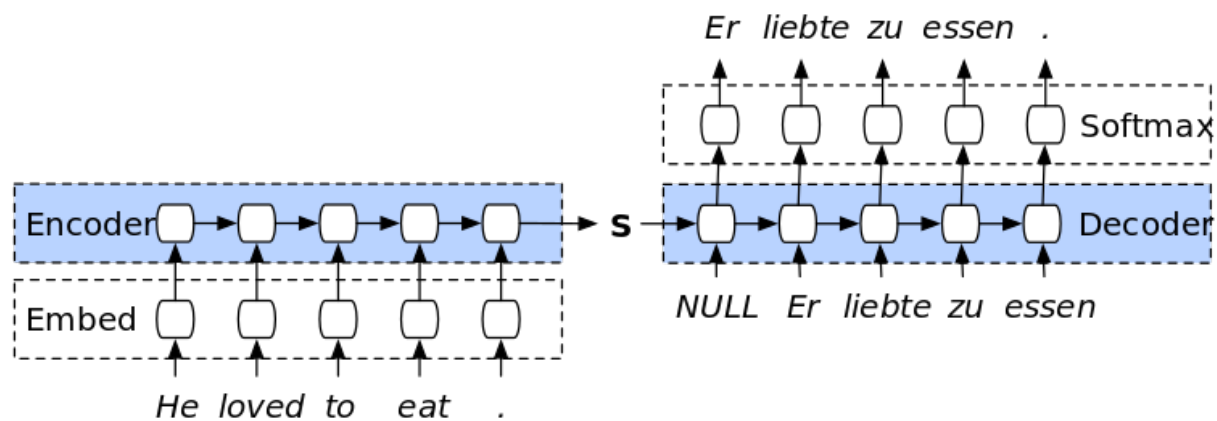
**One to many: image description generation by image**

**Many to one: text classification**

**Many to many: machine translation, POS-tagging**

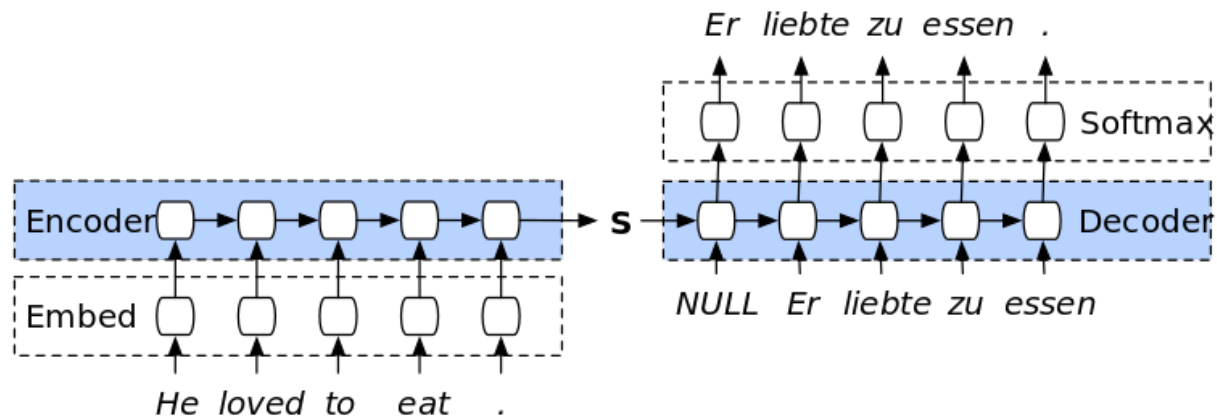
# Seq2Seq

Задача машинного перевода: в чем здесь недостаток?



# Seq2Seq

- Нужно сжать весь текст в один вектор
- Теряется информация о первых словах
- Декодер тоже может терять информацию по мере генерации последовательности
- Можно использовать BiLSTM, но тогда будет теряться информация о словах в середине
- И непонятно, как им декодировать





# Возможное решение

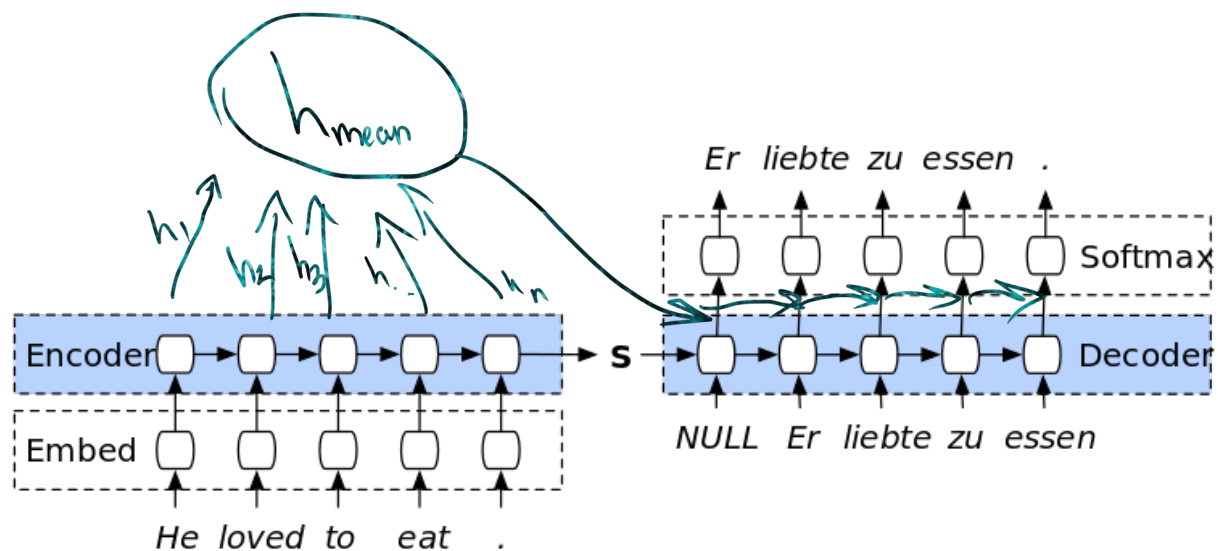
Во время генерации каждого следующего слова будем  
смотреть на *всю* входную последовательность

А как это сделать?

