

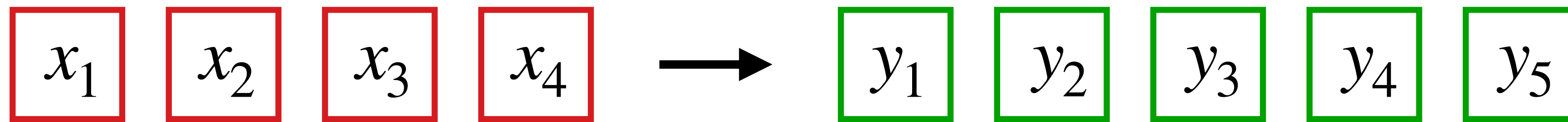
Seq2seq и Трансформер

План

- Постановка задачи Seq2seq
- Рекуррентные сети для Seq2seq
- Механизм внимания
- Трансформер
- Метрики качества Seq2seq
- Методы выбора токенов из их распределения

Seq2seq

Перевод входной последовательности в новую последовательность



Длины последовательностей могут отличаться

Примеры Seq2seq

1. Машинный перевод

Русский ↔ Английский

2. Суммаризация

Длинный текст → короткий текст

3. Изменение стиля текста

Грубый ↔ Вежливый

Формальный ↔ Неформальный

Формальная постановка задачи

Генерация текста:

$$p(x_1, \dots, x_m) = \prod_{i=1}^m p(x_i | x_{<i})$$

Seq2seq: добавляется условие

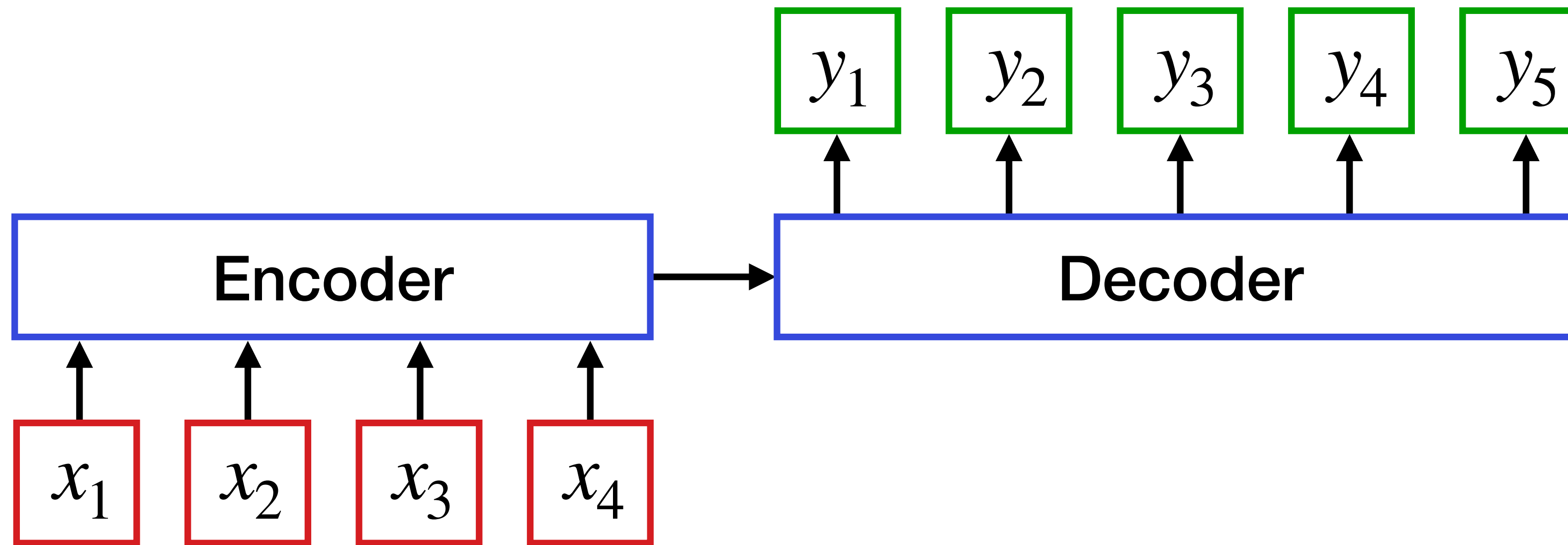
$$p(y_1, \dots, y_n | x_1, \dots, x_m) = \prod_{i=1}^n p(y_i | y_{<i}, x_1, \dots, x_m)$$

x – входная последовательность

y – новая (выходная) последовательность

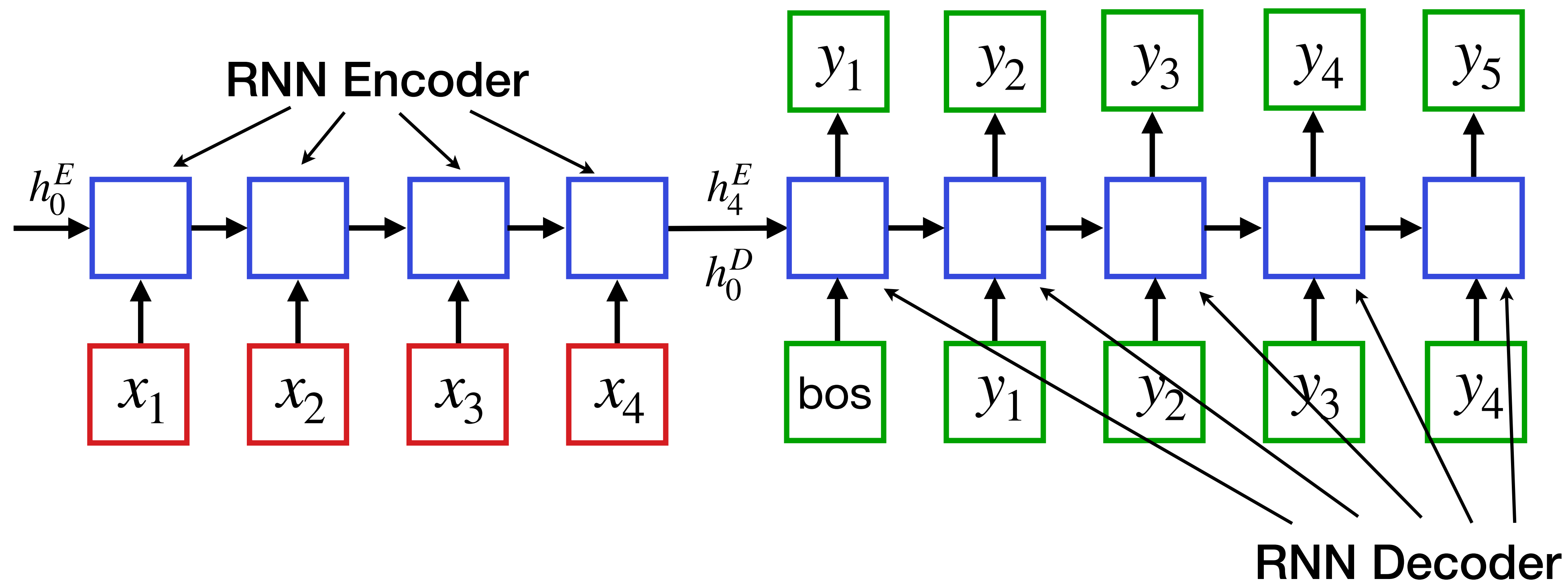
Стандартный подход

- Входная последовательность кодируется моделью **Encoder**
- **Decoder** принимает выход кодировщика и генерирует новую последовательность



