



Deep Learning School

Transformer architecture

Encoder block

План занятия

- Идея архитектуры Transformer для машинного перевода
- Устройство Transformer Encoder
- Устройство Transformer Decoder
- QKV Attention

В этом видео

- **Идея архитектуры Transformer**
- **Устройство Transformer Encoder**
- Устройство Transformer Decoder
- QKV Attention

Недостатки RNN

Encoder на основе RNN очень медленный:

- Forward pass работает медленно, так как обрабатывает входящие токены последовательно;
- Backward pass работает медленно, так как происходит распространение градиентов сквозь время;

У RNN возникают проблемы во время обучения:

- Затухание/взрыв градиентов;
- Плохая утилизация GPU

Недостатки RNN

Нужны ли нам все еще RNN?

Мы использовать RNN для решения задачи машинного перевода, потому что их структура позволяет собирать информацию из входной последовательности и передавать ее декодеру.

Но сейчас у нас есть механизм Attention, который позволяет декодеру в любой момент времени “смотреть” на любую часть исходного предложения.

Идея Transformer

Transformer — архитектура, основа которой — механизм Attention.

У Transformer нет RNN-слоев.

Идея Transformer

Transformer впервые предложили в 2017 году в статье от Google “Attention is all you need”

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

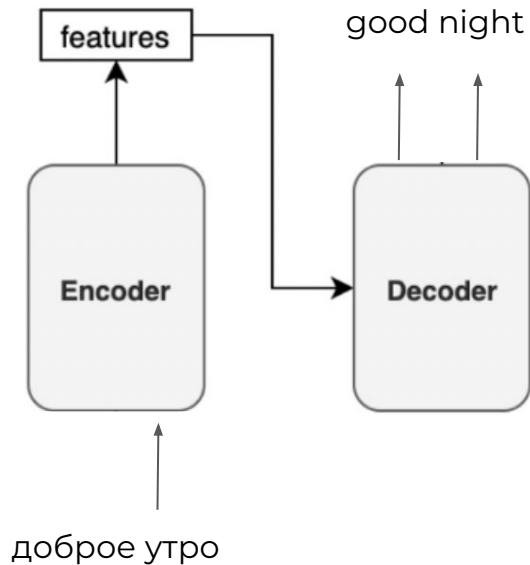
Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

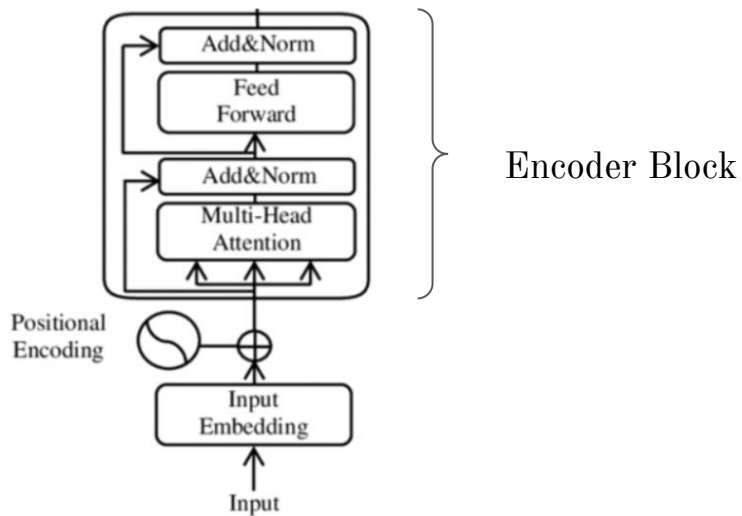
Идея Transformer

У Transformer архитектура тоже имеет тип Encoder-Decoder



Transformer Encoder

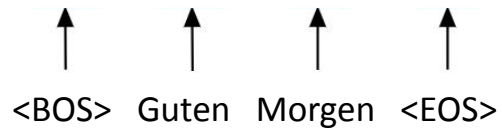
Encoder обрабатывает токены
входящей последовательности
параллельно



Source
embeddings

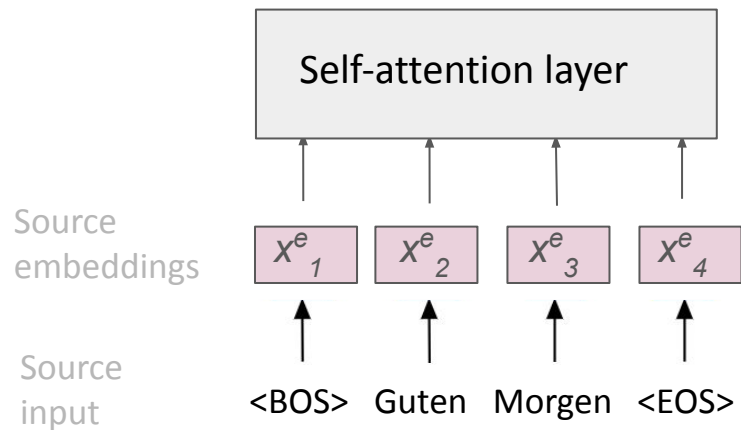


Source
input



Слой self-attention вычисляет дополнительную информацию для эмбединга каждого токена на основе его контекста (т.е. эмбедингов всех других токенов)

“Sport is good, it helps a lot”



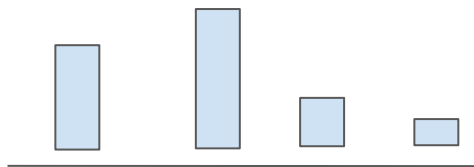
$$a_1^e = \sum_{i=1}^4 w_{1i} x_i^e$$

a_1^e

Агрегация

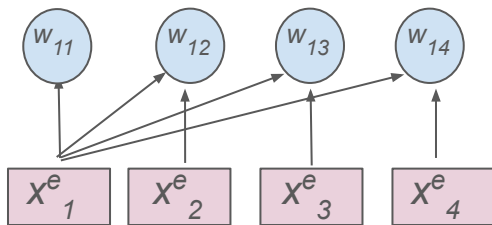
0.4 0.55 0.1 0.05

SoftMax



Self-attention

$$w_{1i} = s(x_1^e, x_i^e)$$



Source embeddings

Source input

<BOS> Guten Morgen <EOS>

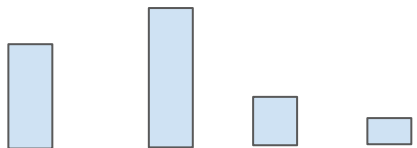
$$a_2^e = \sum_{i=1}^4 w_{2i} x_i^e$$

a_1^e a_2^e

Агрегация

0.25 0.45 0.25 0.05

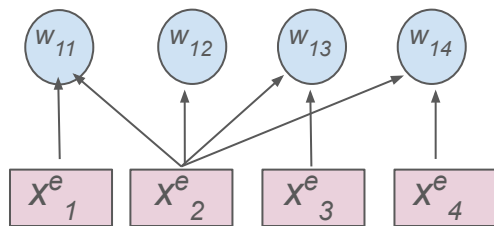
SoftMax



Self-attention

$$w_{2i} = s(x_2^e, x_i^e)$$

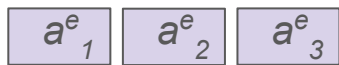
Source embeddings



Source input

\uparrow \uparrow \uparrow \uparrow
 <BOS> Guten Morgen <EOS>

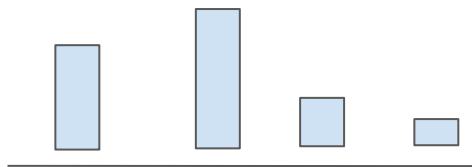
$$a_3^e = \sum_{i=1}^4 w_{3i} x_i^e$$



Агрегация

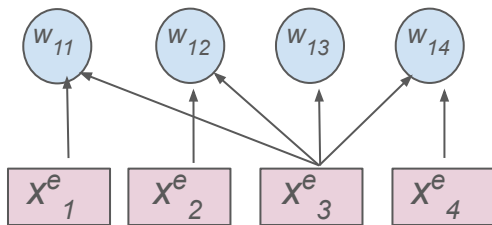
0.4 0.5 0.05 0.05

SoftMax



Self-attention

$$w_{3i} = s(x_2^e, x_i^e)$$



Source embeddings

Source input

<BOS> Guten Morgen <EOS>

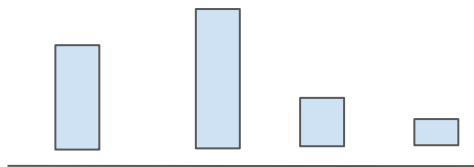
$$a_4^e = \sum_{i=1}^4 w_{4i} x_i^e$$



Агрегация

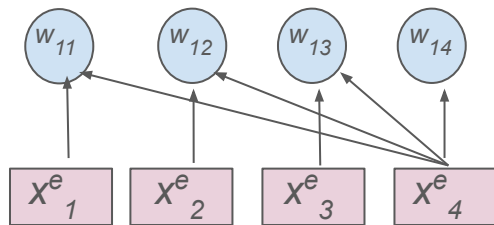
0.2 0.1 0.65 0.05

SoftMax



Self-attention

$$w_{4i} = s(x_4^e, x_i^e)$$

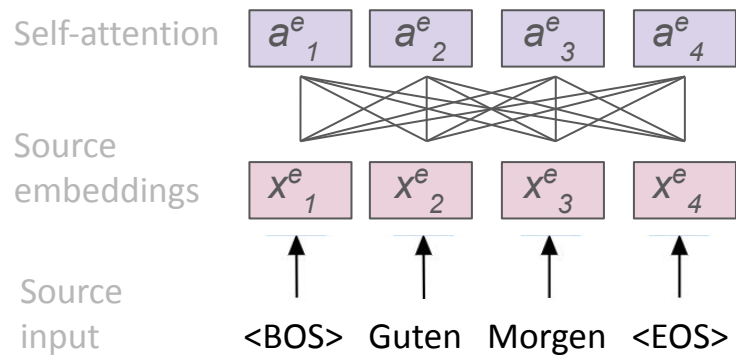


Source embeddings

Source input

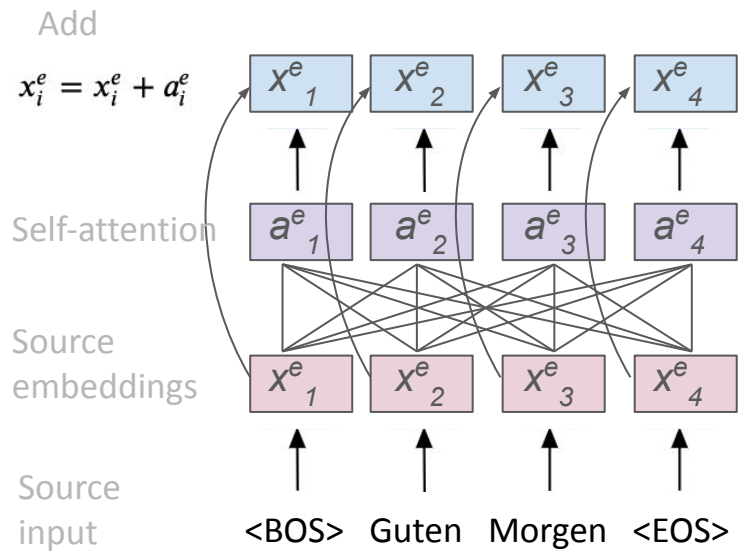
<BOS> Guten Morgen <EOS>

Self-Attention применяется ко всем
эмбедингам параллельно

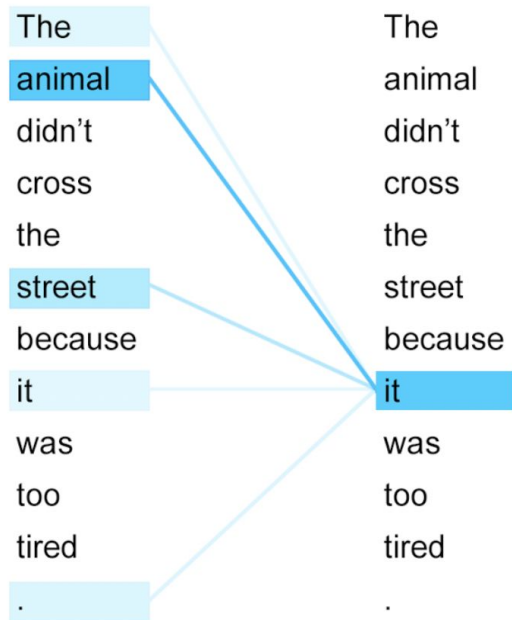


Слой self-attention вычисляет дополнительную информацию для эмбединга каждого токена на основе его контекста