# Возможное решение

Во время генерации каждого следующего слова будем смотреть на *всю* входную последовательность

Усредним все скрытые состояния





# Возможное решение

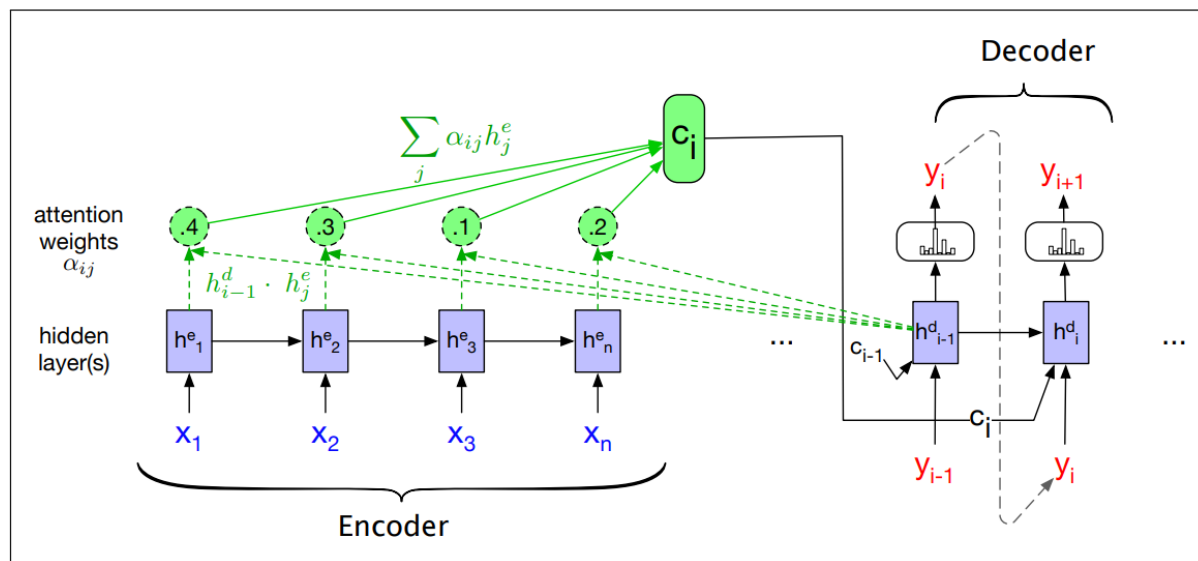
**Во время генерации каждого следующего слова будем смотреть на *всю* входную последовательность**

**Усредним все скрытые состояния?**

**Недостаток: все слова будут влиять на каждое слово перевода одинаково**

# Механизм внимания

- Нам нужна взвешенная сумма всех скрытых состояний!
- А значит, там рисуется какая-то линейная модель
- На каждом шаге эти веса должны будут пересчитываться





# Механизм внимания

- Вектор  $c$  – функция скрытых состояний энкодера:  $c = f(h_1^e, \dots, h_n^e)$ .

- Еще жутких формул:  $h_i^d = g(\hat{y}_{i-1}, h_{i-1}^d, c_i)$

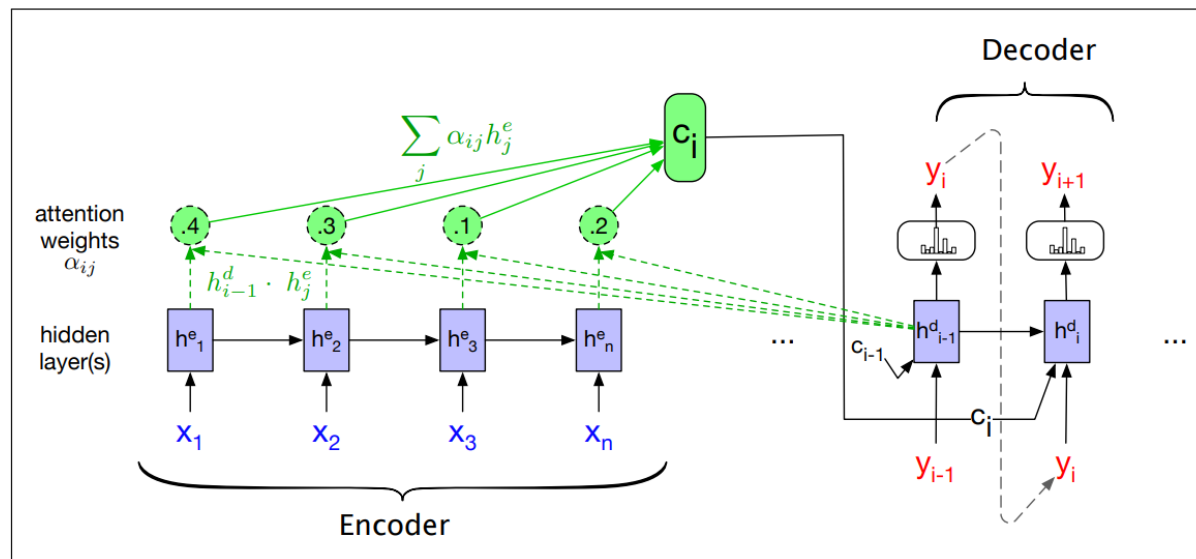
Текущее  
скрытое  
состояние  
декодера

пред.  
выход  
декодера

пред.  
скрытое  
сост.  
декодера

кон.  
контекстный  
вектор

- Как же вычислить  $c_i$ ?



# Механизм внимания

- Надо сперва вычислить, насколько релевантно каждое  $h_i^e$  текущему  $h_i^d$
- Самое простое – косинусное расстояние
- $score(h_{i-1}^d, h_j^e) = h_{i-1}^d \cdot h_j^e \leftarrow \text{это скаляр!}$
- Для текущего скрытого состояния декодера можем посчитать вектор косинусных расстояний со всеми скрытыми состояниями энкодера
- И потом от этого взять softmax, чтобы превратить в вероятности:
$$\alpha_{ij} = softmax(score(h_{i-1}^d, h_j^e) \forall j \in e) = \frac{exp(score(h_{i-1}^d, h_j^e))}{\sum_k exp(score(h_{i-1}^d, h_k^e))}$$
- Так мы получим веса: теперь можно их перемножить с исходными скрытыми состояниями  $c_i = \sum_j \alpha_{ij} h_j^e$