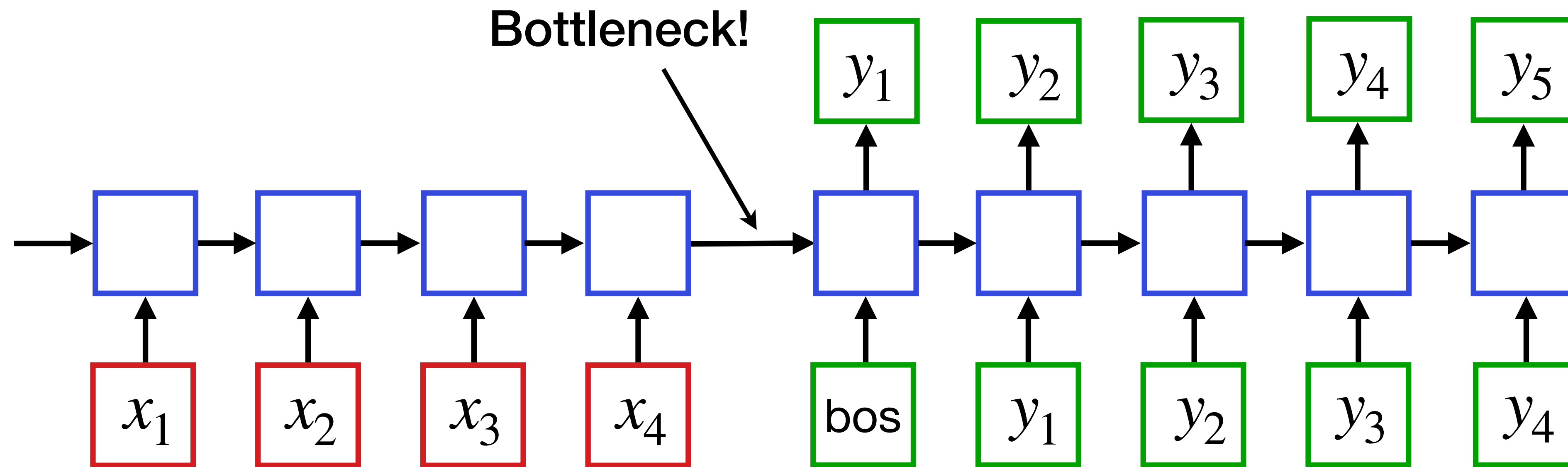
RNN подход

- Входная последовательность кодируется моделью **Encoder**
- **Decoder** принимает выход кодировщика и генерирует новую последовательность



RNN подход: проблема

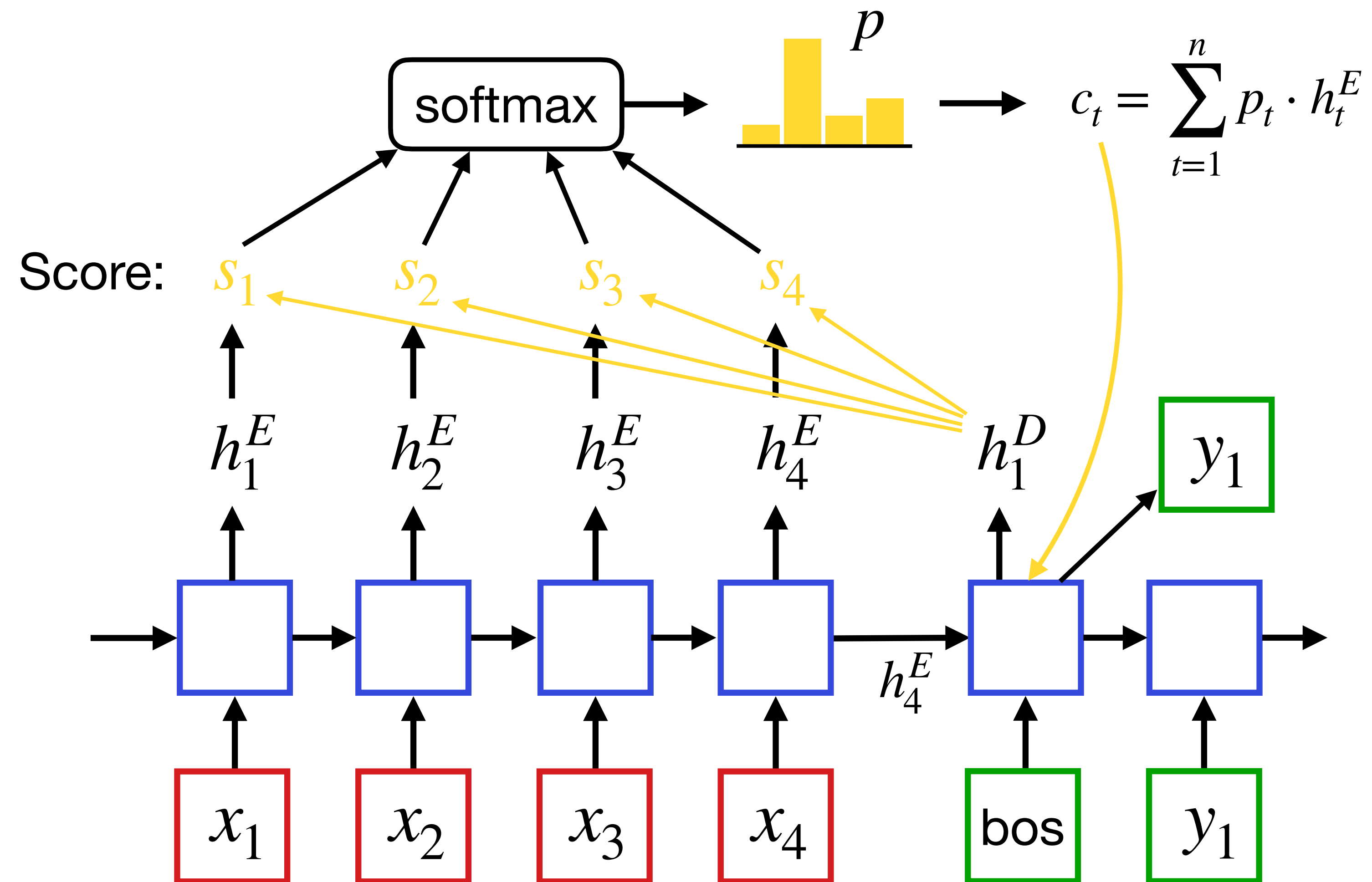
- Входной текст может иметь произвольную длину
- Мы пытаемся уместить всю информацию о нем в один вектор h
- Получаем бутылочное горлышко (bottleneck)



Механизм внимания

Идея:

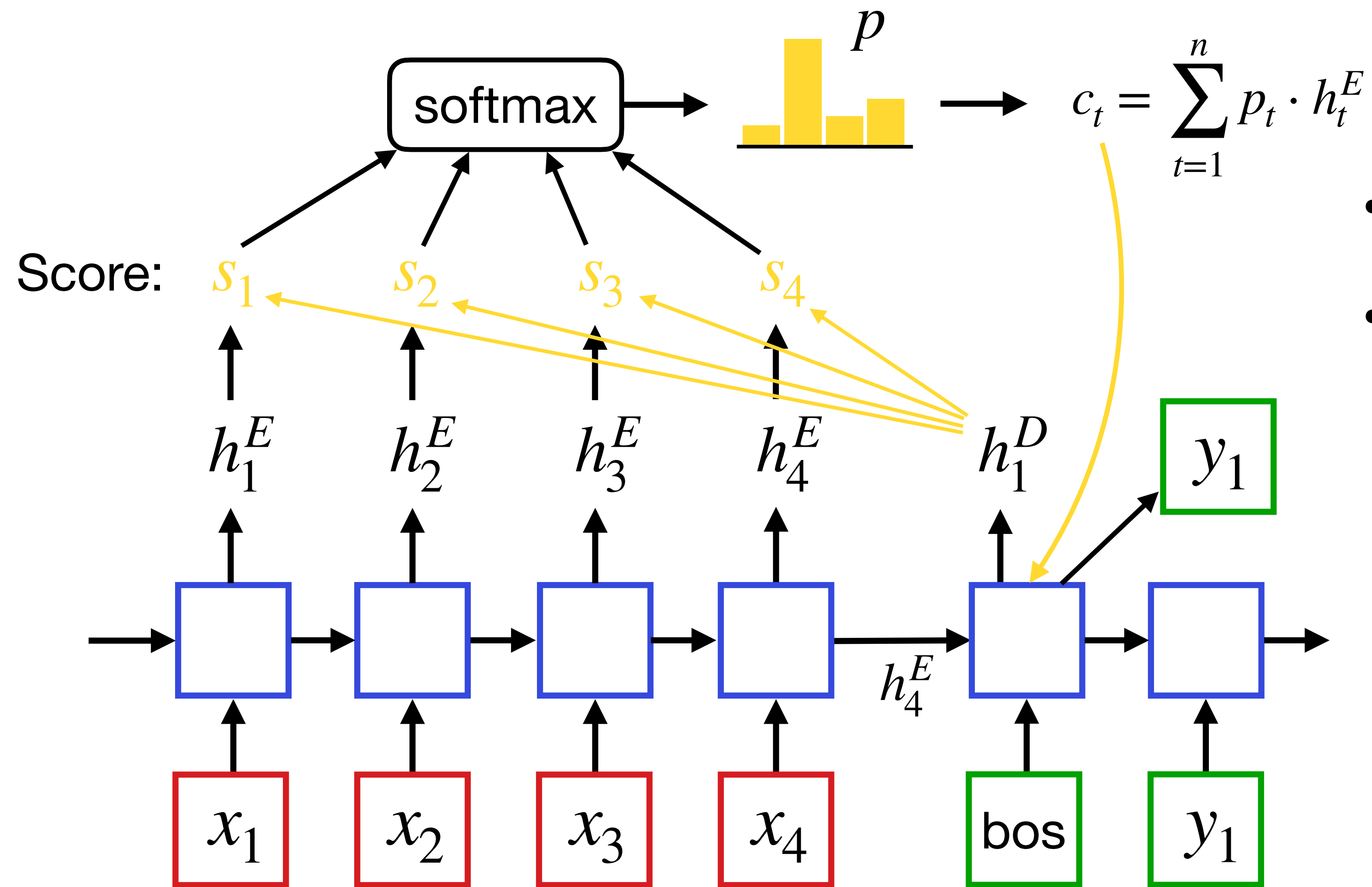
Позволим декодеру "смотреть" на произвольные выходы кодировщика