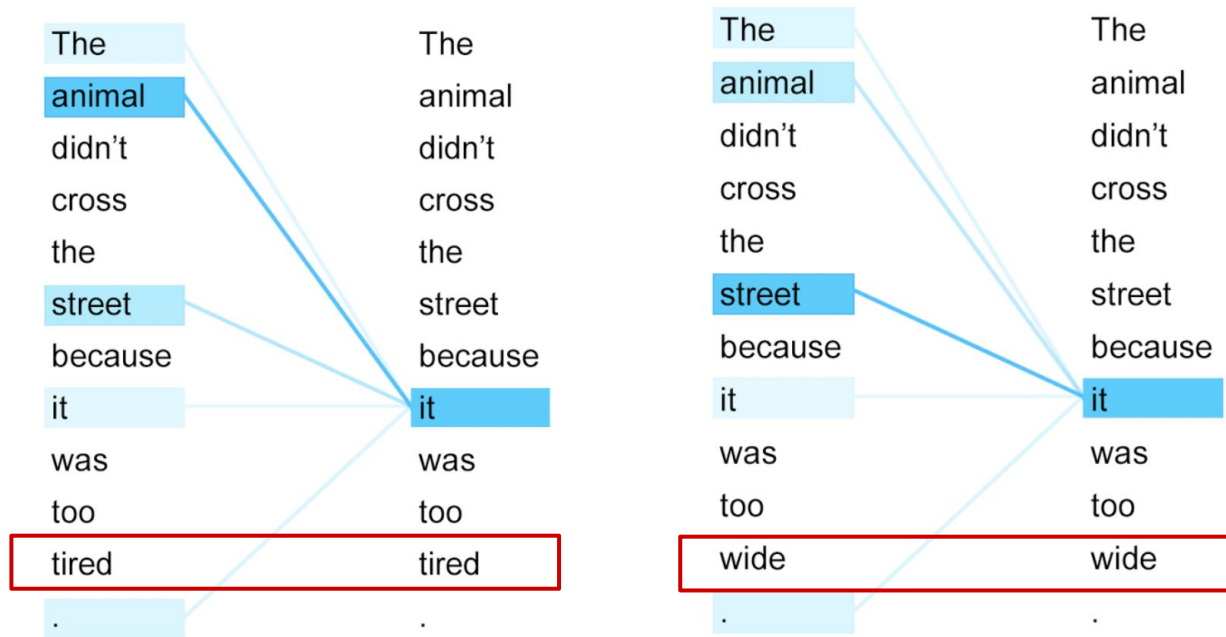
“Sport is good, it helps a lot”



Визуализация весов слоя self-attention

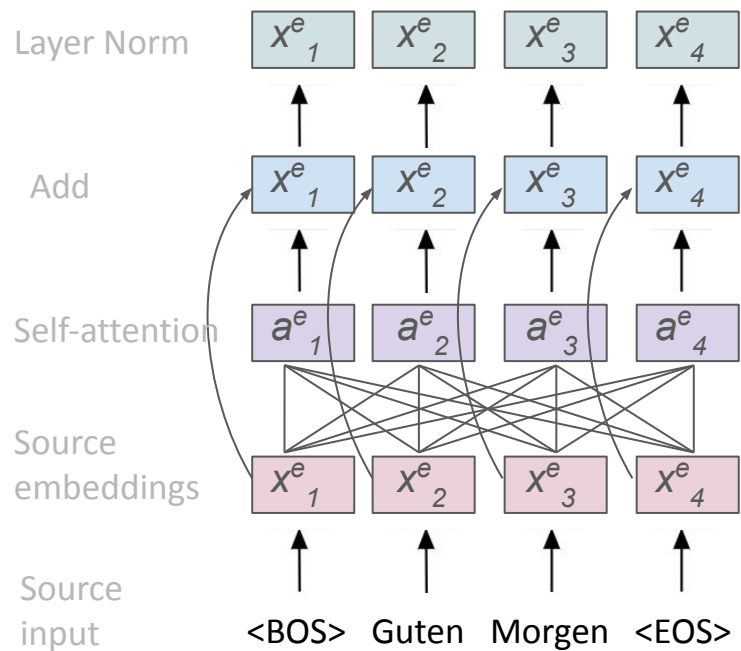


Визуализация весов слоя self-attention

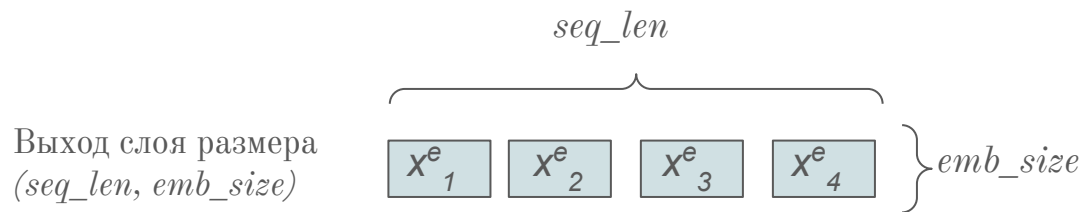


LayerNorm делает так, что выходы слоя всегда будут иметь одинаковые mean и variance.

Это делает обучение сети более стабильным



Layer Norm

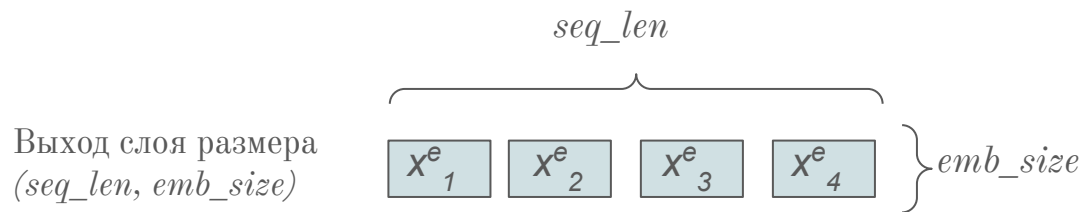


Считаем среднее и стандартное отклонение по всем координатам каждого эмбединга x_i^e :

$$\mu_i = \frac{1}{emb_size} \sum_{j=1}^{emb_size} x_j^e \quad \sigma_i = \sqrt{\frac{1}{emb_size} \sum_{j=1}^{emb_size} (x_j^e - \mu_i)^2}$$

$$1 \leq i \leq seq_len$$

Layer Norm



После этого обновляем элементы каждого эмбединга x_i^e следующим образом:

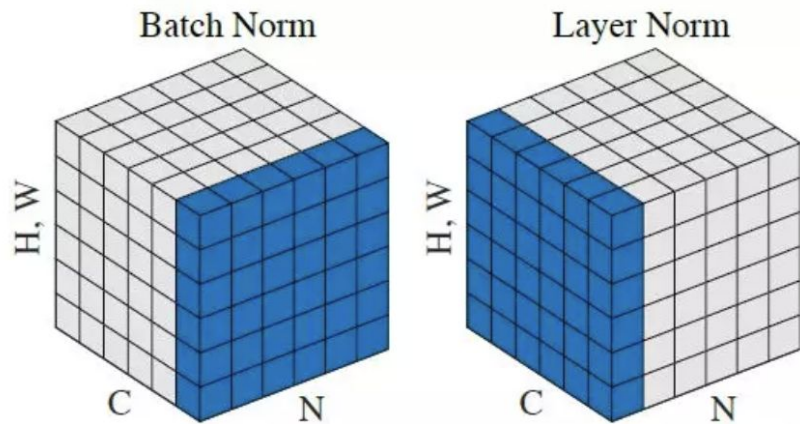
$$x_i^e = \frac{x_i^e - \mu_i}{\sigma_i + \epsilon} \cdot \gamma + \beta$$

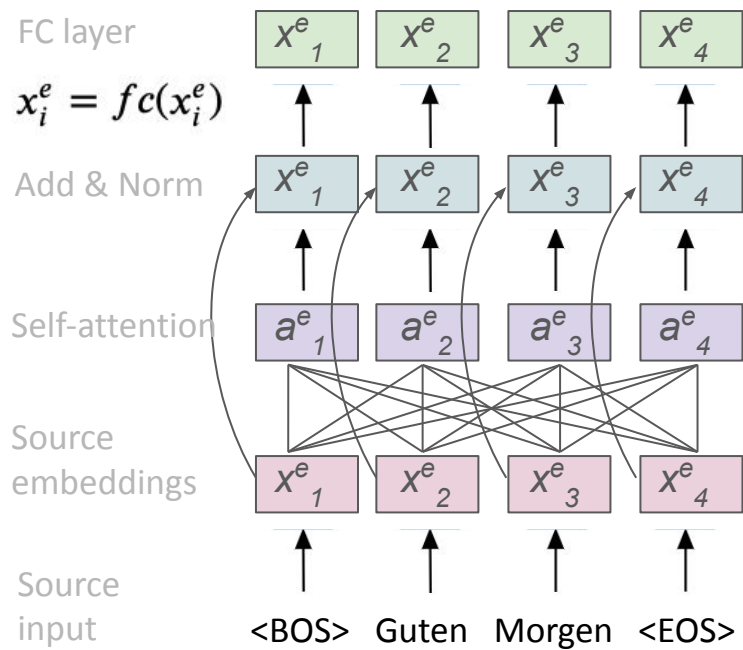
$$1 \leq i \leq seq_len$$

Здесь γ и β — обучаемые параметры

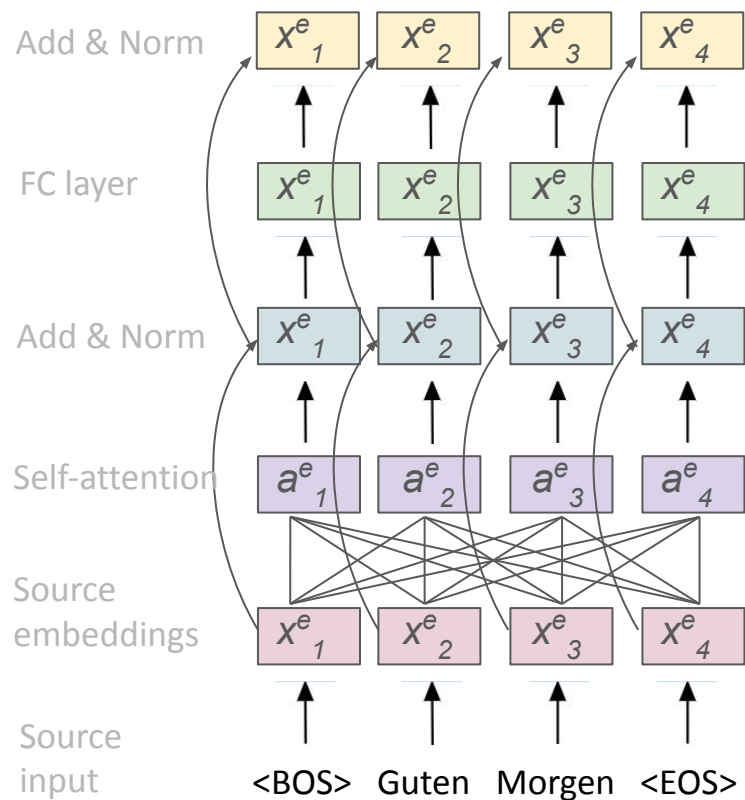
Layer Norm

LayerNorm можно лучше понять, если сравнить его с BatchNorm:





Transformer Encoder



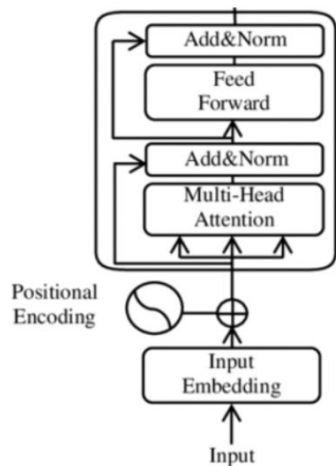
- Результат работы блока Encoder — обновленные эмбединги токенов.
- Каждый слой обрабатывает эмбединги параллельно
- Каждый слой, кроме слоя self-attention, обрабатывает эмбединги независимо
- Transformer Encoder может обрабатывать последовательности разной длины
- Слои в энкодере можно менять местами

Transformer Encoder

Мы почти собрали Encoder.

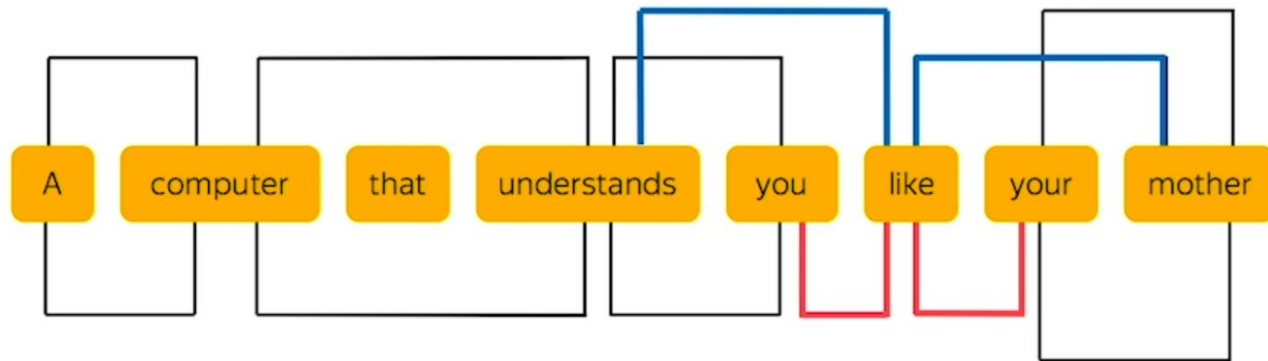
Чего не хватает:

- Multi-head attention
- Positional Encoding

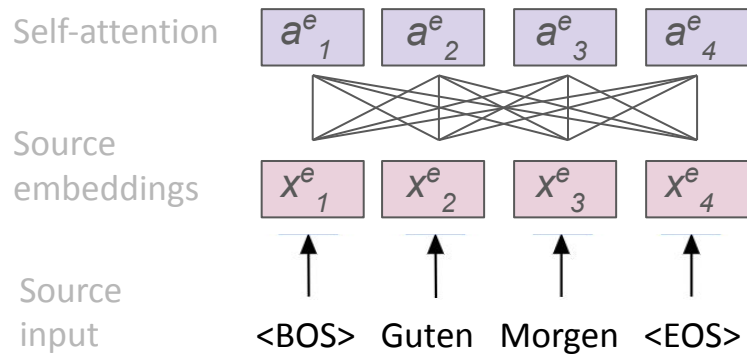


Multi-Head Self-Attention

Между словами в предложении могут быть зависимости разного типа



Multi-Head Self-Attention



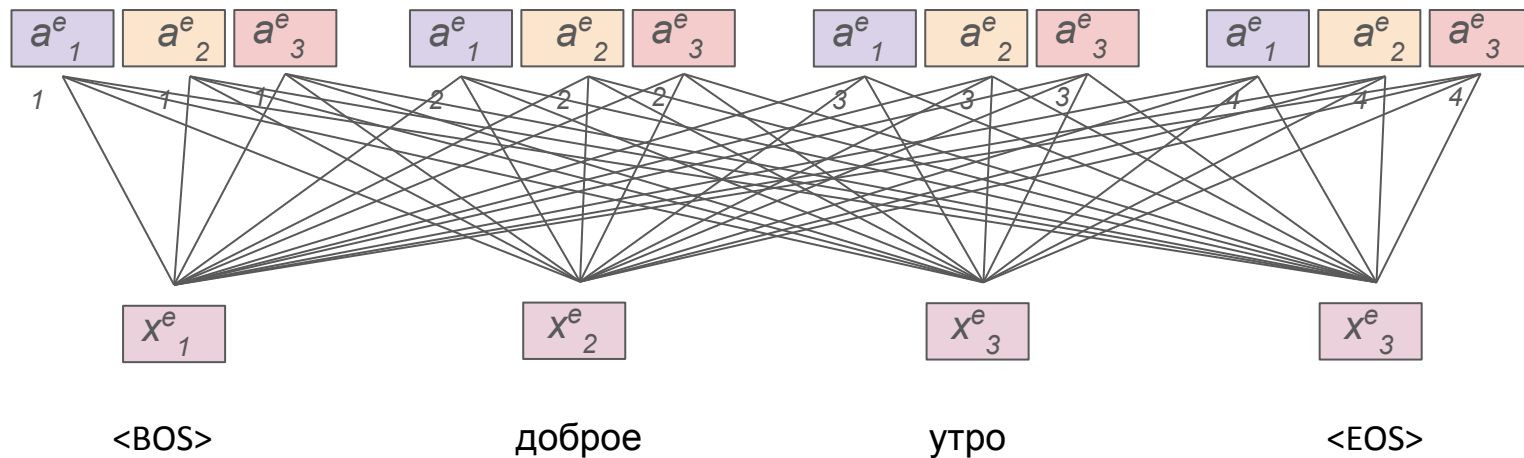
Multi-Head Self-Attention

У всех голов attention
различные веса

k — количество голов Attention:

$$w_{ij}^k = s^k(x_i^e, x_j^e)$$

$$a_{ki}^e = \sum_{j=1}^4 w_{ij}^k x_j^e$$

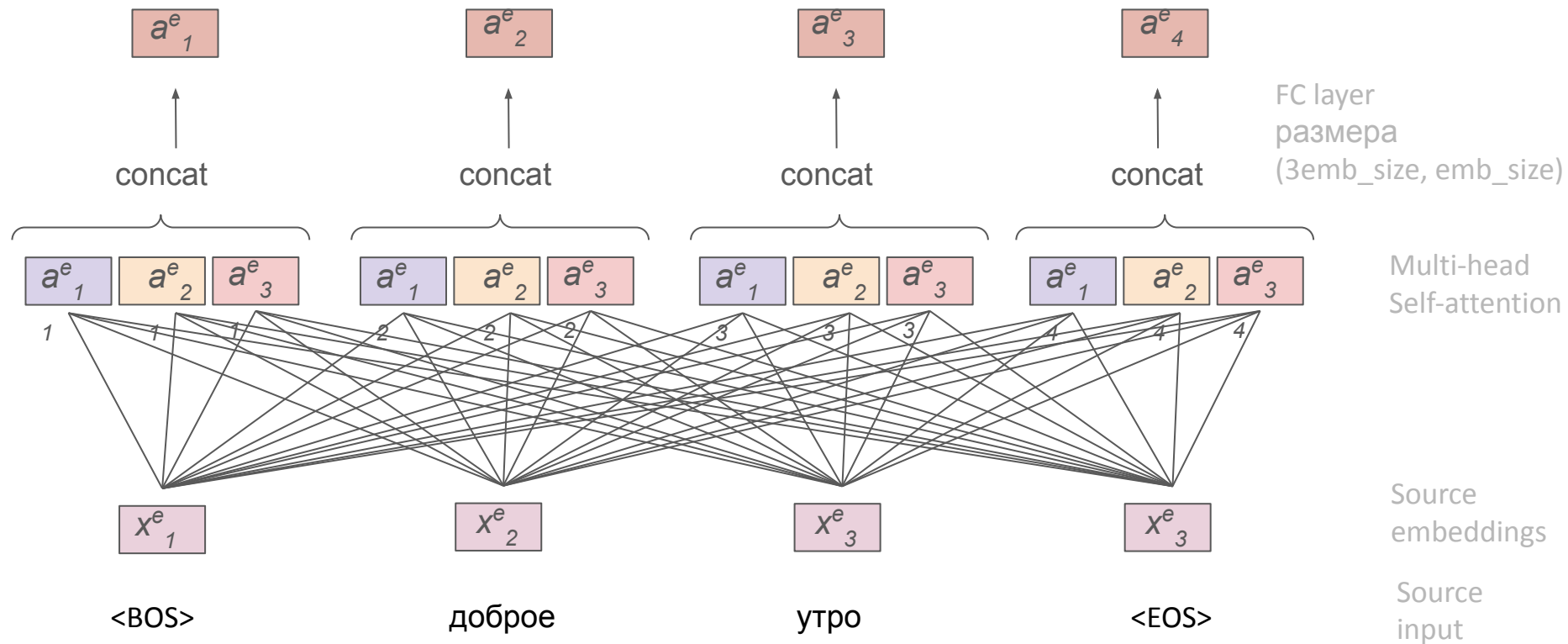


Multi-head
Self-attention

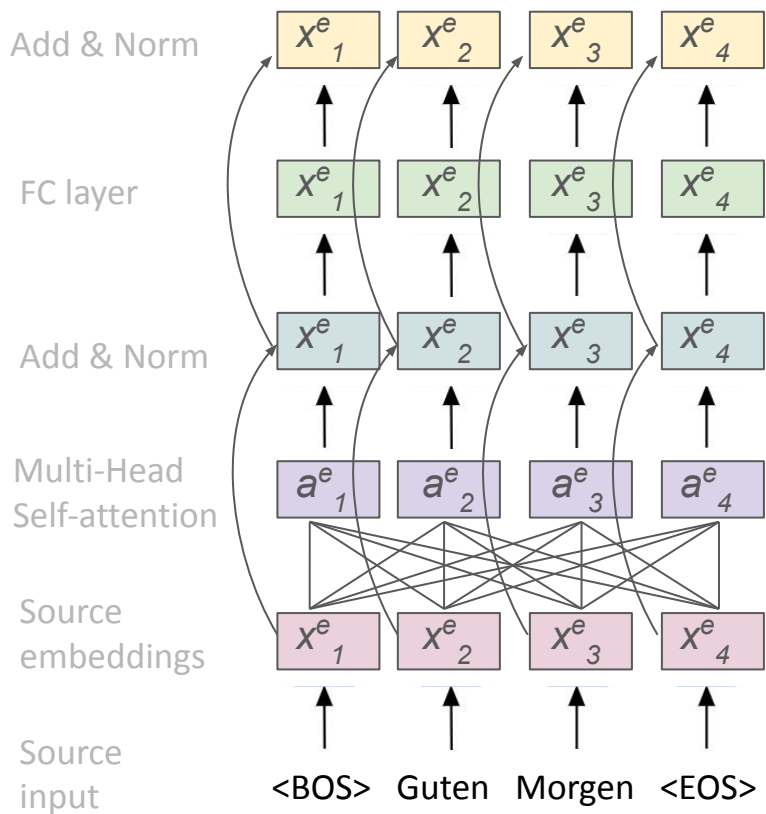
Source
embeddings

Source
input

$$a_i^e = \sigma(W[a_{1i}^e, a_{2i}^e, a_{3i}^e] + b)$$

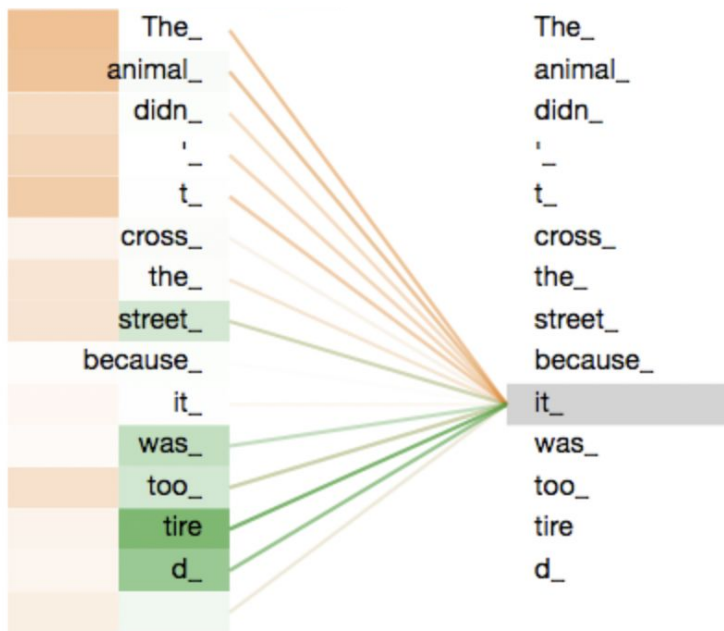


Transformer Encoder



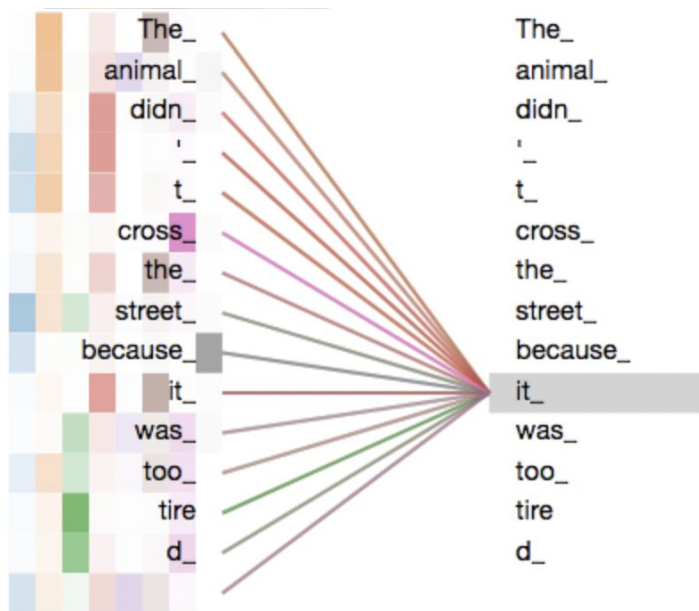
Multi-Head Self-Attention

Разные головы self-attention “обращают внимание”
на разные токены и разную информацию в них



Multi-Head Self-Attention

Разные головы self-attention “обращают внимание”
на разные токены и разную информацию в них

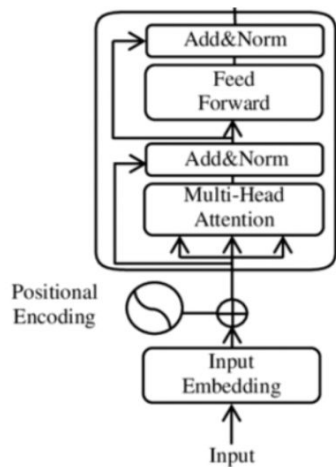


Transformer Encoder

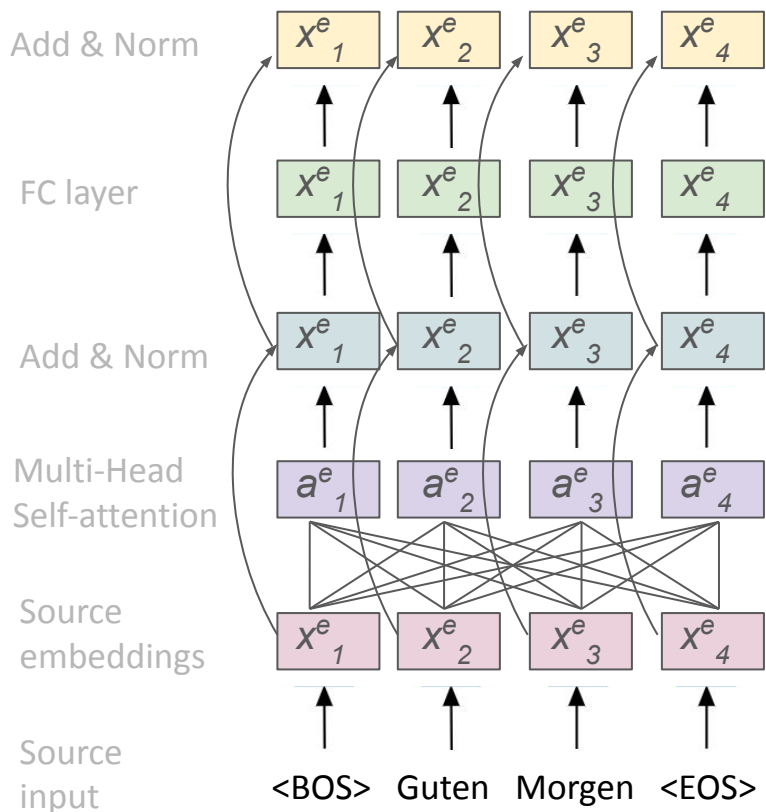
Мы почти собрали Encoder.