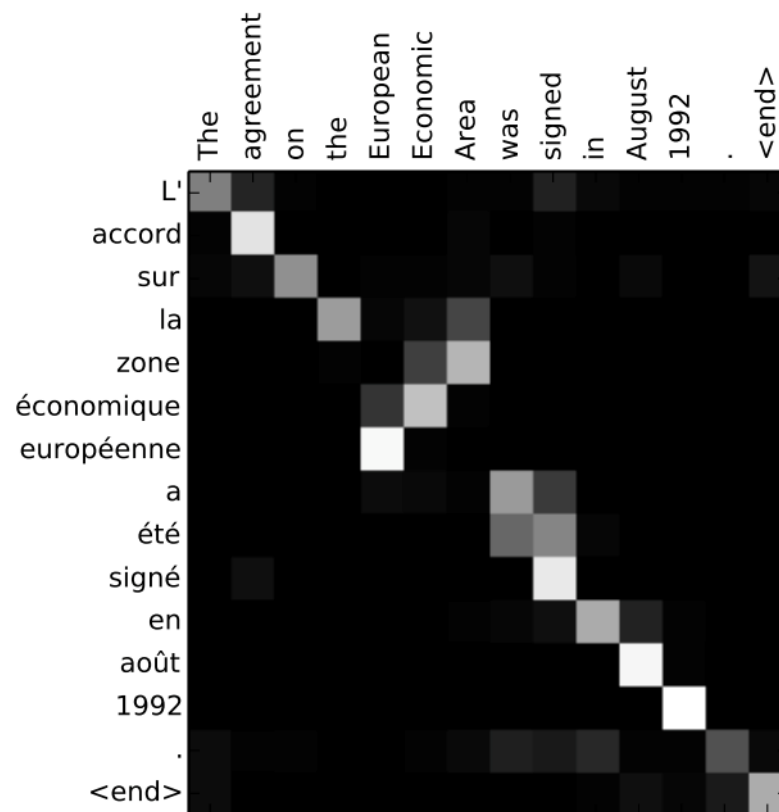




# Механизм внимания

- dot product – это самый простой способ вычислить внимание (без обучаемых параметров!)
- но можно сделать похитрее и вместо косинусного расстояния добавить свою линейную функцию с обучаемыми весами:
$$score(h_{i-1}^d, h_j^e) = h_{i-1}^d W_s h_j^e$$
- этот способ позволяет декодеру и энкодеру иметь вектора различающейся размерности

## A decorative graphic on the right side of the page featuring stylized leaves. The leaves are outlined in green, orange, and blue, with some filled in solid colors. They are arranged in a layered, overlapping fashion, creating a vibrant, naturalistic feel.





# Языковые модели



# Понятие языковой модели

Языковая модель – модель вероятностного распределения слов естественного языка; языковая модель использует алгоритмы МО для того, чтобы предсказывать наиболее вероятное следующее слово в предложении на основании предыдущих слов.



# Понятие языковой модели

Мы с вами уже знаем:

- Модели на n-грамах (с использованием Марковских цепей)
- BoW и skip-gram: они лежат в основе word2vec

Статические эмбединги: word2vec, fasttext, GloVE

Есть еще контекстные эмбединги. А почему, кстати, все вышеназванные – не контекстные?



# ELMo и контекстные эмбединги

- word2vec учим решать задачи BoW & skip-gram на больших датасетах
- а если взять LSTM? А лучше Bi-LSTM
- и обучить сетку решать те же самые задачи...
- ELMo (Embeddings from Language Models) – фреймворк, созданный AllenNLP.
- ELMo – Bi-LSTM, которую учили предсказывать следующее слово в предложении





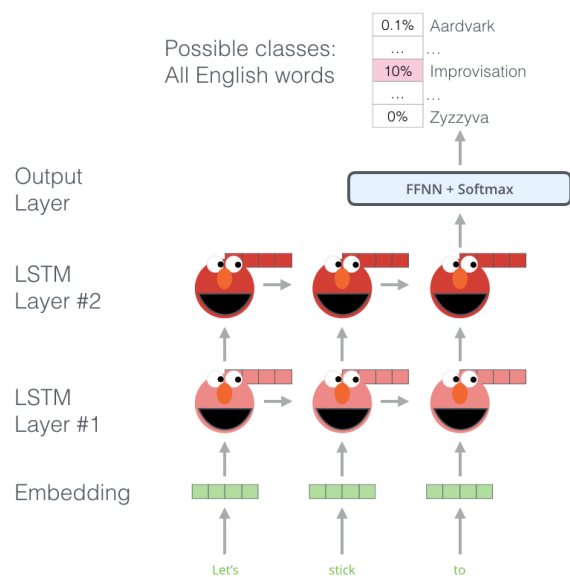
# ELMo и контекстные эмбединги

- ELMo – модель, которая учитывает контекст слова
- Поэтому и эмбединги называются contextualized
- elmo – подмодуль allennlp