Механизм внимания

Идея:


Позволим декодеру "смотреть" на произвольные выходы кодировщика



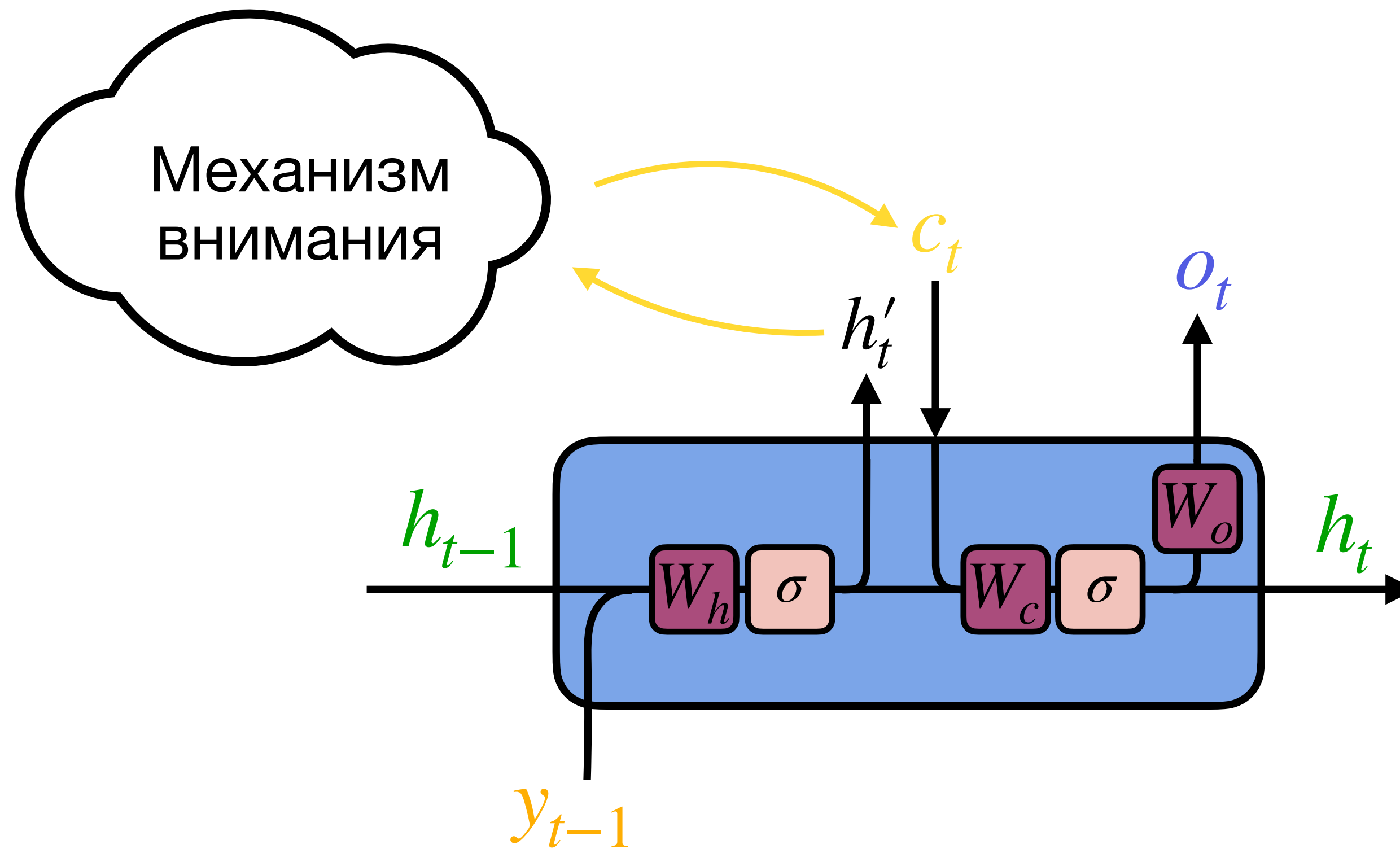
- Что такое Score?
- Как объединяются векторы?

Методы подсчета Score

$$s(h_E, h_D) \in \mathbb{R}$$

1. Скалярное произведение: $s(h_E, h_D) = \langle h_E, h_D \rangle$
2. Билинейная форма: $s(h_E, h_D) = h_E^T W h_D$
 Обучаемая матрица
3. Любая произвольная функция

Блок декодера



$$h'_t = \tanh(W_h[h_{t-1}, y_{t-1}] + b_h)$$

$$h_t = \tanh(W_c[h'_t, c_t] + b_c)$$

$$o_t = W_o h_t + b_o$$

Другие варианты:
[Bahdanau et al, 2014]
[Luong et al, 2015]