Чего не хватает:

- Positional Encoding



Transformer Encoder



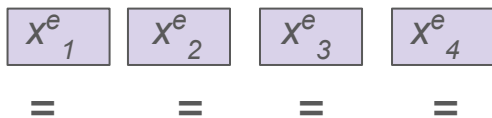
Наша сеть не получает информацию о порядке следования токенов в предложении

Но эта информация важна для понимания смысла эмбедингов и обновления информации о них:

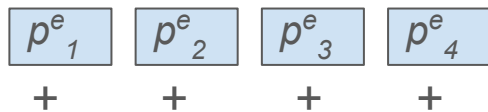
- cat runs horse
- horse runs cat

Positional Encoding

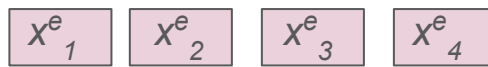
Эмбеддинги с
временным
сигналом



Positional
Encodings



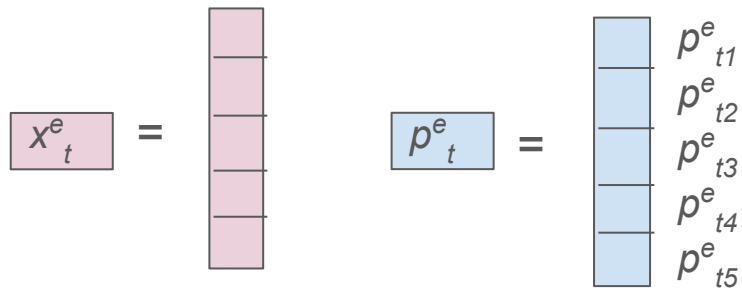
Source
embeddings



Source
input

<BOS> Guten Morgen <EOS>

Positional Encoding



$$p_{ti}^e = \begin{cases} \sin\left(\frac{t}{10000^{\frac{2k}{emb_dim}}}\right), & \text{if } i = 2k \\ \cos\left(\frac{t}{10000^{\frac{2k}{emb_dim}}}\right), & \text{if } i = 2k + 1 \end{cases}$$

Positional Encoding

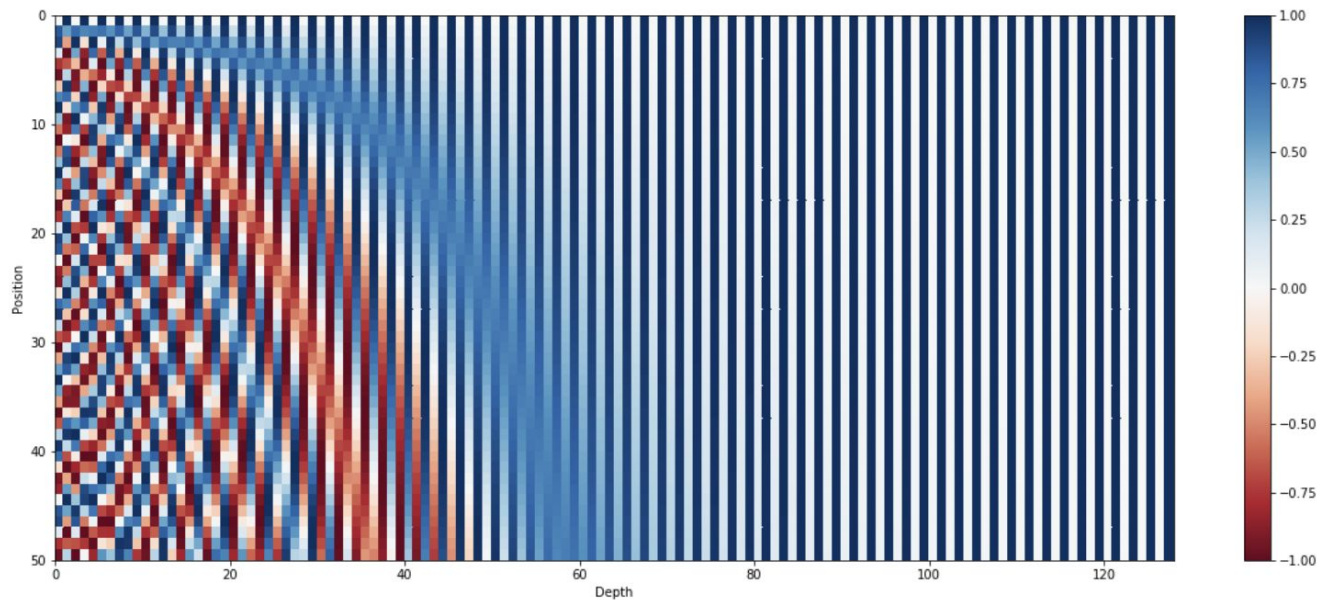


Figure 2 - The 128-dimensional positional encoding for a sentence with the maximum length of 50. Each row represents the embedding vector \vec{p}_t

Positional Encoding

Почему функция positional encoding именно такая:

In this work, we use sine and cosine functions of different frequencies:

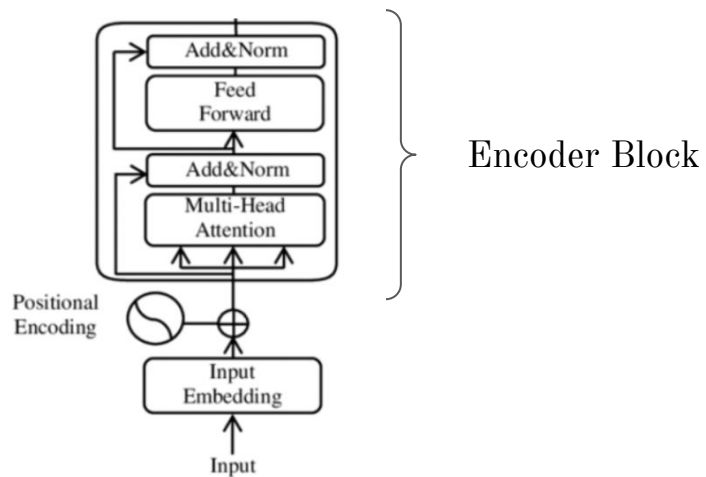
$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

where pos is the position and i is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k , PE_{pos+k} can be represented as a linear function of PE_{pos} .

—

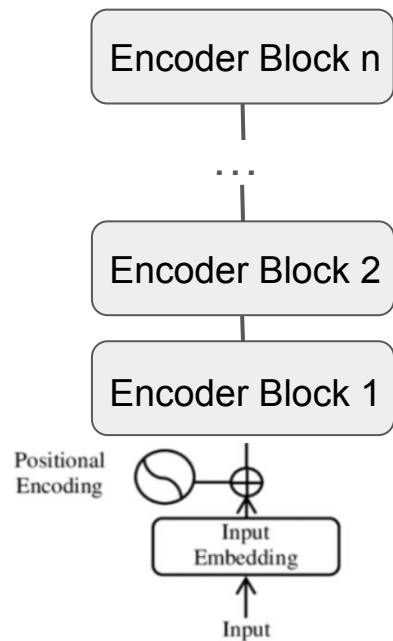
Transformer Encoder

Ура, мы собрали Encoder!

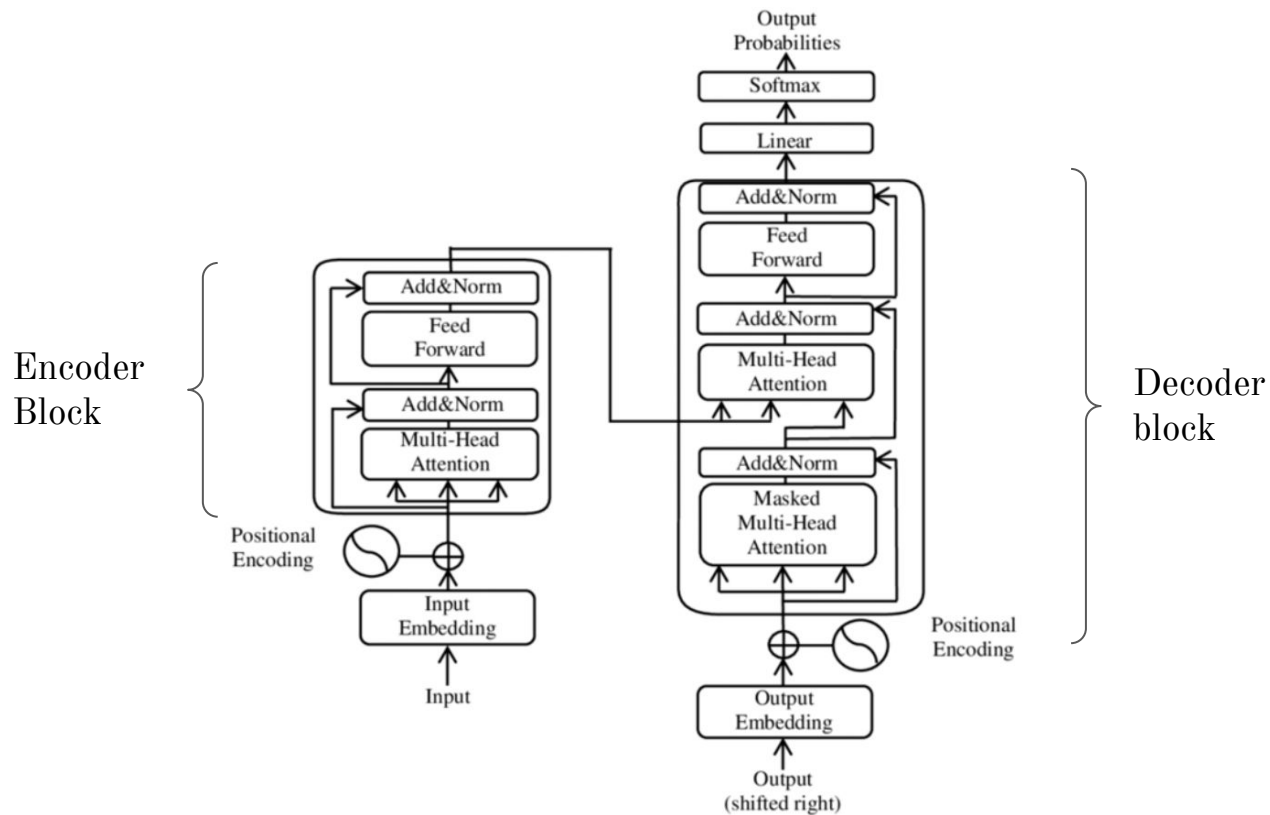


Transformer Encoder

У Encoder могут быть несколько последовательно соединенных блоков



Transformer



Итоги видео

В этом видео мы:

- Обсудили идею устройства архитектуры Transformer;
- Детально разобрали устройство encoder блока Transformer;
- Познакомились с такими вещами, как:
 - Self-attention;
 - Multi-head Attention;
 - Positional Encodings



Deep Learning School

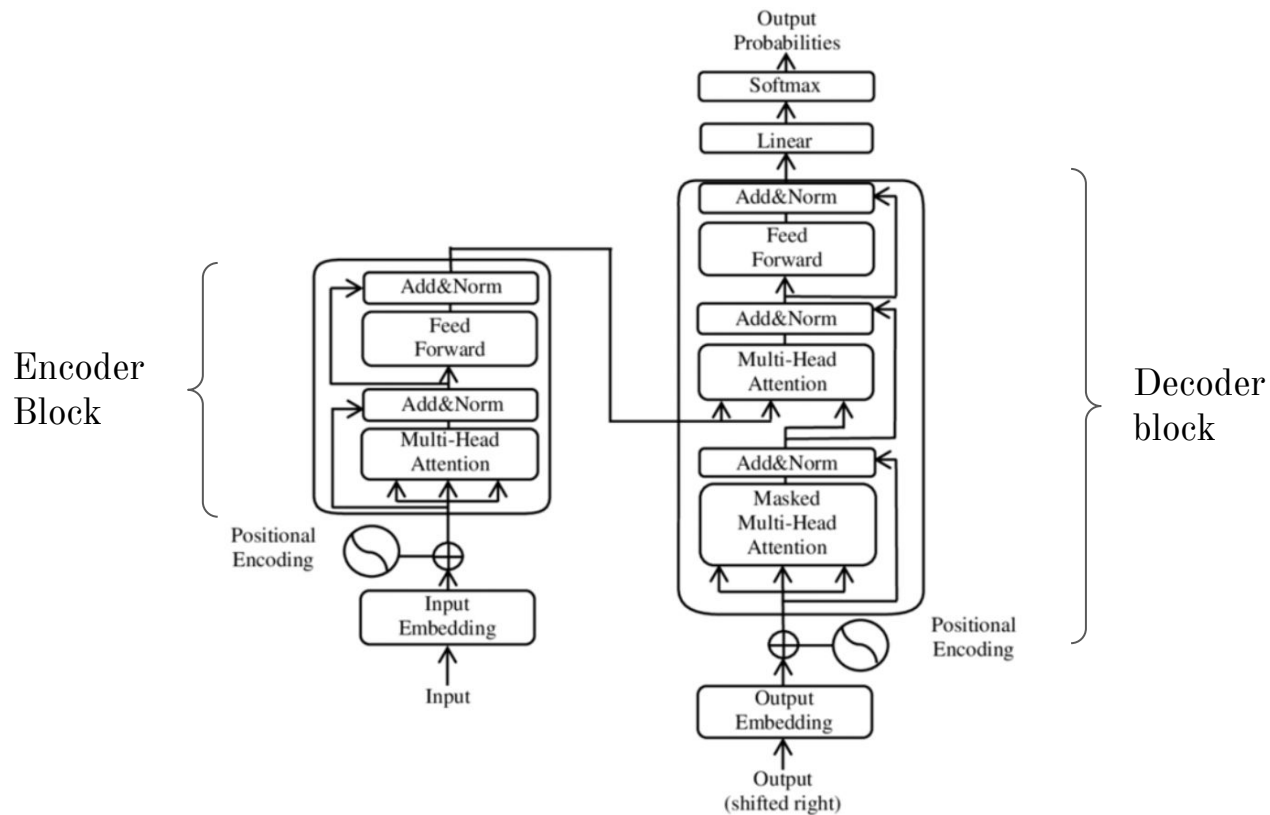
Transformer architecture

Decoder block

В этом видео

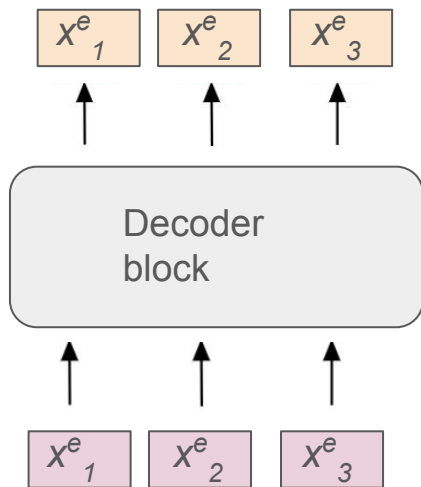
- Идея архитектуры Transformer
- Устройство Transformer Encoder
- **Устройство Transformer Decoder**
- **QKV Attention**