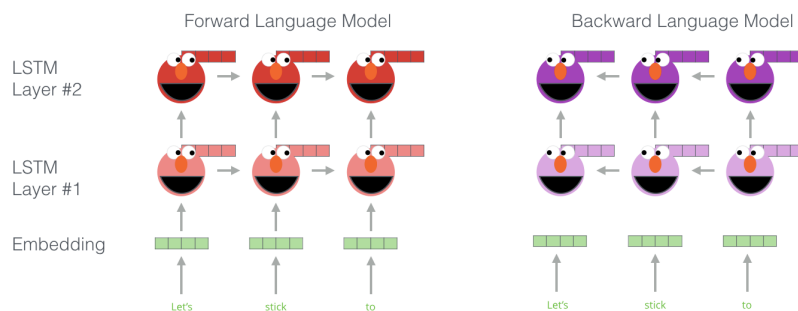


# ELMo и контекстные эмбединги

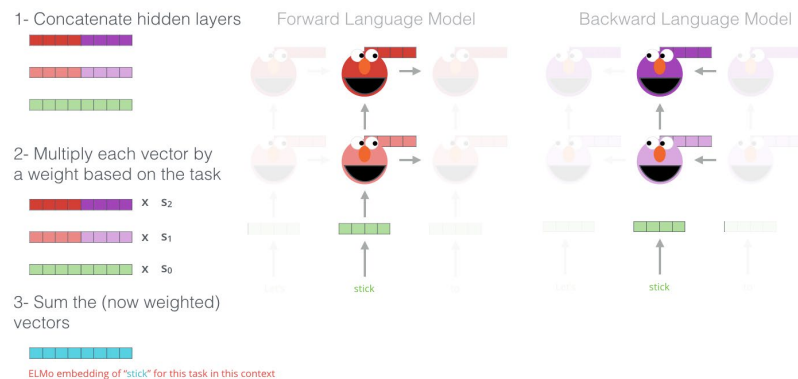
- Как работает?
- Подробнее



Embedding of "stick" in "Let's stick to" - Step #1



Embedding of "stick" in "Let's stick to" - Step #2



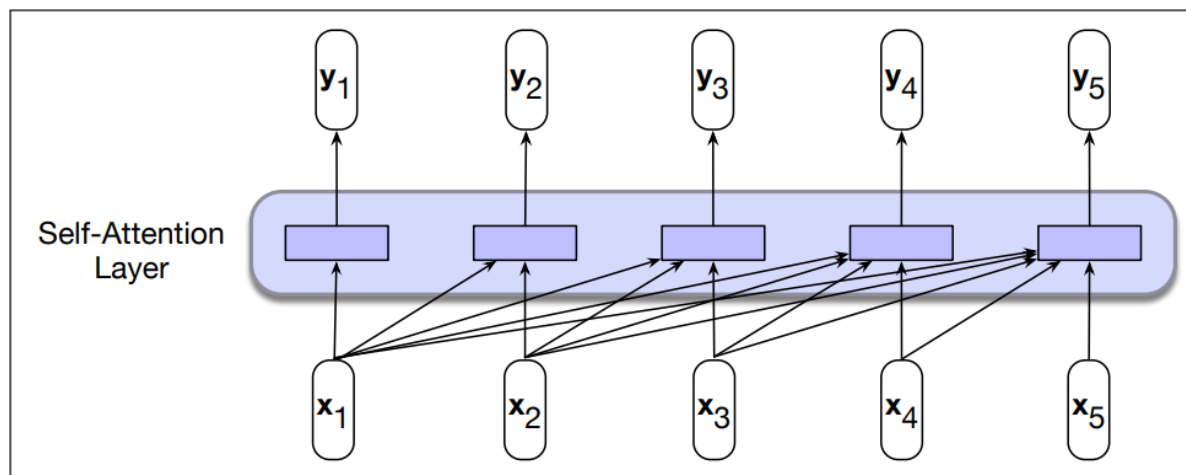


# Трансформеры



# Основная идея

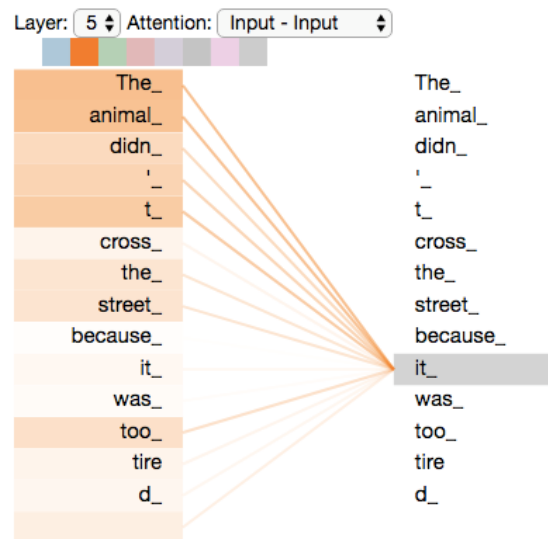
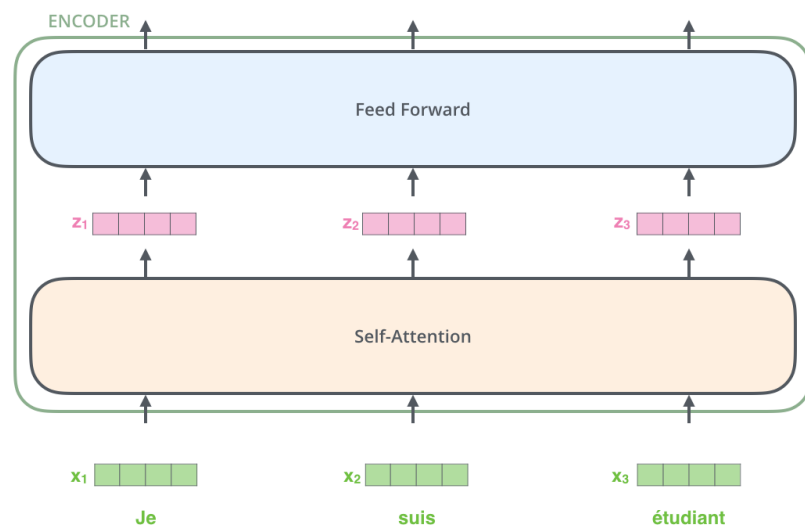
- А что, если механизм внимания применить к самому же энкодеру?..
- Возьмем предложение, рассчитаем взаимоотношения его слов и получим новые эмбединги для них



# Основная идея

Что нам это даст?

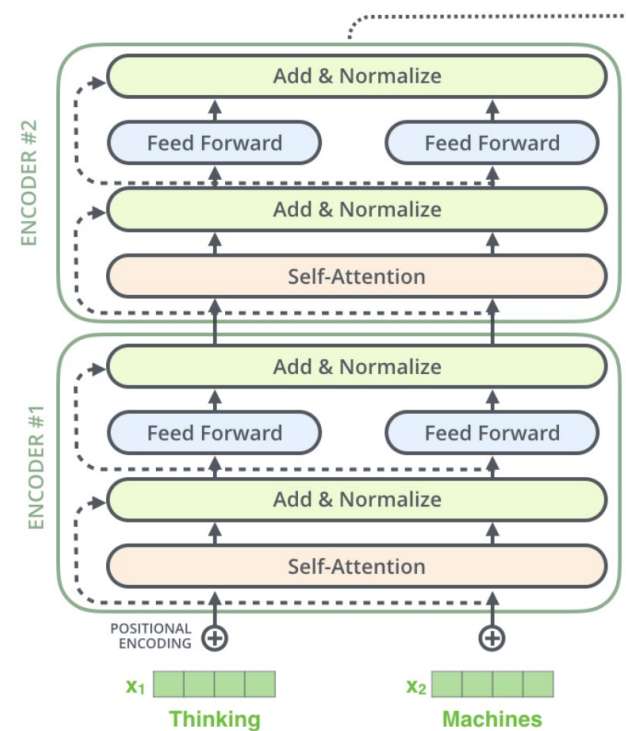
- мы можем обойтись без RNN и одновременно смотреть на все слова в инпуте
- (а это можно вычислительно распараллелить)
- можно будет установить связи между словами исходного предложения



# Self-attention

- Энкодеры можем наstackать
- Отлично, получим:

- А что это там внутри?