Transformer

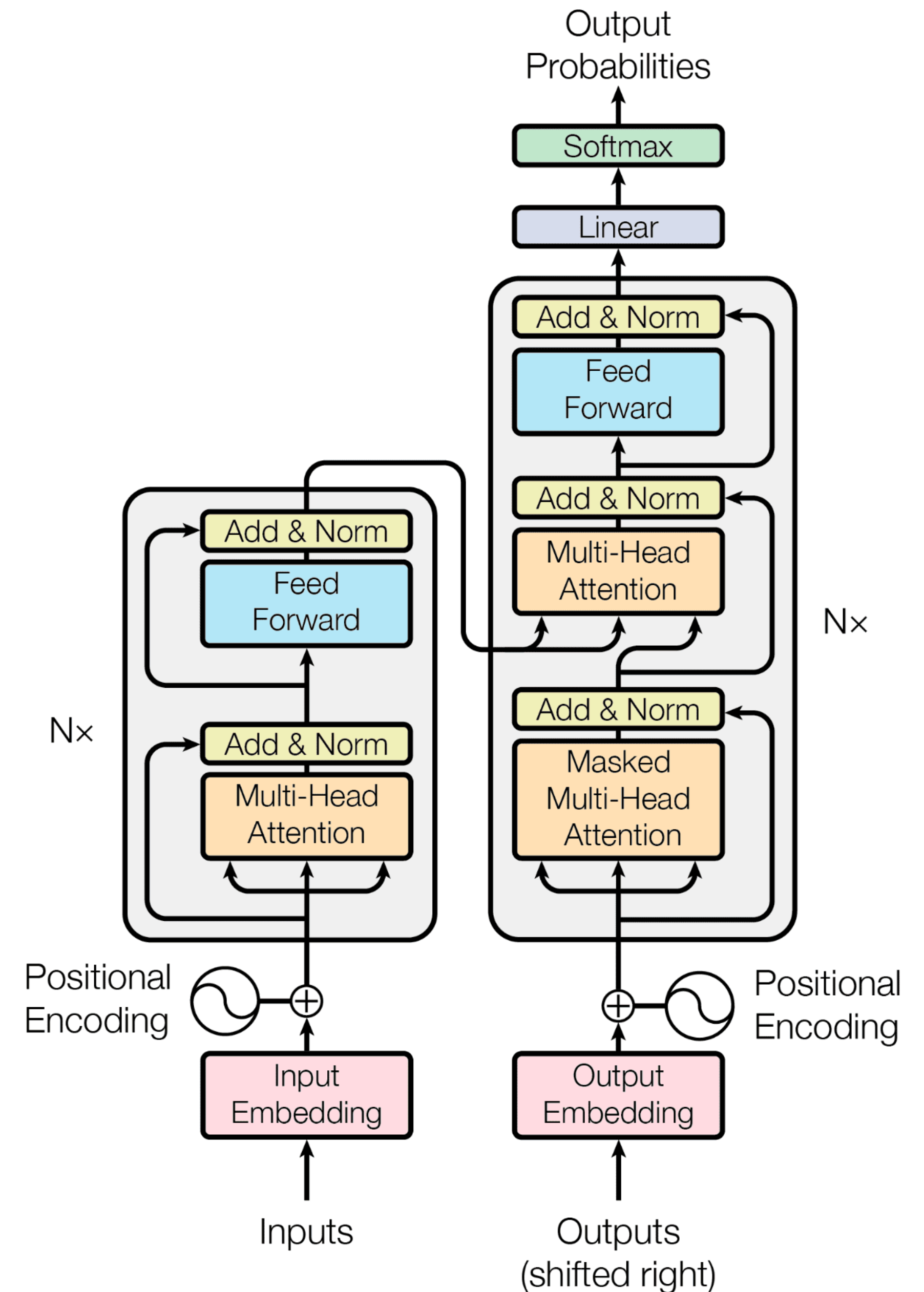
Attention is all you need

- Самая популярная архитектура с 2017 года
- ~200k цитат у оригинальной статьи

Разберемся с тем, как она работает

Компоненты:

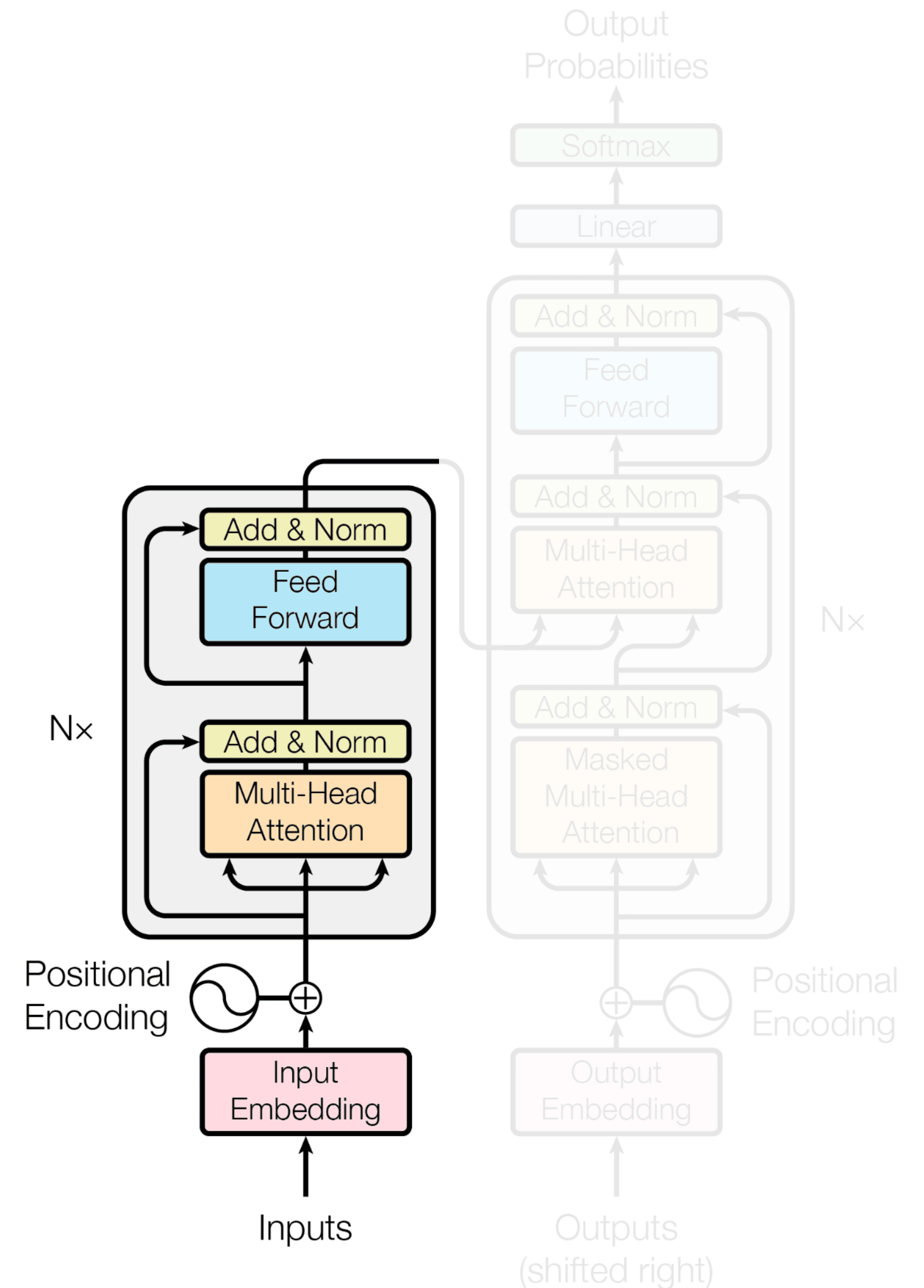
1. Multi-Head Self-attention
2. Feed Forward Network
3. Masked Multi-Head Self-attention
4. Positional Encodings



Encoder

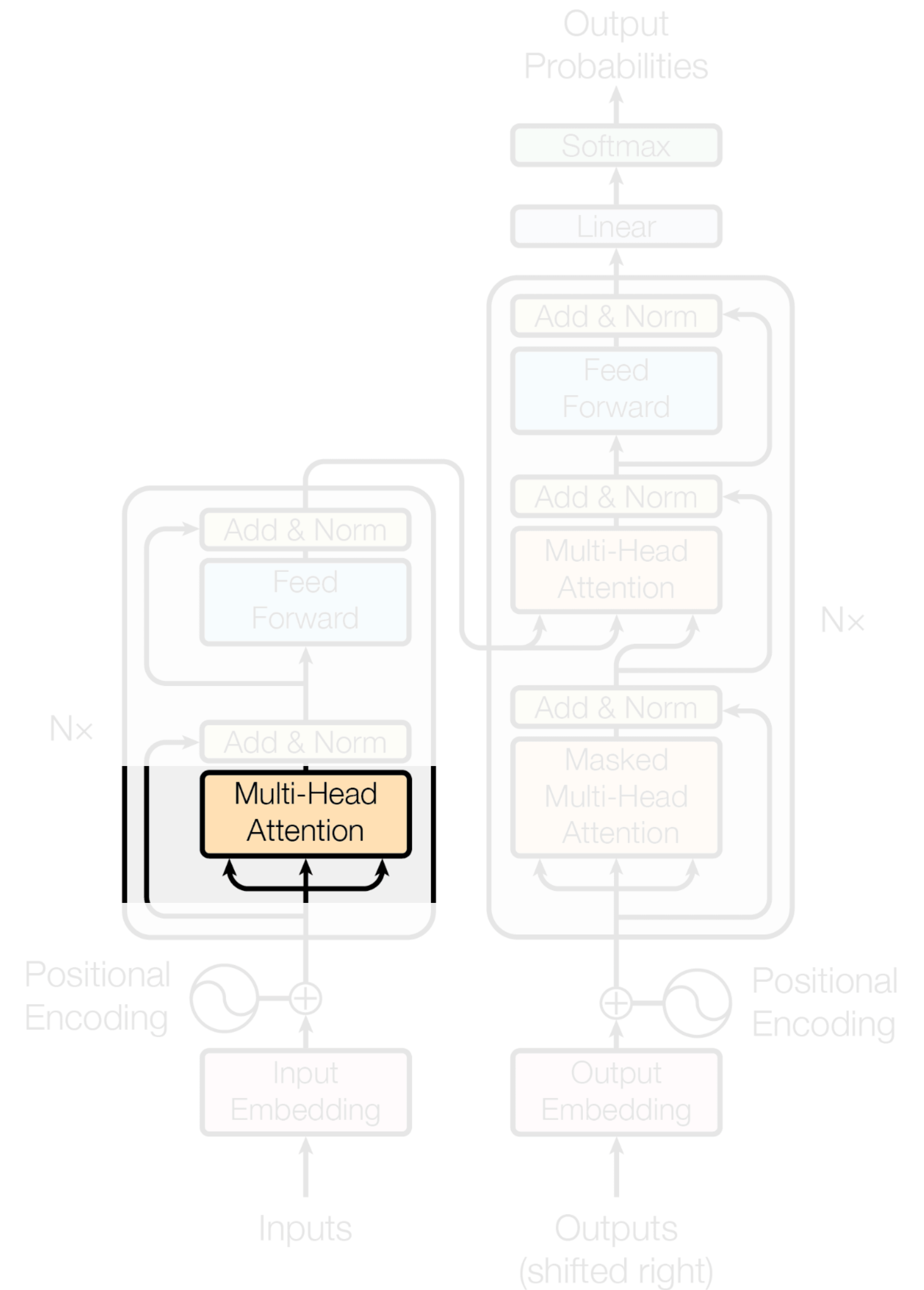
Encoder нужен для извлечения признаков из последовательности