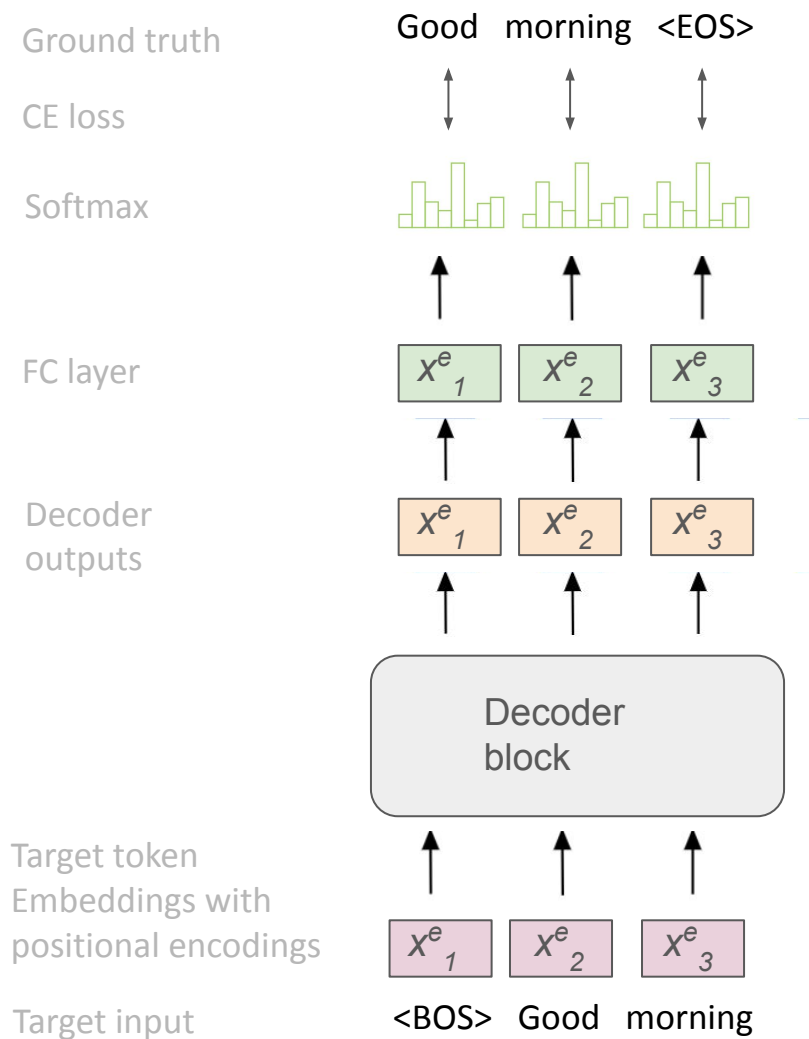
Transformer



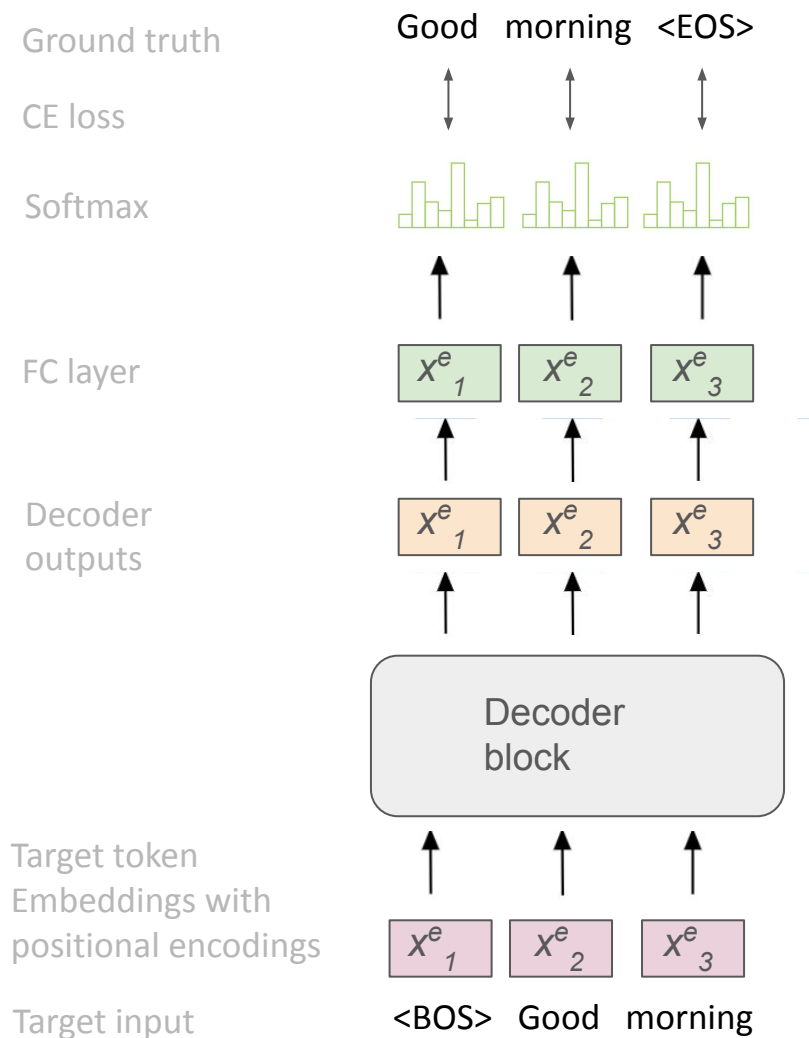
Выходы decoder
block
(обновленные
эмбединги)

Decoder block
inputs



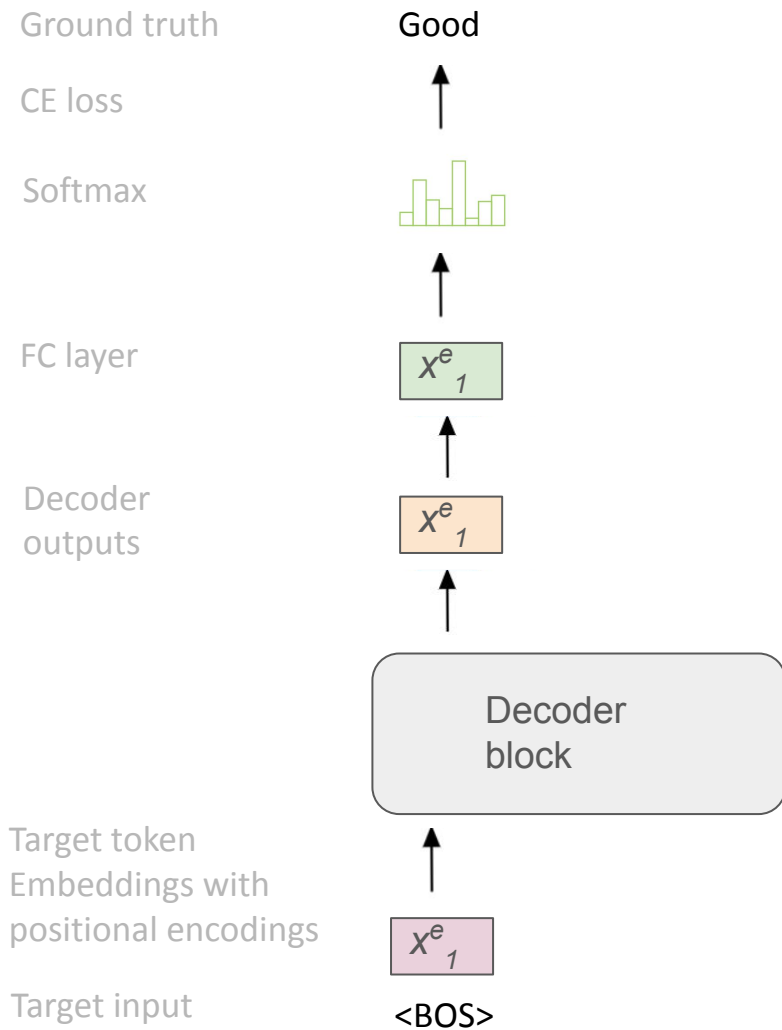


Во время обучения мы можем
подать на вход декодеру сразу все
токены target предложения

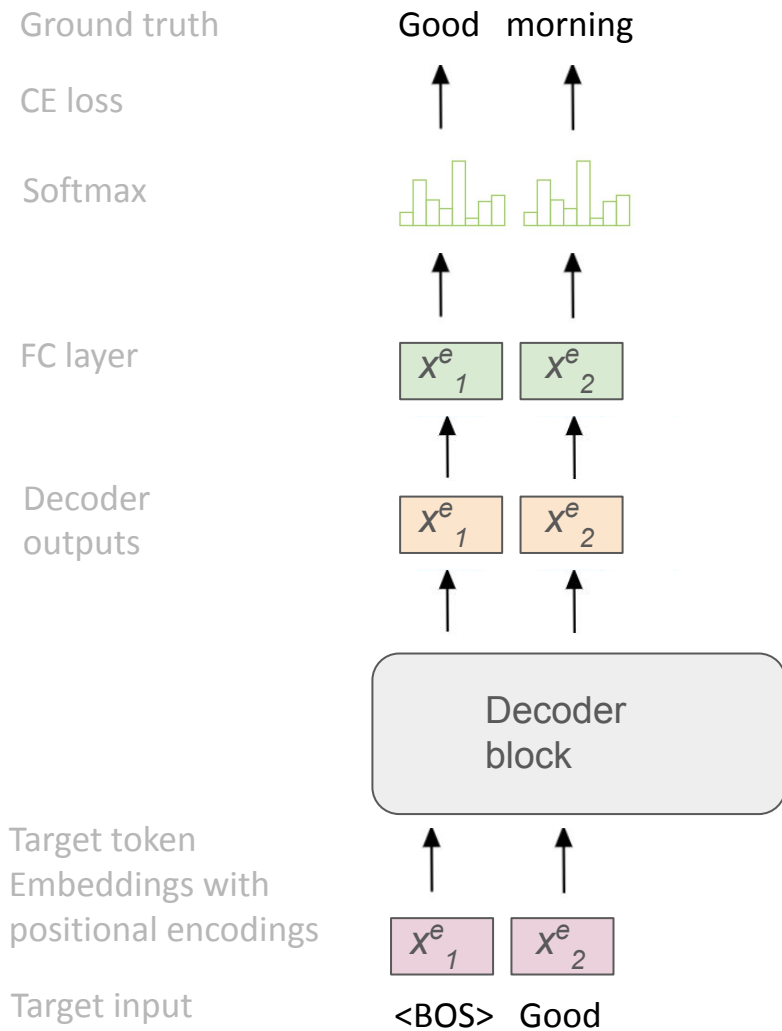


Во время обучения мы можем
подать на вход декодеру сразу все
токены target предложения