# Self-attention

**Будем для каждого слова  $x_j$  обучать три вектора:**

- Запрос (query) – текущий фокус внимания  $q_j = W_Q x_j$
- Ключ (key) – предыдущий инпут в сравнении с запросом  $k_j = W_K x_j$
- Значение (value) – для вычисления аутпута текущего фокуса  $v_j = W_V x_j$

**«Важность» слова  $x_i$  для слова  $x_j$ :  $\langle q_j, k_i \rangle$**

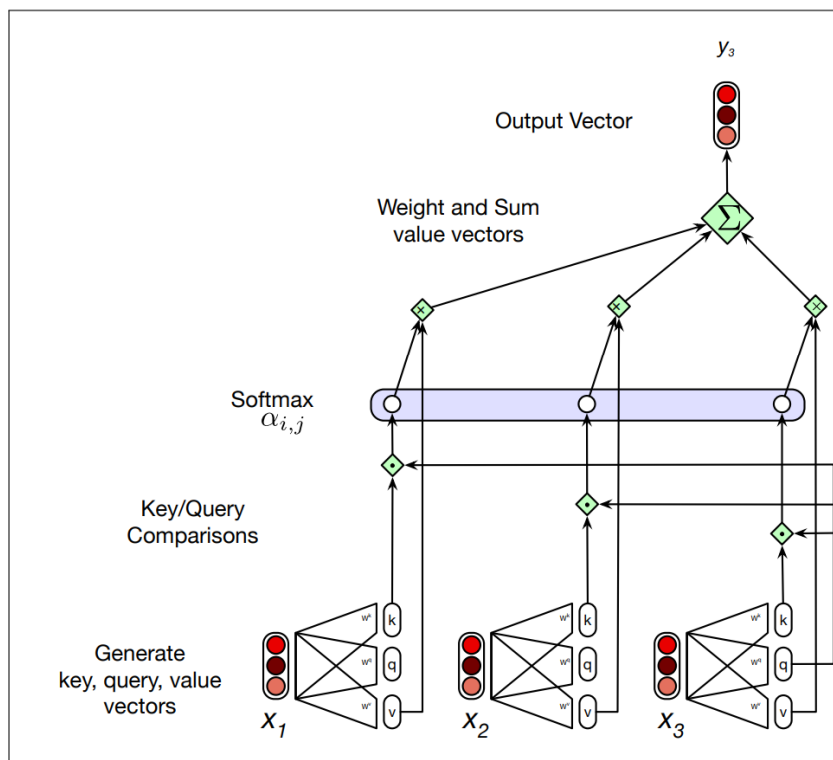


# Self-attention

Вычисляем новый эмбединг слова:

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

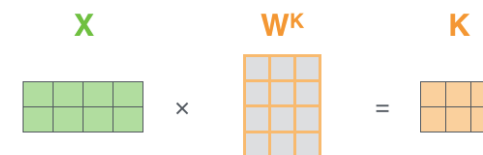
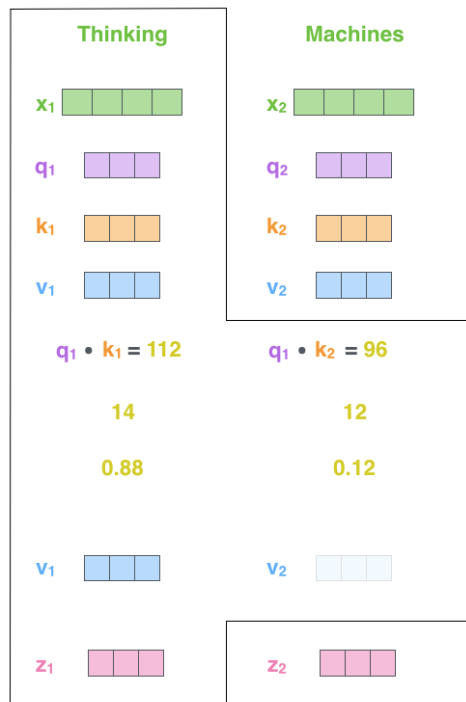
дм нормирование  
 $d_k$  - размер  
матрицы  
 $QK$



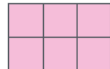


# То же в картинках

Input  
Embedding  
Queries  
Keys  
Values  
Score  
Divide by 8 (  $\sqrt{d_k}$  )  
Softmax  
Softmax  
X  
Value  
Sum



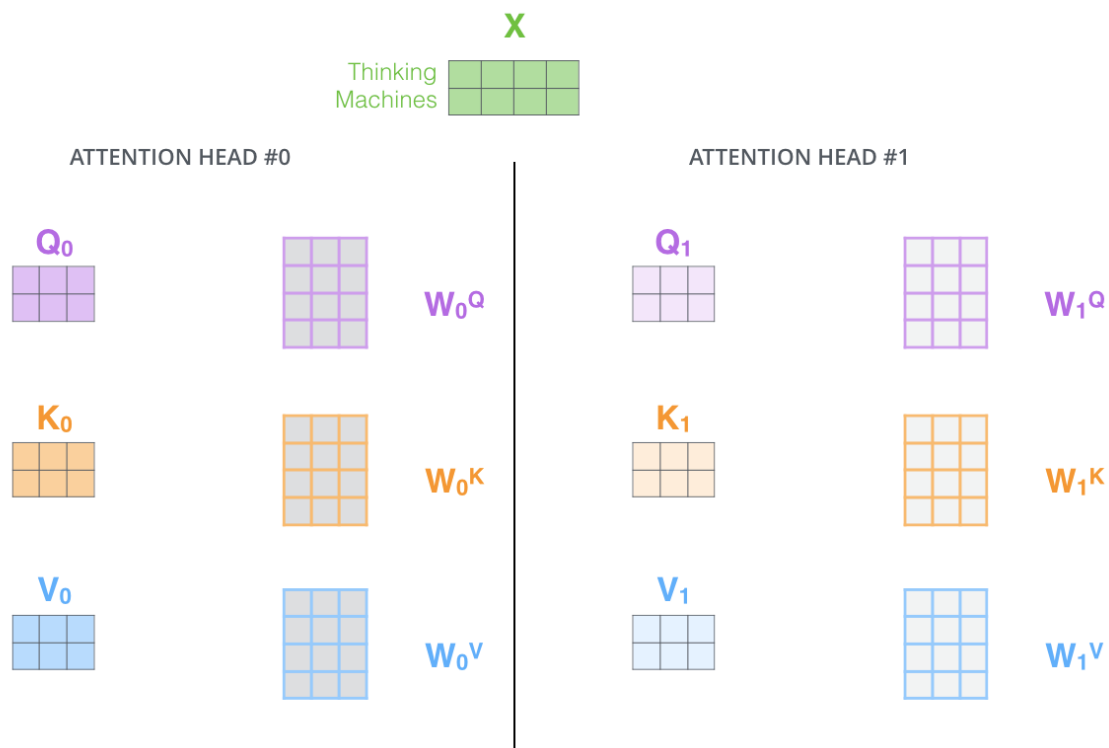
$$\text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) V$$

= 

# Multi-headed attention

Механизм работает очень хорошо

Давайте настакаем?