1. Multi-Head Self-attention
2. Positional Encoding
3. Feed Forward Network
4. Add & Norm



Encoder

1. Multi-Head Self-attention
2. Positional Encoding
3. Feed Forward Network
4. Add & Norm



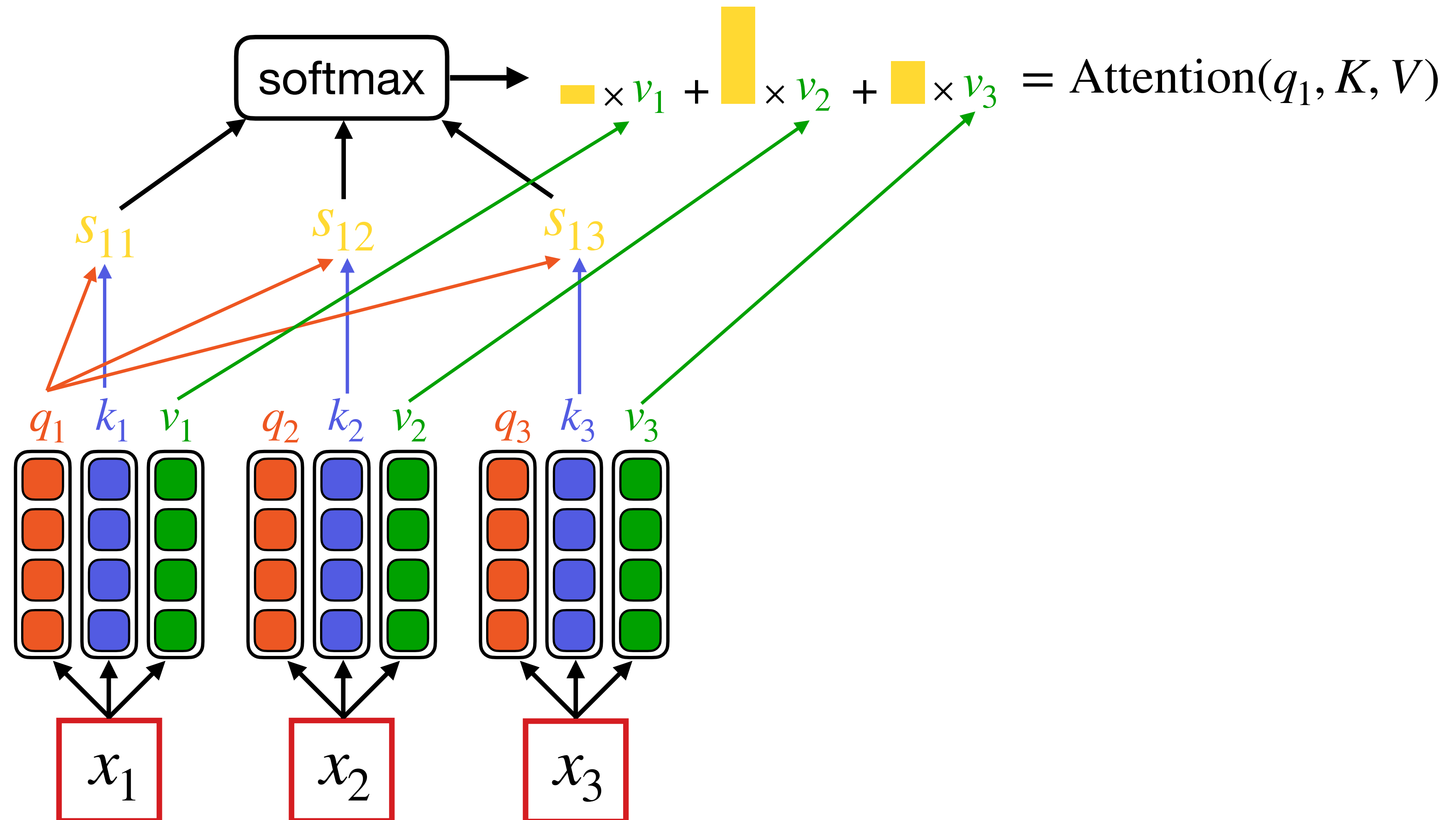
Self-attention

Позволяет каждому токенту посмотреть на все остальные.

q – запрос (query)

k – ключ (key)

v – значение (value)



Self-attention

Позволяет каждому токенту посмотреть на все остальные.

Query: $Q = W_q x + b_q$

Key: $K = W_k x + b_k$

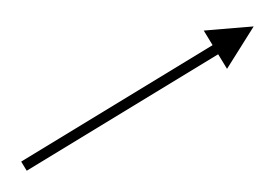
Value: $V = W_v x + b_v$

$$x \in \mathbb{R}^{n \times d}$$

$$W_q, W_k, W_v \in \mathbb{R}^{d \times d}$$

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

нормировочная
константа



Multi-Head Self-attention

Self-attention позволяет запрашивать только информацию одного вида.
А что если мы хотим узнать разные аспекты?

На улице ярко светит солнце

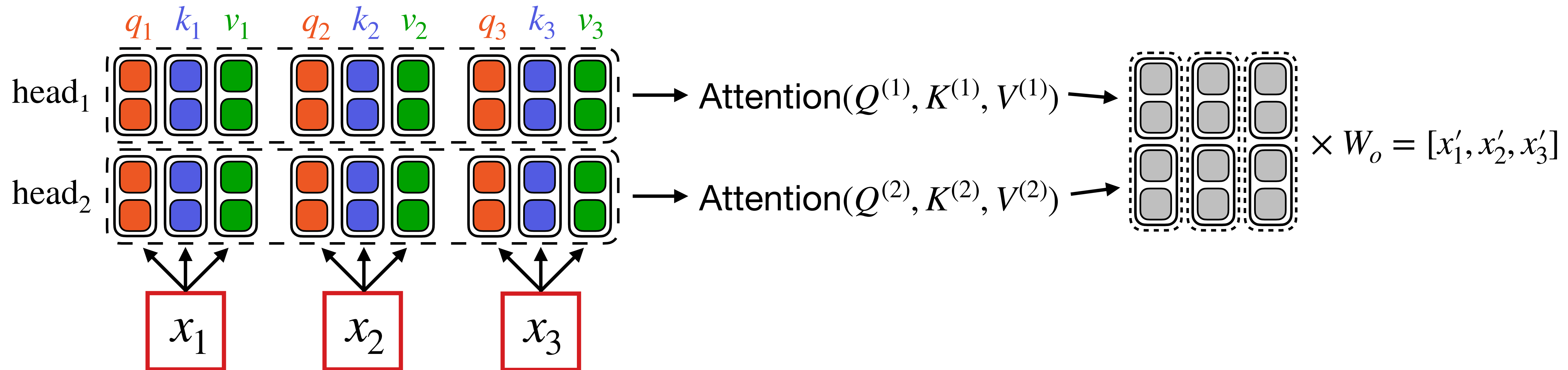


как? что?