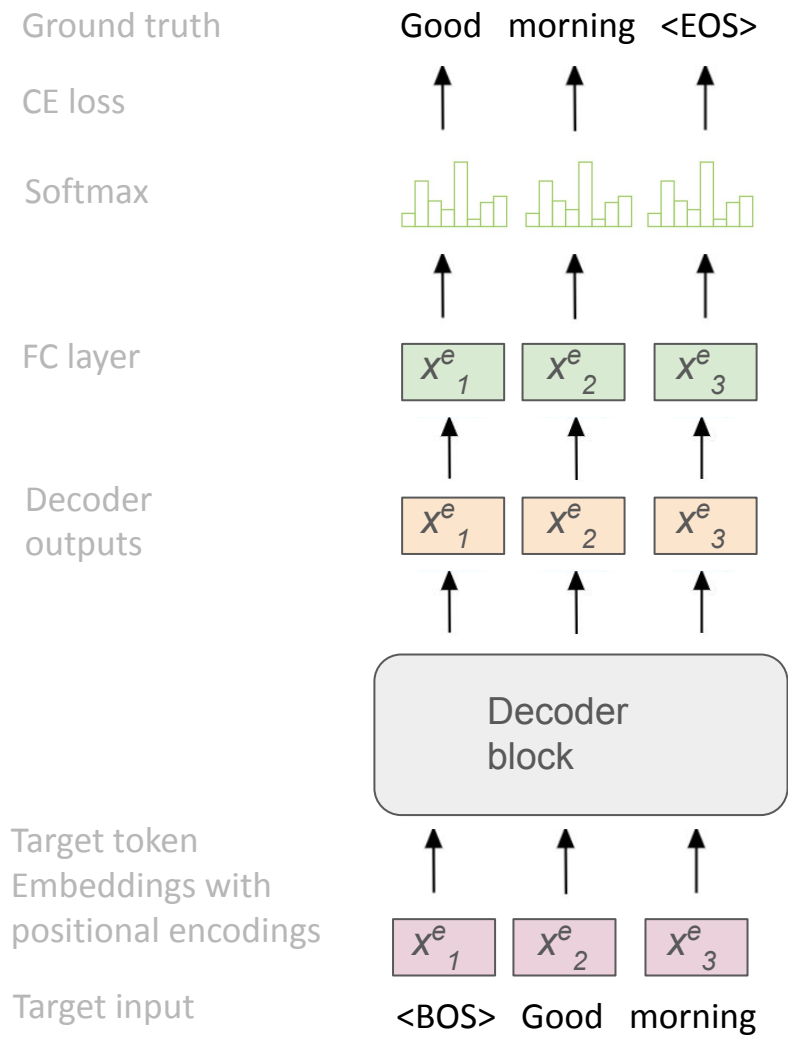
Во время инференса decoder
генерирует выходную
последовательности
авторерессивно



Во время инференса decoder
генерирует выходную
последовательности
авторерессивно

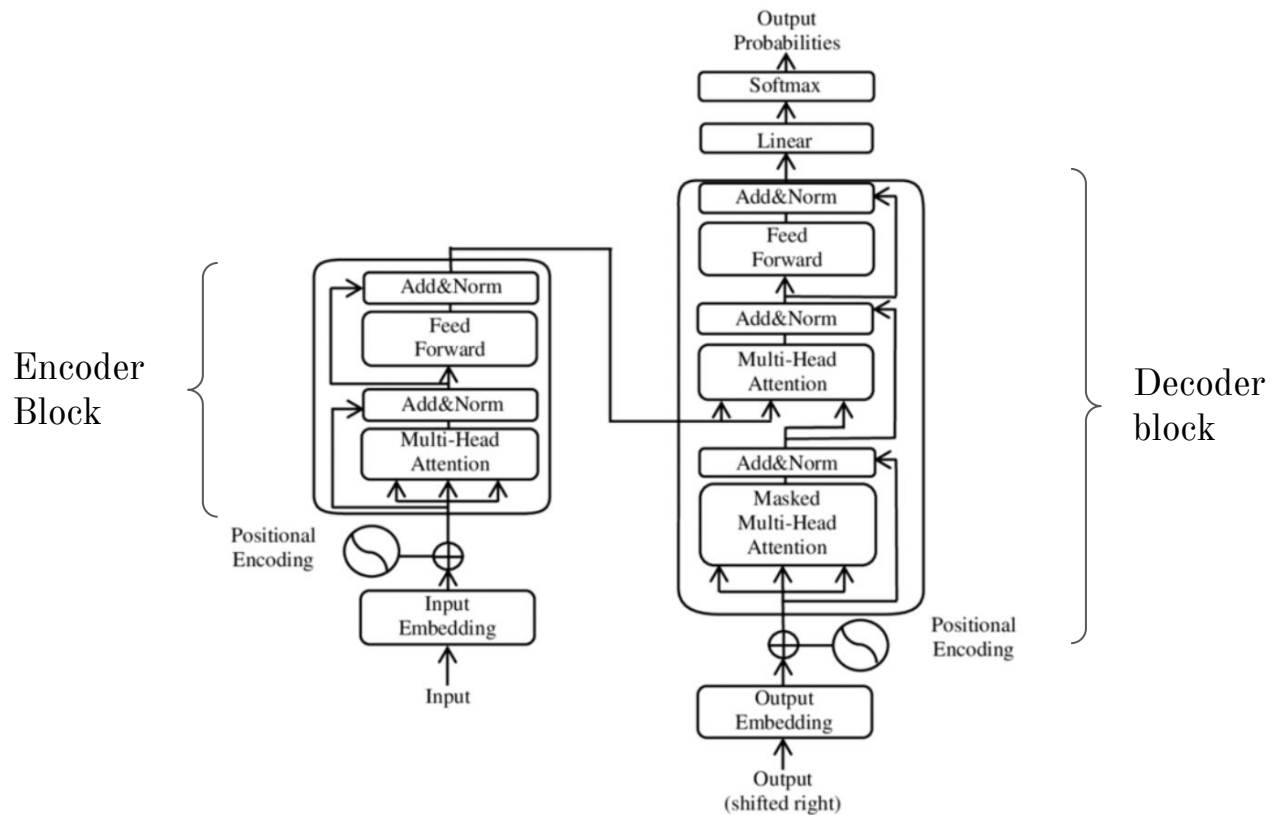


Во время инференса decoder
генерирует выходную
последовательности
авторерессивно



Во время инференса decoder
генерирует выходную
последовательности
авторерессивно

Transformer



Masked Self-Attention

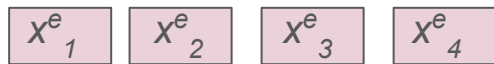
Что было в Encoder блоке:

Encoder

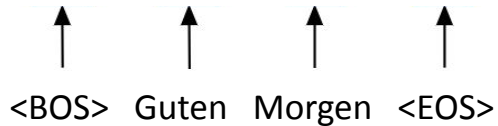
Self-attention



Source token
embeddings

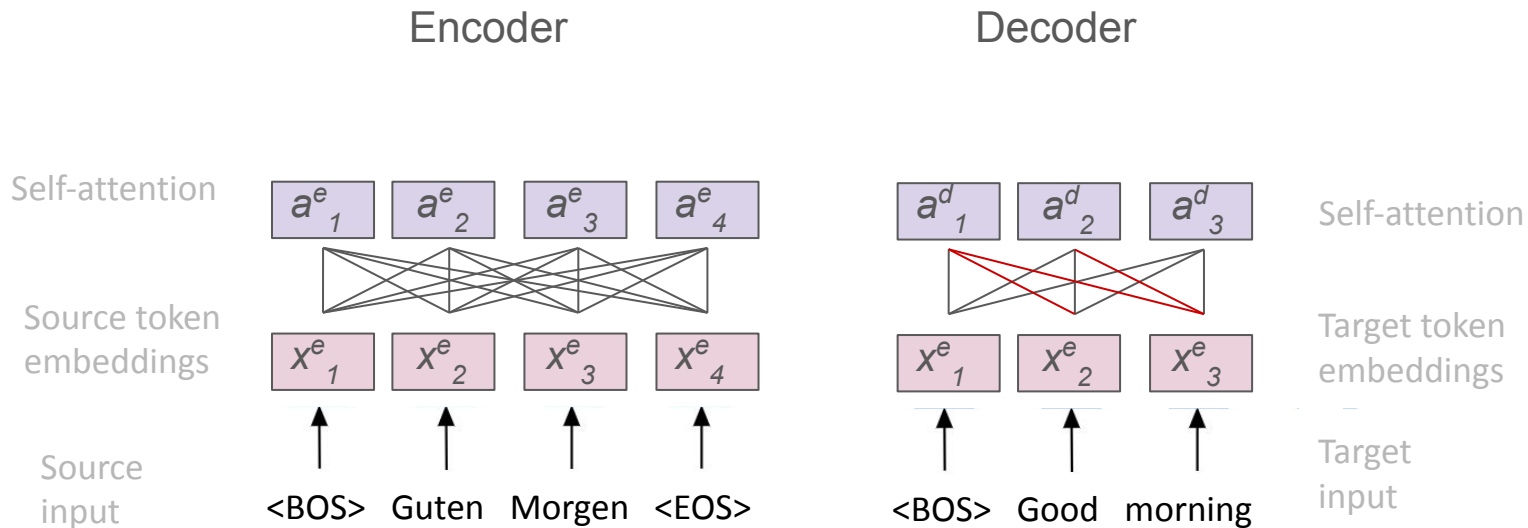


Source
input

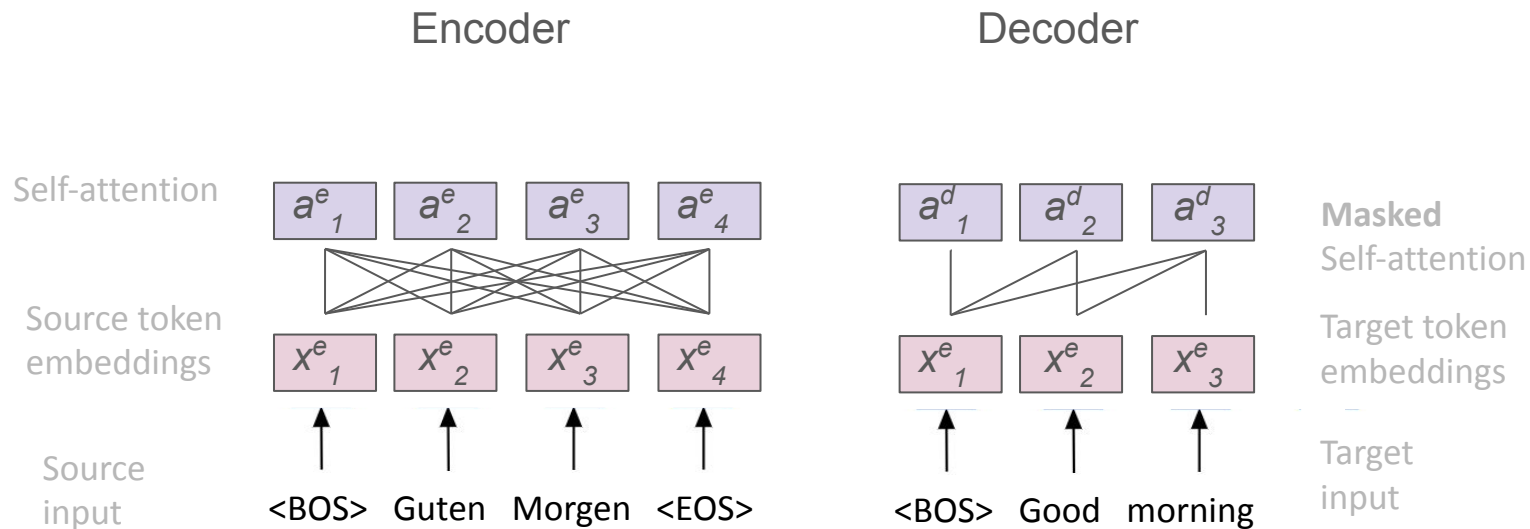


Masked Self-Attention

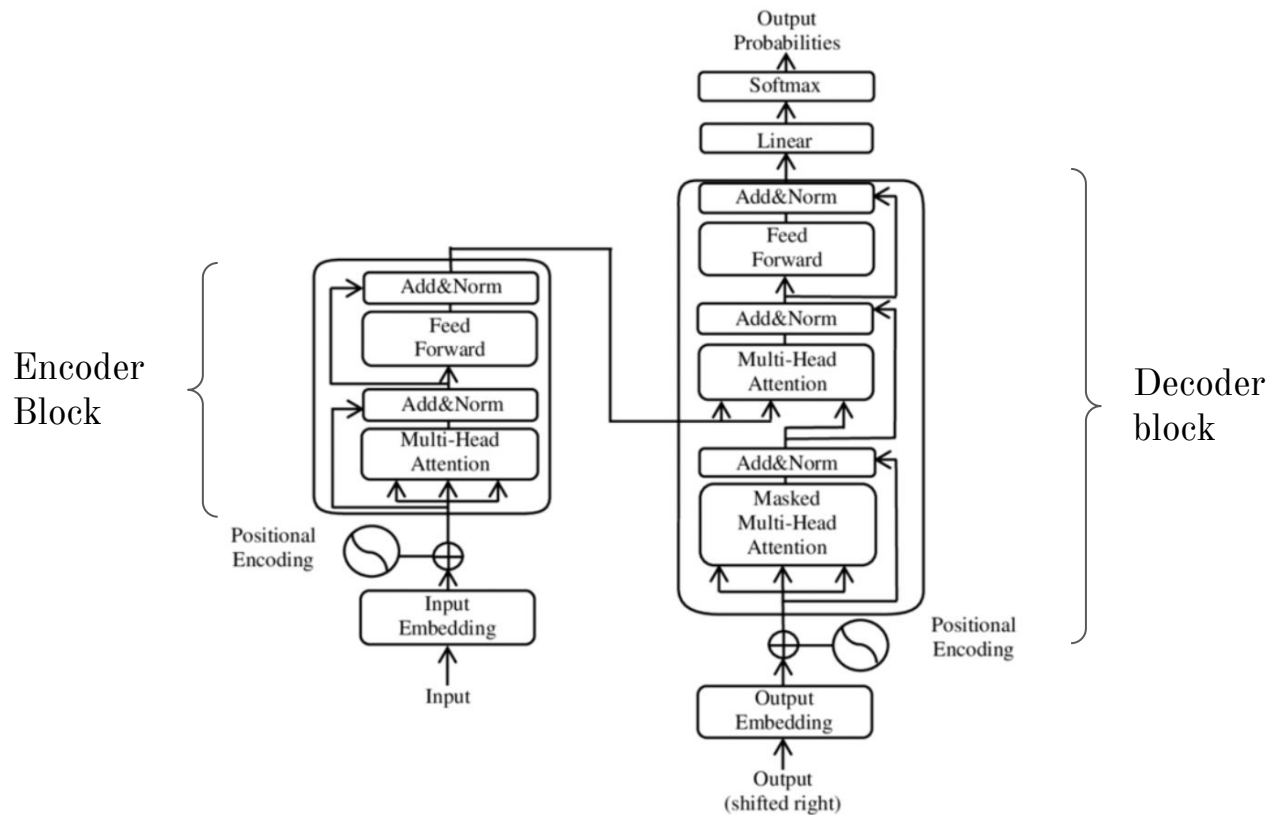
Во время обучения эмбединги x_i^e не могут брать информацию из “будущих” токенов