# Multi-headed attention

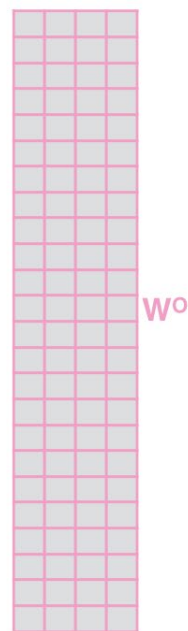
- Рассчитаем новые эмбединги для слов в энкодере, например, 8 раз
- А потом их сконкатенируем

1) Concatenate all the attention heads



2) Multiply with a weight matrix  $W^O$  that was trained jointly with the model

x

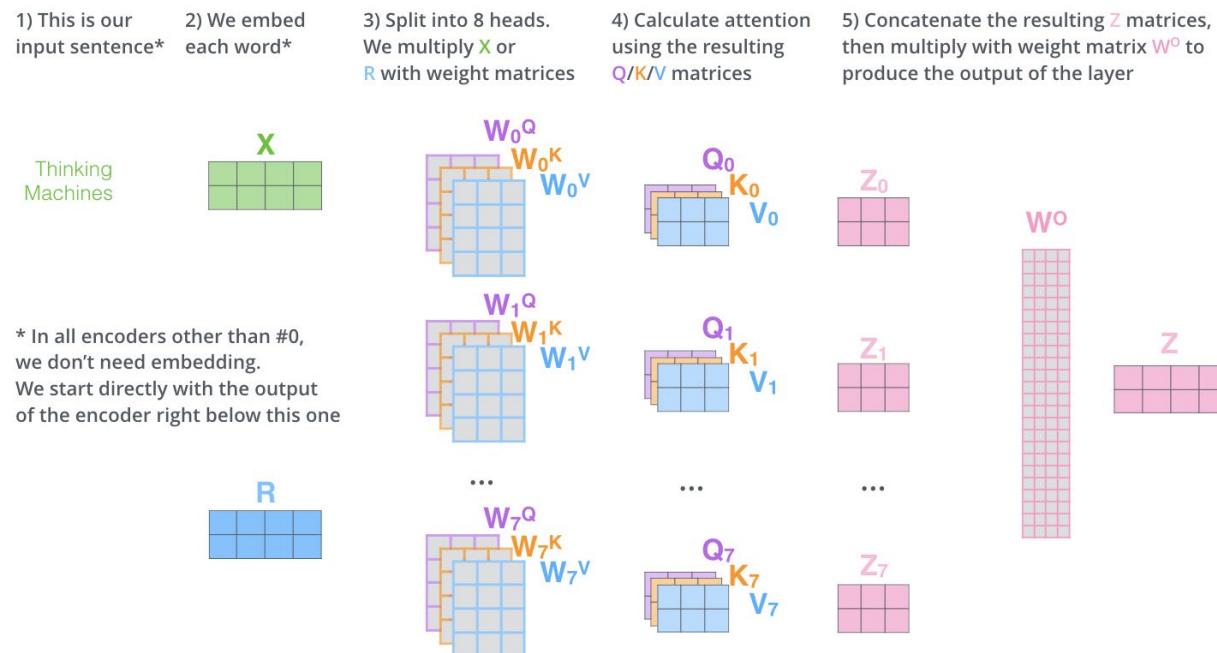


3) The result would be the  $Z$  matrix that captures information from all the attention heads. We can send this forward to the FFNN



# Multi-headed attention

- Все махинации на одной картинке:





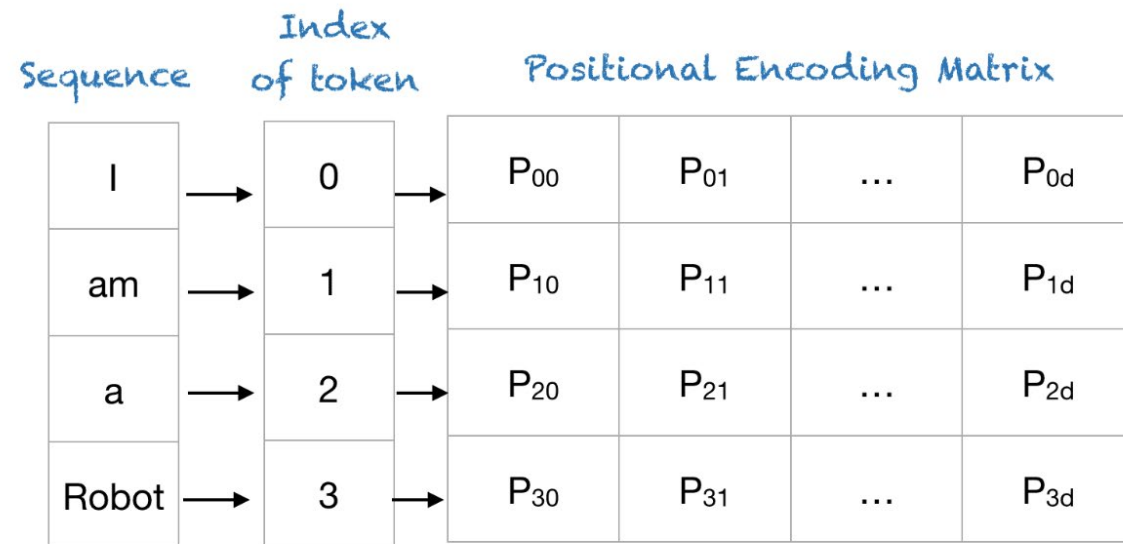
# Transformers vs RNN

**Чего-то еще не учили?**

**Как насчет порядка слов?**

# Positions

Давайте добавим закодированное положение слова в тексте.  
Это будут тоже вектора