Разделим внимание на несколько голов

Multi-Head Self-attention

- Делим каждый вектор q , k , v на равные части
- Считаем self-attention для каждой части отдельно
- Конкатенируем результат
- Домножаем на выходную матрицу



Multi-Head Self-attention

Query: $Q^{(i)} = W_q^{(i)}x + b_q^{(i)}$

Key: $K^{(i)} = W_k^{(i)}x + b_k^{(i)}$

Value: $V^{(i)} = W_v^{(i)}x + b_v^{(i)}$

$$\text{head}^{(i)} = \text{Attention}(Q^{(i)}, K^{(i)}, V^{(i)})$$

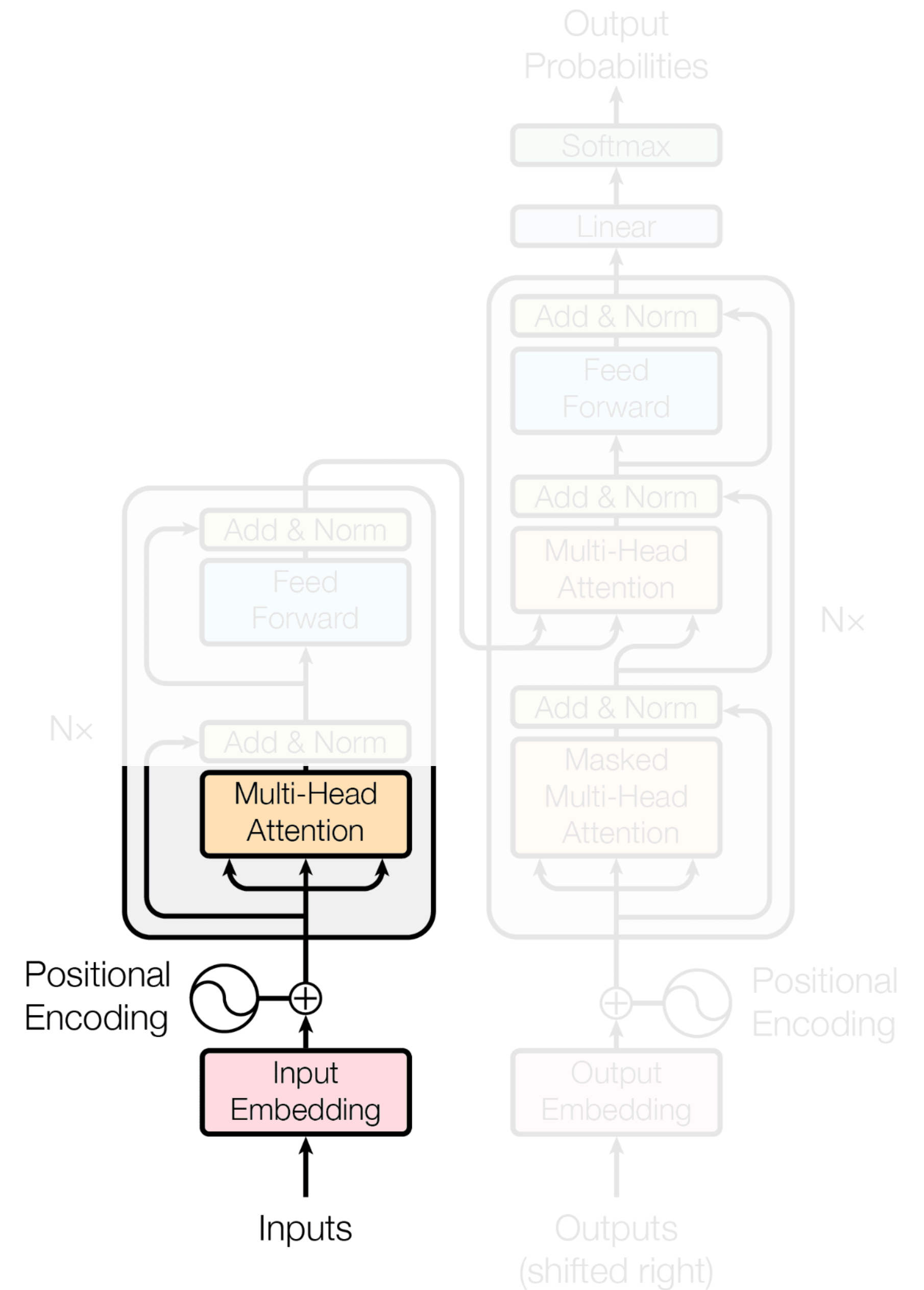
$$\text{MultiHead}(Q, K, V) = W_o \times [\text{head}^{(1)}, \dots, \text{head}^{(h)}]$$

$$W_q^{(i)}, W_k^{(i)}, W_v^{(i)} \in \mathbb{R}^{d_{\text{head}} \times d}$$

$$W_o \in \mathbb{R}^{d \times h \cdot d_{\text{head}}}$$

Encoder

1. Multi-Head Self-attention
2. **Positional Encoding**
3. Feed Forward Network
4. Add & Norm



Self-attention не учитывает позиции

При перестановке токенов местами выход не изменится!

$$w = \text{softmax} \left(\frac{q_i K^T}{\sqrt{d}} \right)$$

$$\text{Attention}(q_i, K, V) = w_1 V_1 + \dots + w_n V_n$$

=> Не сможем различить два случая

очень хорошо, совсем **не** плохо

vs

не очень хорошо, совсем плохо

Positional Encoding

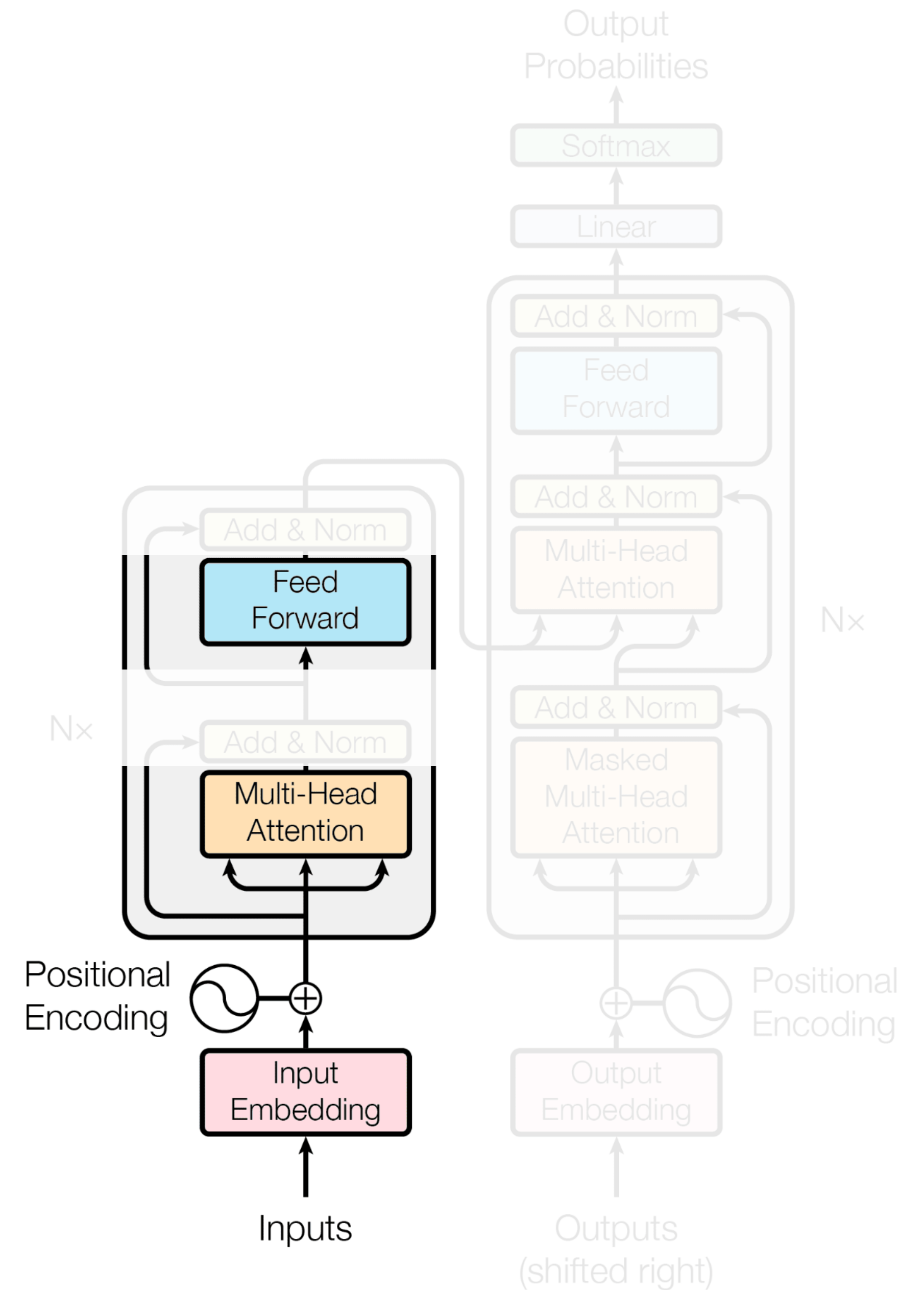
Необходимы для учета позиций токенов в тексте.