Masked Self-Attention

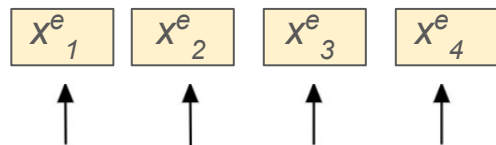


Transformer

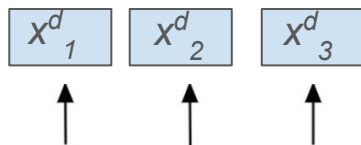


Cross-Attention

Каждый эмбеддинг декодера соберет информацию из эмбеддингов энкодера



Финальные выходы энкодера

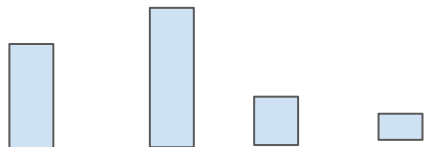


Выходы слоя декодера

$$a_1^{ed} = \sum_{i=1}^4 w_{1i} x_1^d$$

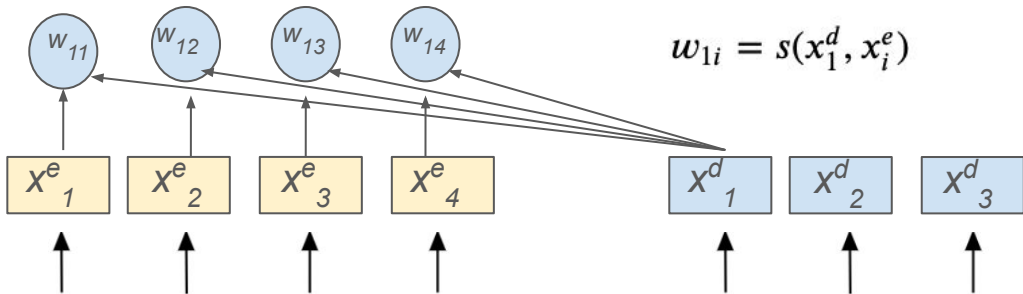
a^{ed}

0.4 0.55 0.1 0.05



SoftMax

$$w_{1i} = s(x_1^d, x_i^e)$$



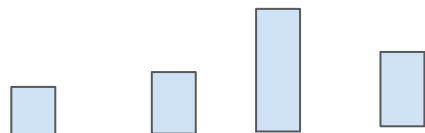
Финальные выходы энкодера

Выходы слоя декодера

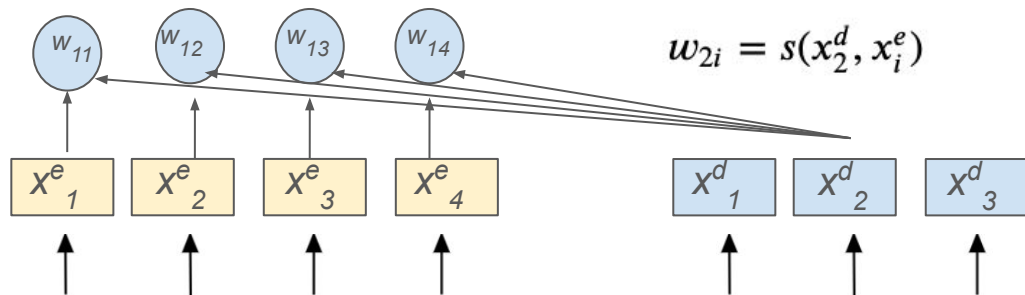
$$a_2^{ed} = \sum_{i=1}^4 w_{2i} x_2^d$$