Positional Encoding Matrix for the sequence 'I am a robot'

# Positions

Вычислять их будем с помощью тригонометрии

Sequence	Index of token, $k$	Positional Encoding Matrix with $d=4$ , $n=100$			
		$i=0$	$i=0$	$i=1$	$i=1$
I	0	$P_{00}=\sin(0)$ = 0	$P_{01}=\cos(0)$ = 1	$P_{02}=\sin(0)$ = 0	$P_{03}=\cos(0)$ = 1
am	1	$P_{10}=\sin(1/1)$ = 0.84	$P_{11}=\cos(1/1)$ = 0.54	$P_{12}=\sin(1/10)$ = 0.10	$P_{13}=\cos(1/10)$ = 1.0
a	2	$P_{20}=\sin(2/1)$ = 0.91	$P_{21}=\cos(2/1)$ = -0.42	$P_{22}=\sin(2/10)$ = 0.20	$P_{23}=\cos(2/10)$ = 0.98
Robot	3	$P_{30}=\sin(3/1)$ = 0.14	$P_{31}=\cos(3/1)$ = -0.99	$P_{32}=\sin(3/10)$ = 0.30	$P_{33}=\cos(3/10)$ = 0.96

Positional Encoding Matrix for the sequence 'I am a robot'

[подробное объяснение](#)

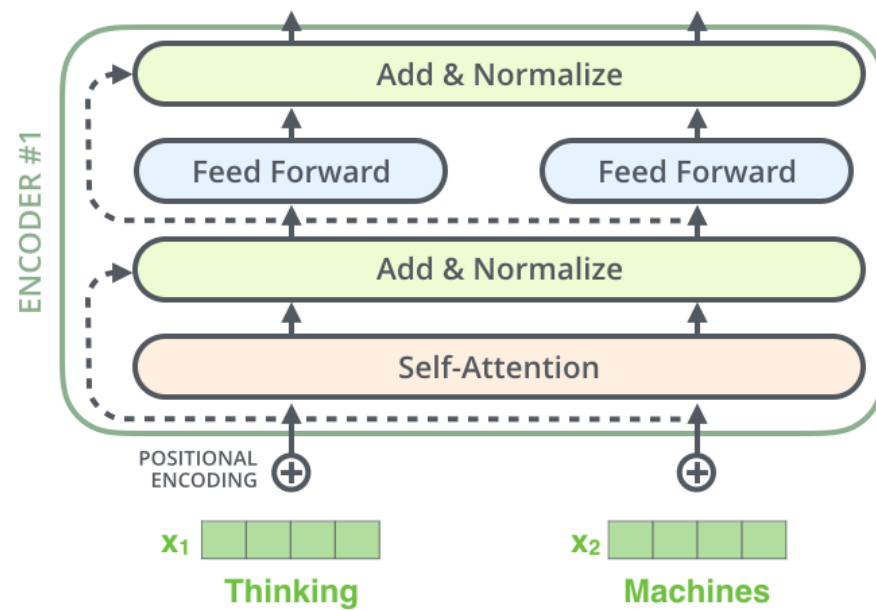
# Positional encodings

Будем их просто добавлять на старте:



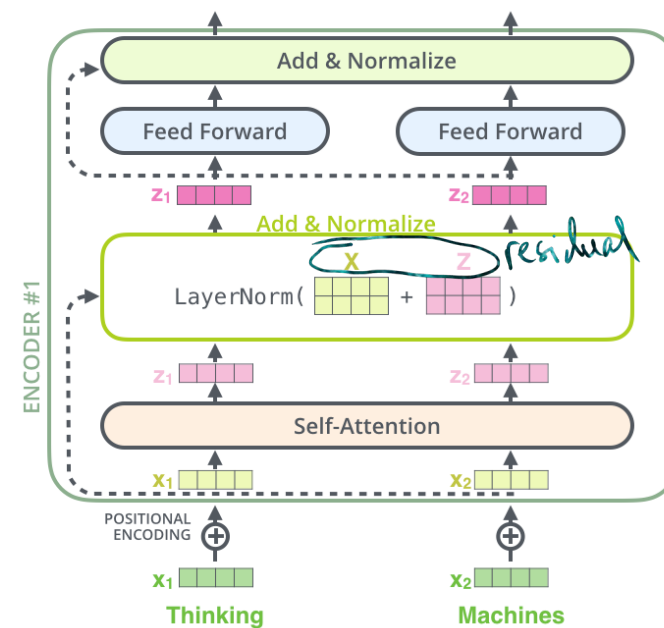


# Энкодер в трансформере



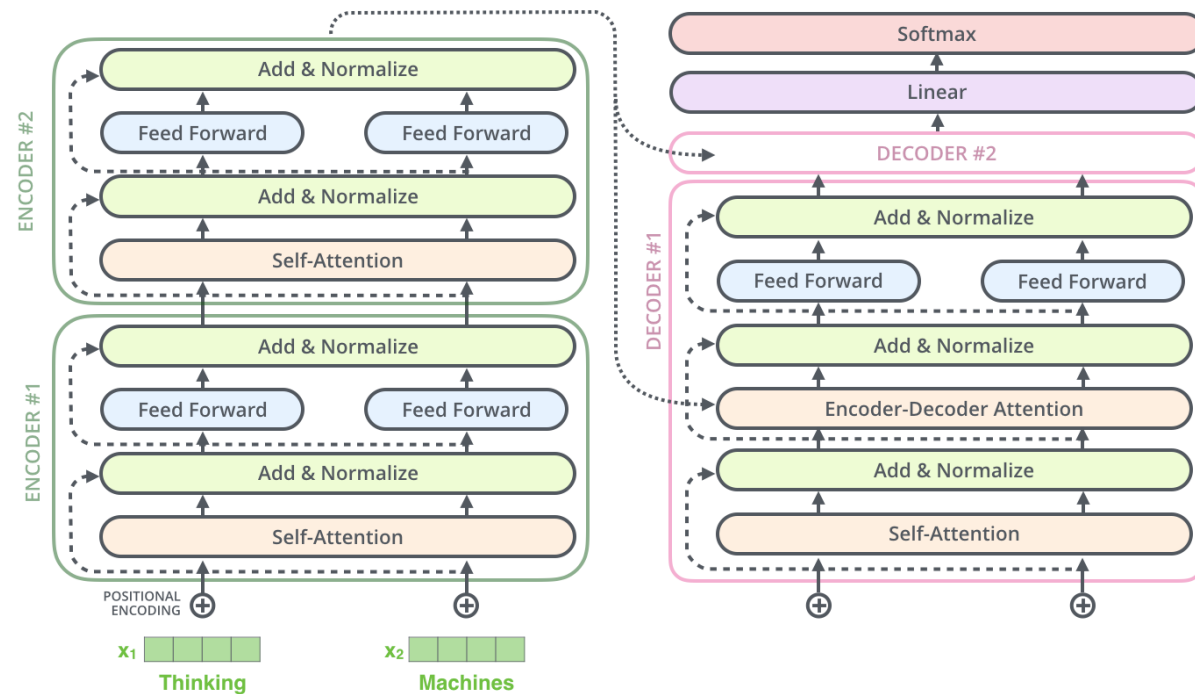
# Энкодер в трансформере

- Исходные эмбединги конкатенируем с позиционными
- Вычисляем новые эмбединги  $z$  (SelfAttention)
- Делаем skip-connections, чтобы градиенты лучше текли
- Layer Norm ??
- Просто разновидность BatchNorm ([статья](#))
- FF изменяет размерность как нам нужно



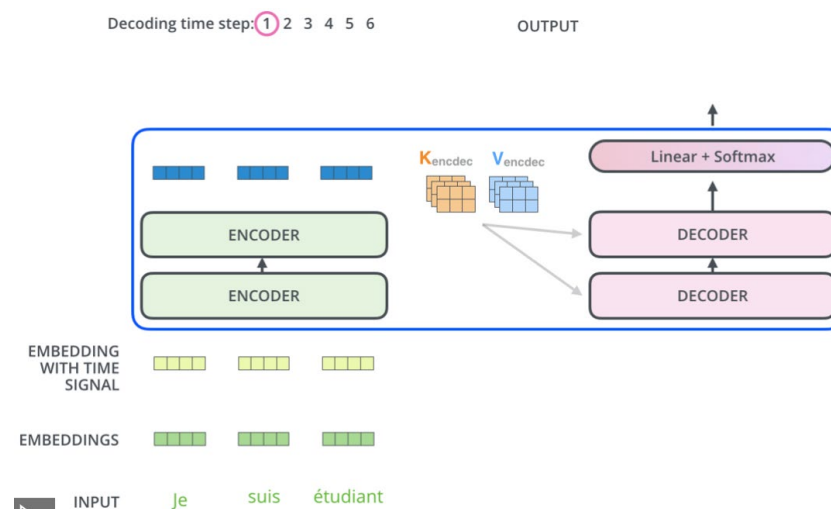
# Трансформер

Для некоторых задач достаточно энкодера, но у трансформера есть и декодер

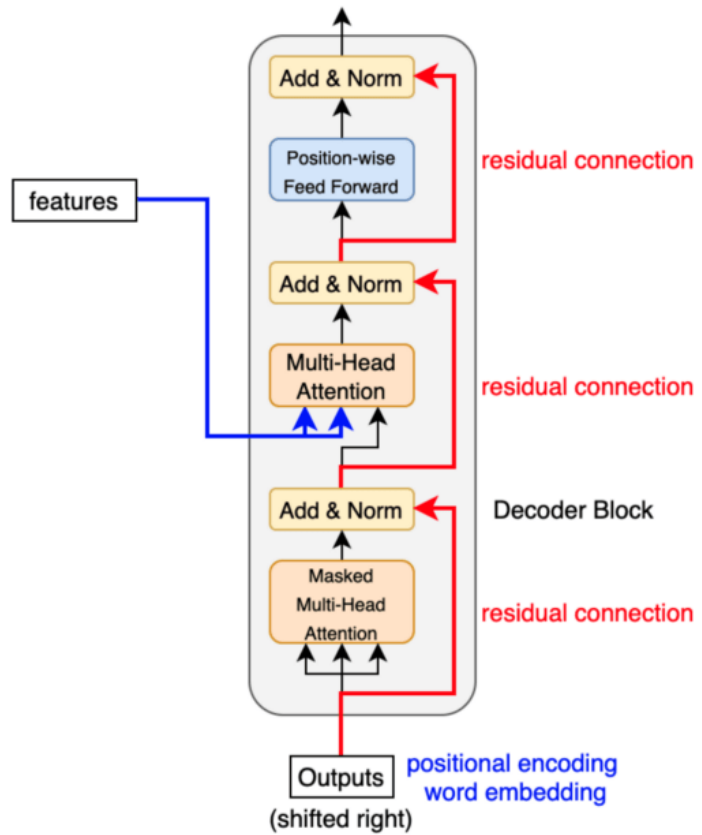


# Encoder-Decoder attention

- Энкодер обрабатывает инпут, получает эмбединги
- Их использует каждый из декодеров в своем слое encoder-decoder attention:
- Векторы  $k_j$  и  $v_j$  получаются домножением матриц  $K_{encdec}$  и  $V_{encdec}$  на выходы последнего энкодера
- Векторы  $q_j$  получаются стандартным образом из предыдущего слоя декодера



# Encoder-Decoder attention





# Masked self-attention

- В некоторых задачах нам нельзя смотреть на слова справа (для генерации текста)
- Декодер на них не должен смотреть
- Занулим их:

$$\begin{array}{ccc} a_1 & -\infty & -\infty \\ b_1 & b_2 & -\infty \\ c_1 & c_2 & c_3 \end{array}$$

- Софтмакс наши  $-\infty$  превратит в нули, а остальное в сумме будет давать единицу

# Задачи генерации и машинного перевода

- **Авторегрессионное применение:**
  - Сначала декодировщик выдаёт одно слово
  - Затем два (первое подаётся как вход)
  - Затем три (первые два подаются ему как вход)
  - И т.д.
- **Teacher forcing:**
  - каждый новый аутпут зависит от предыдущего
  - если сетка один раз облажалась, то все поедет...
  - альтернативы: Scheduled Sampling, Parallel Scheduled Sampling, Professor forcing, Beam Search
- **Greedy decoding:**
  - детерминированный аутпут (тупо выбираем самое вероятное слово)





# Важные ссылки

[Attention is All You Need](#)

[Martin, Jurafsky](#)

[Блог Аламмара](#)

[Seq2Seq and Attention](#)