$$x_i = \text{Emb}(w_i) + \text{Emb}_{pos}(i)$$

Эмбеддинги позиций учатся вместе с остальными параметрами модели.

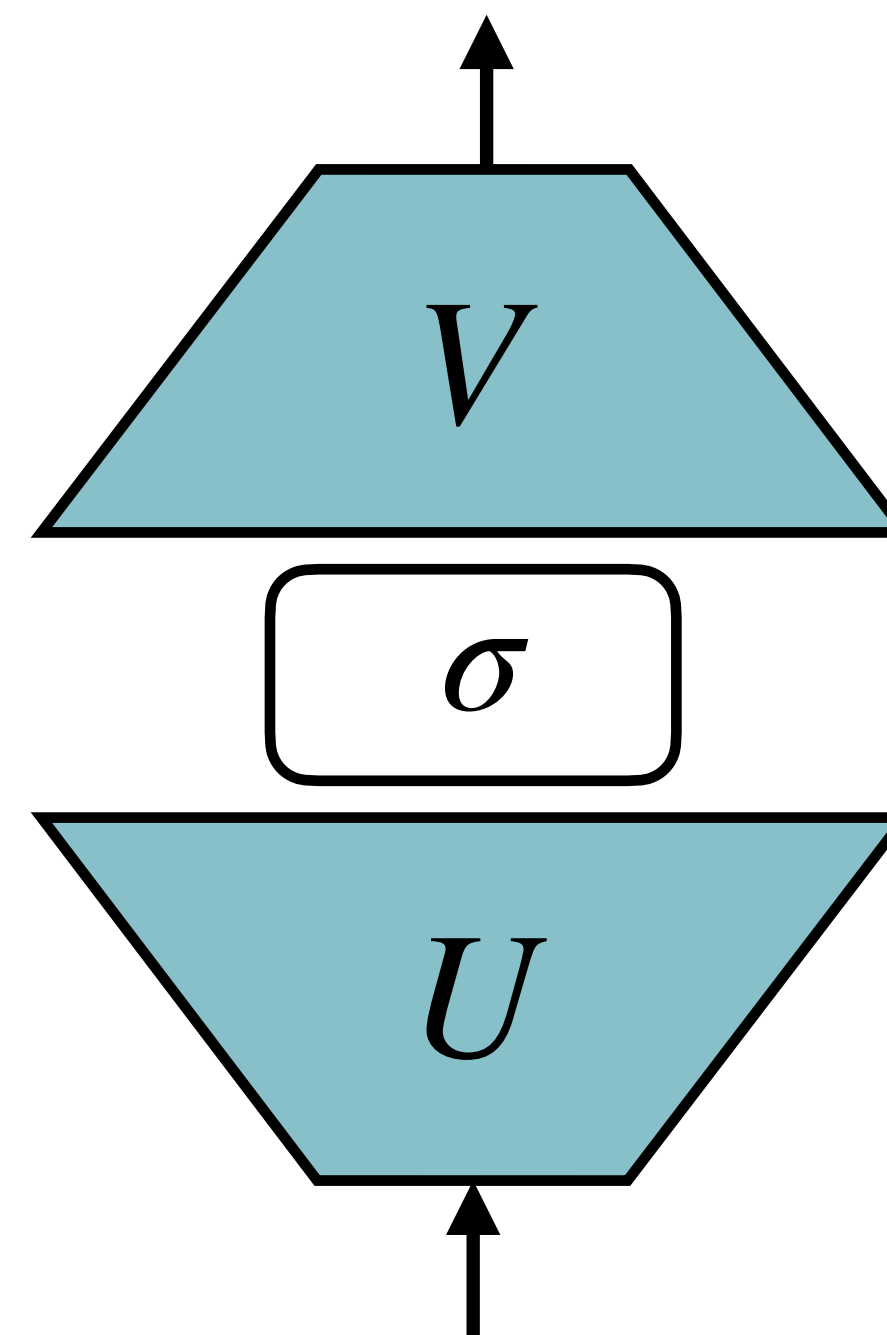
Encoder

1. Multi-Head Self-attention
2. Positional Encoding
3. **Feed Forward Network**
4. Add & Norm



Feed Forward Network

- Полносвязная сеть из двух слоев.
- Первый слой увеличивает размерность, второй – возвращает обратно
- Используется для обработки информации, полученной после self-attention



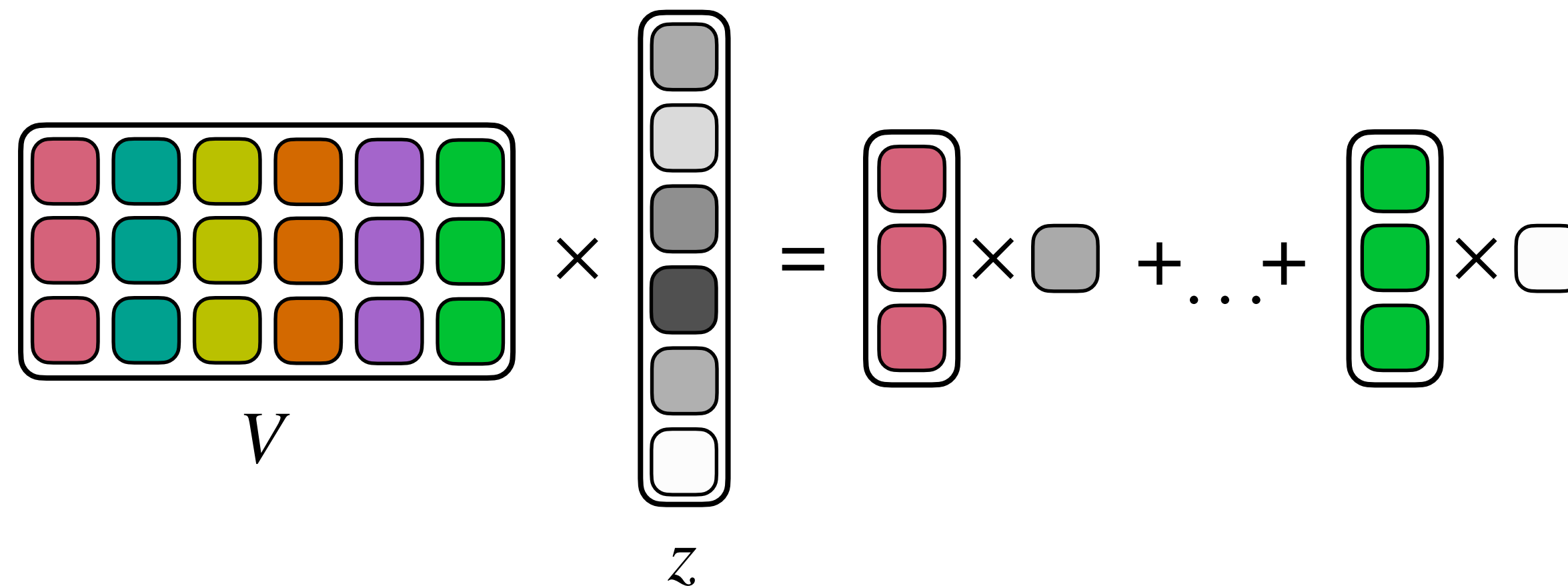
$$U \in \mathbb{R}^{2d \times d}$$

$$V \in \mathbb{R}^{d \times 2d}$$

FFN: Как ещё можно думать?