0.1 0.15 0.5 0.25



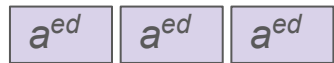
SoftMax



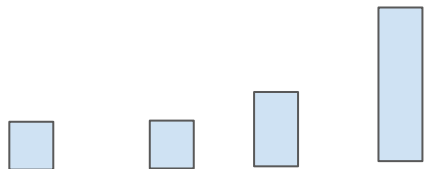
Финальные выходы энкодера

Выходы слоя декодера

$$a_3^{ed} = \sum_{i=1}^4 w_{3i} x_3^d$$

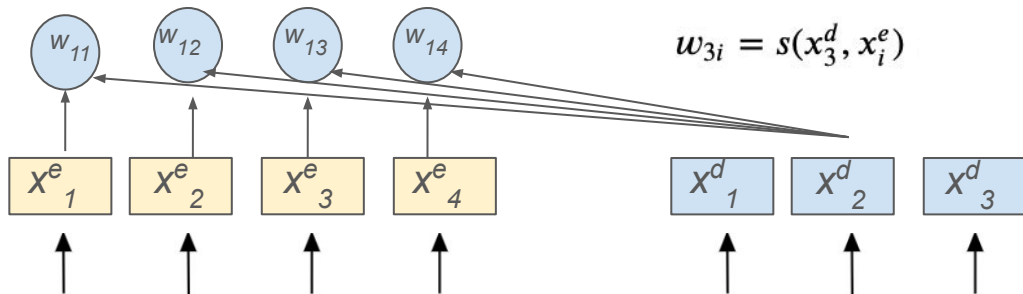


0.11 0.11 0.2 0.56



SoftMax

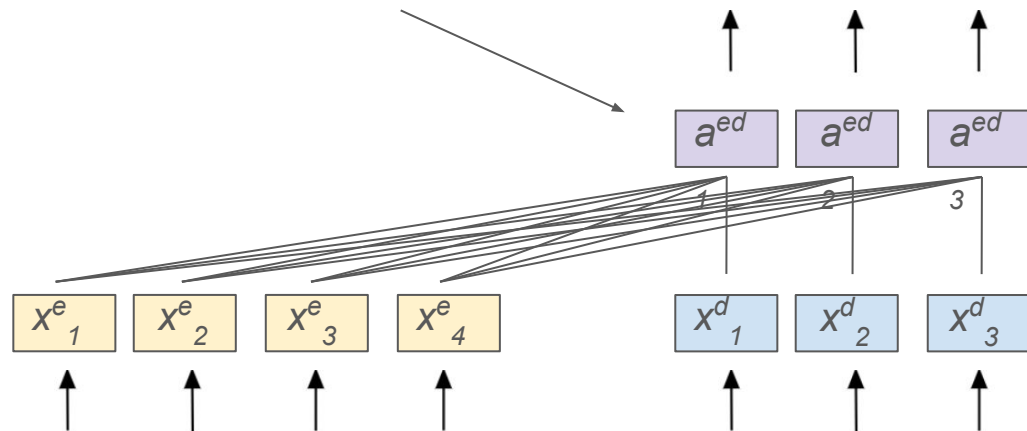
$$w_{3i} = s(x_3^d, x_i^e)$$



Финальные выходы энкодера

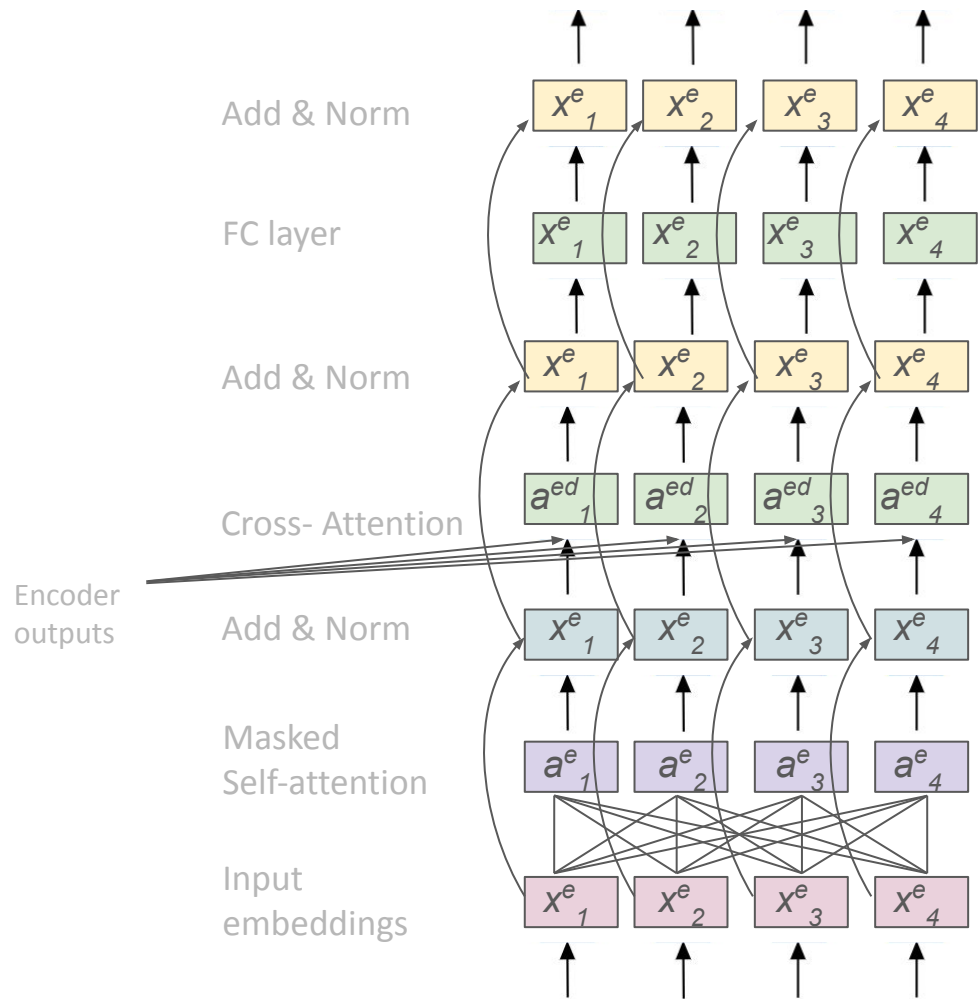
Выходы слоя декодера

Векторы cross-attention

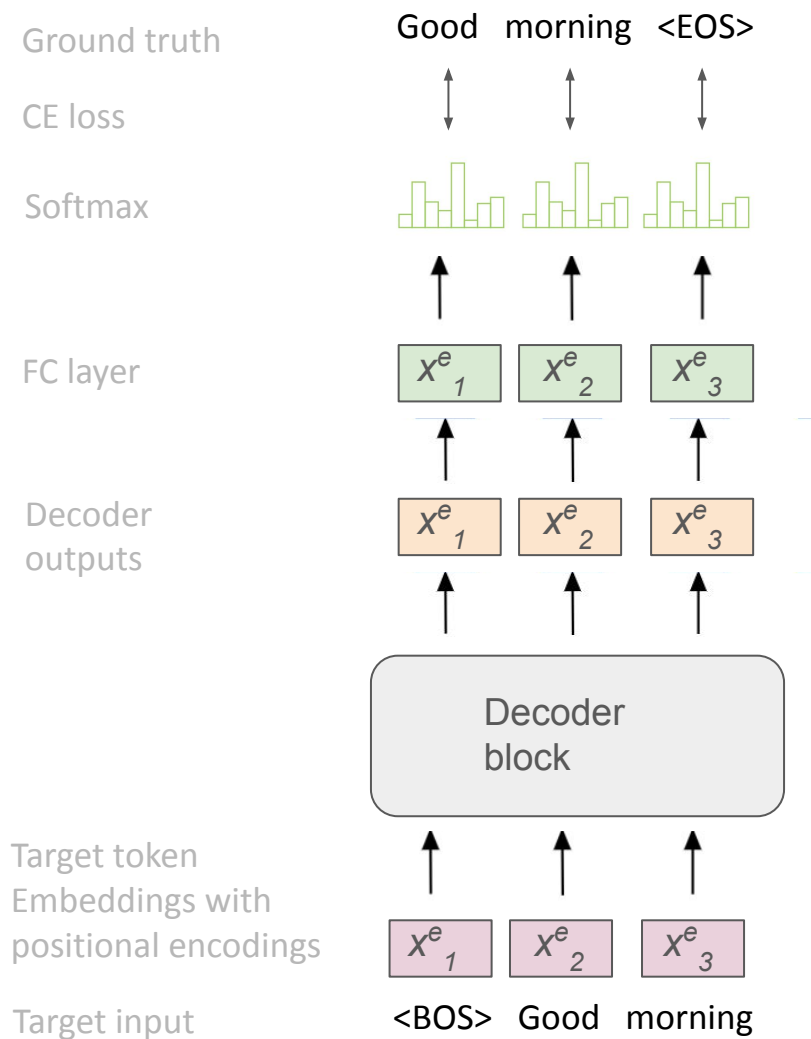


Финальные выходы энкодера

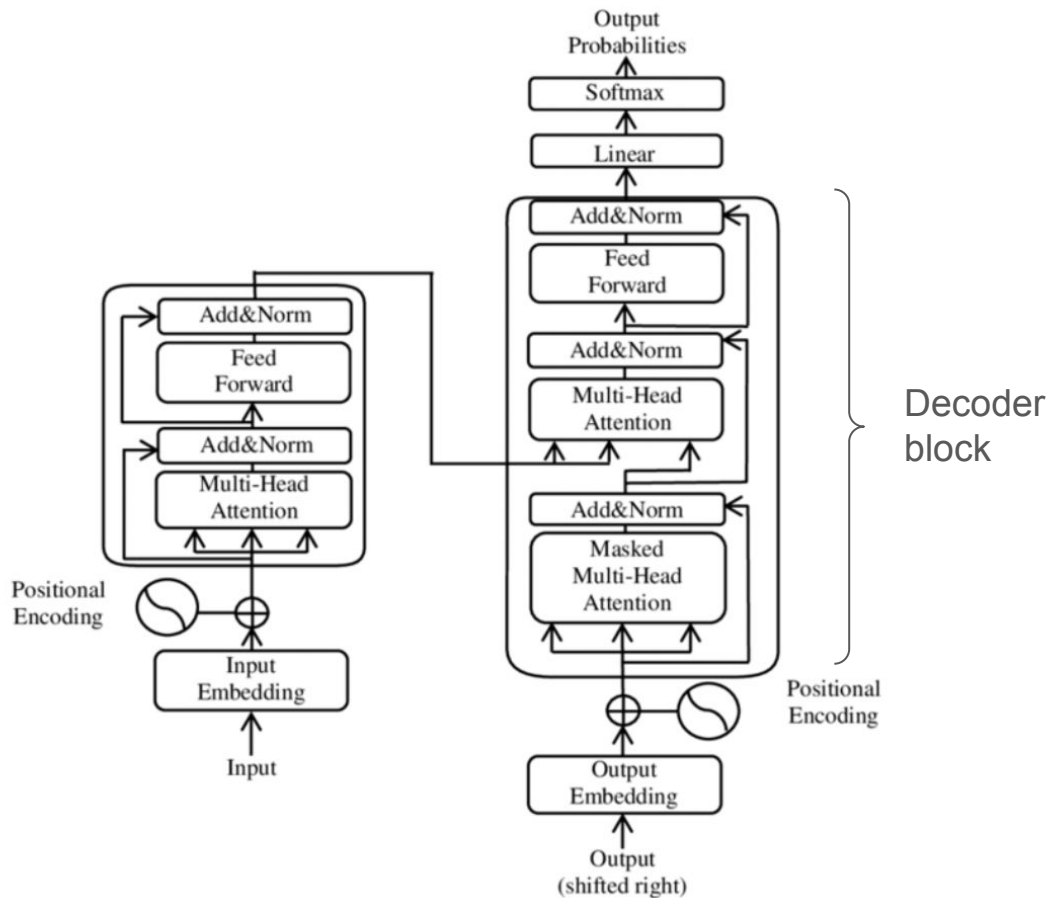
Выходы слоя декодера



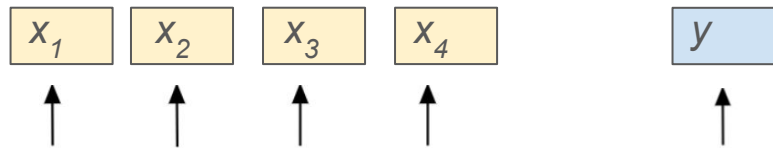
Decoder block



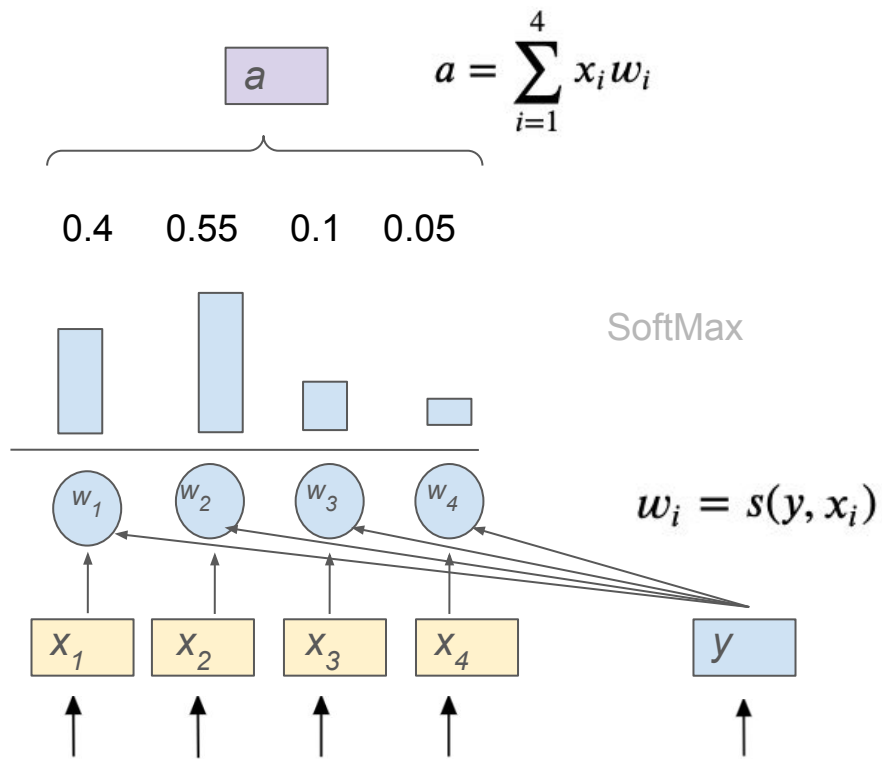
Transformer Decoder



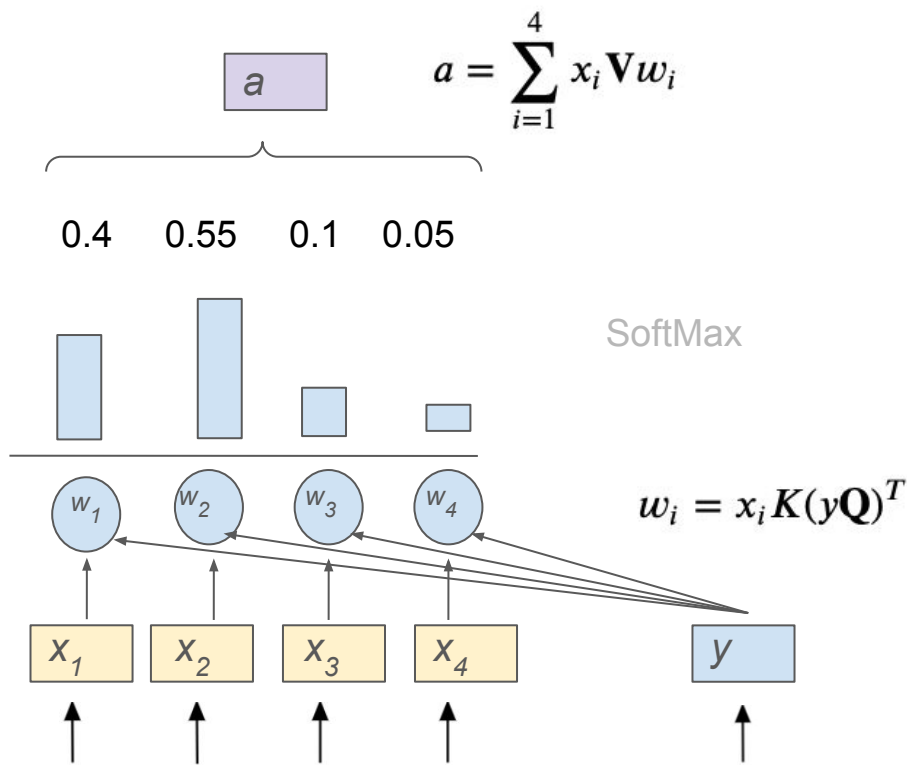
QKV Attention



QKV Attention



QKV Attention



У QKV Attention 3 обучаемые матрицы::

- $Q_{d_{xm}}$ (query)
- $K_{d_{xm}}$ (key)
- $V_{d_{xm}}$ (value)

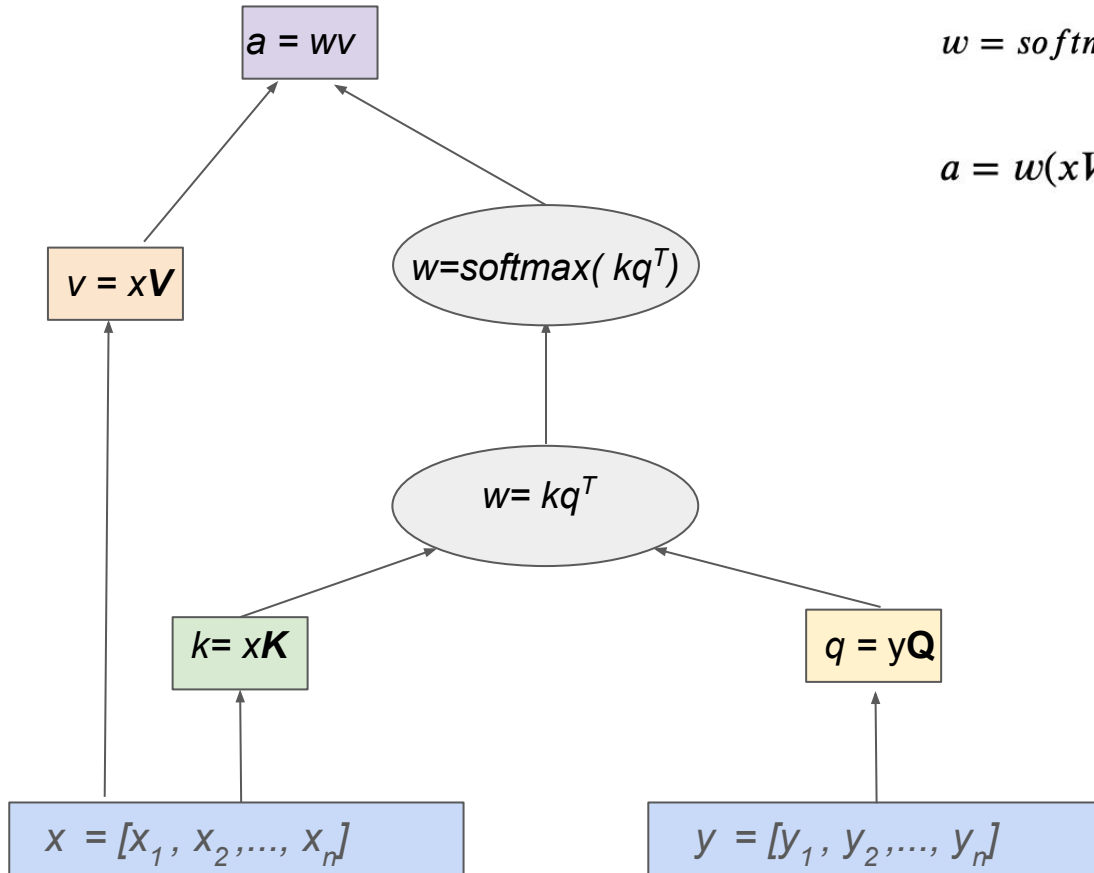
Здесь:

- D — длина эмбеддингов;
- M можно выбирать любым

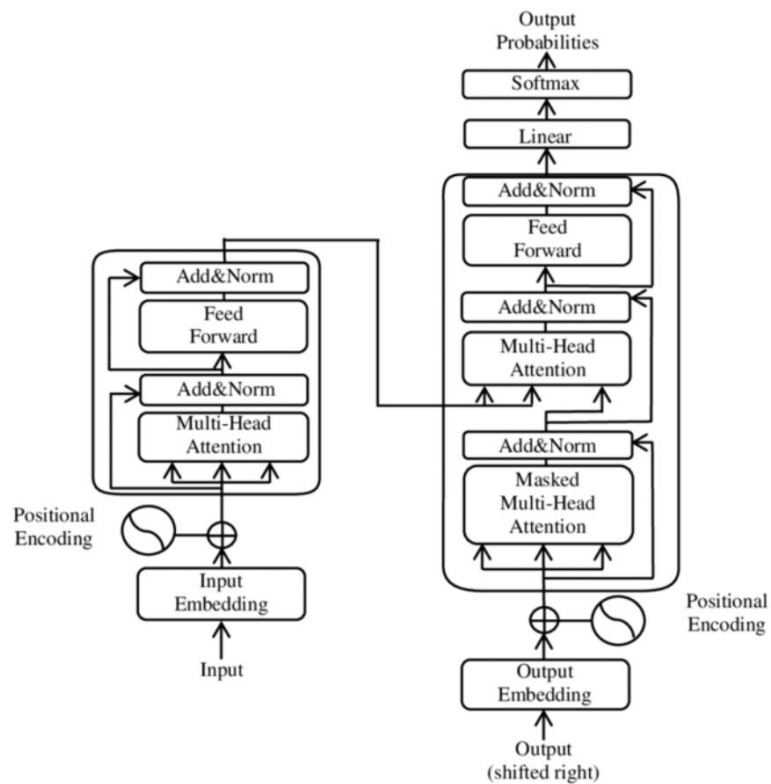
QKV Attention в векторной форме:

$$w = \text{softmax} \left(\frac{xK(yQ)^T}{\sqrt{m}} \right)$$

$$a = w(xV)$$



Transformer



Итоги видео

В этом видео мы:

- Детально разобрали устройство decoder блока Transformer;
- Собрали полную архитектуру Transformer;
- Узнали, как работает QKV Attention.

ИТОГИ ВИДЕО

- Устройство
- Transformer blocks structures:
 - LayerNorm;
 - Self-Attention
 - Masked Self-Attention
 - Cross-Attention
 - QKV Attention