- FFN играет роль базы данных!
- U содержит ключи, а V – значения

$$\sigma(Ux_i) = z \in \mathbb{R}^{2d}$$

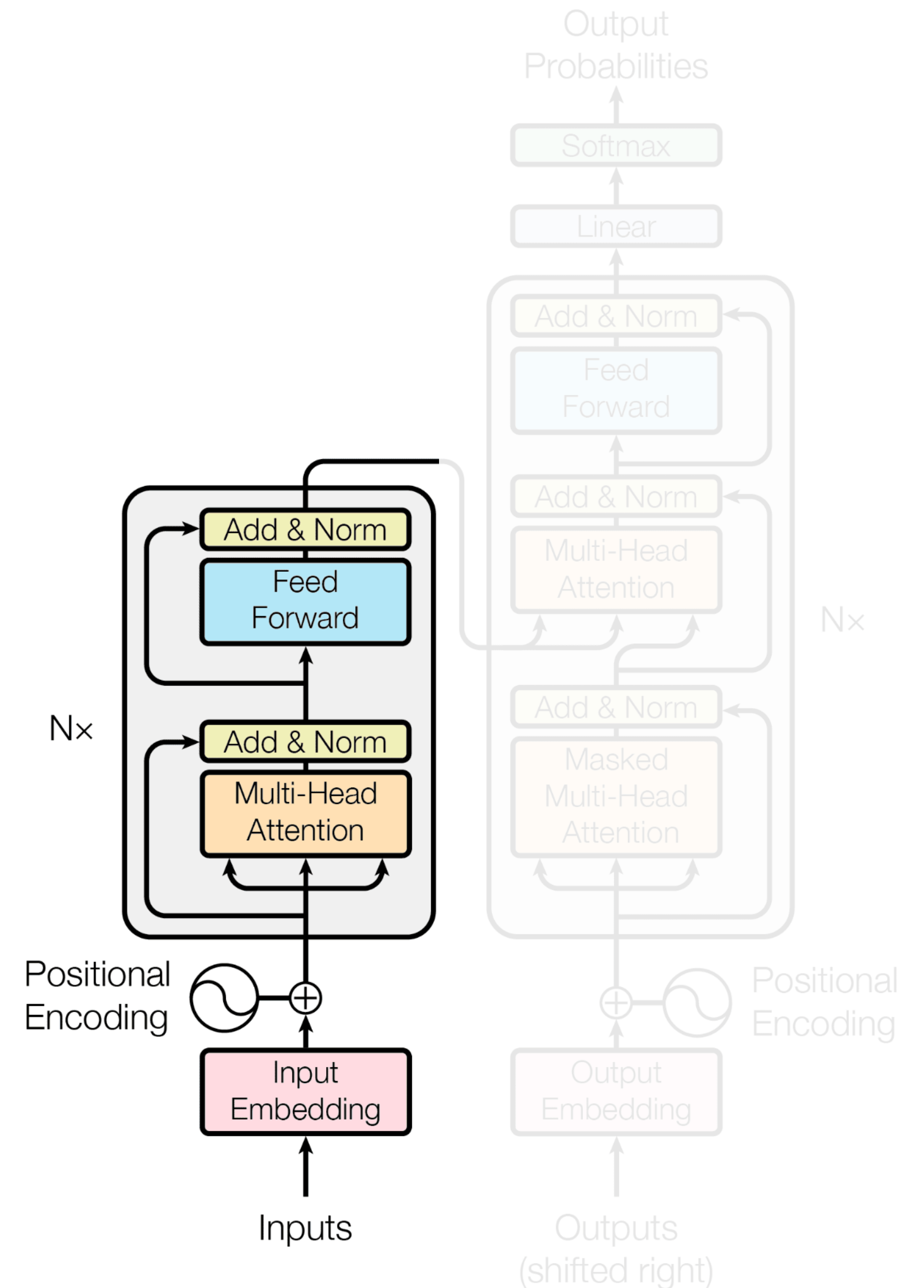


- Столбцы V – определенные факты
- Мы взвешиваем эти факты согласно ключу z

Encoder

1. Multi-Head Self-attention
2. Positional Encoding
3. Feed Forward Network
4. **Add & Norm**

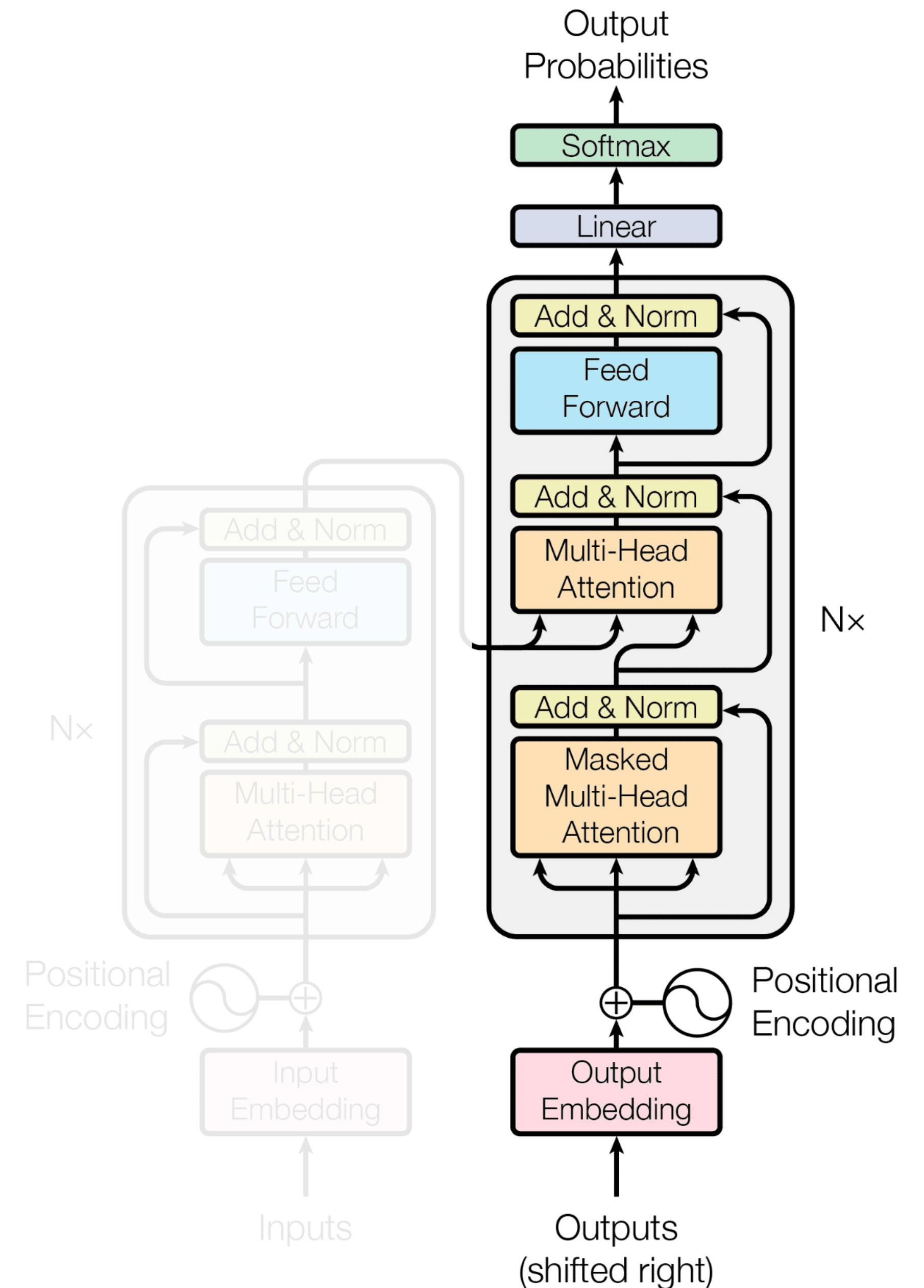
- Добавляем **skip-connection**, чтобы градиенты не затухали
- **Нормализация** для стабилизации обучения



Decoder

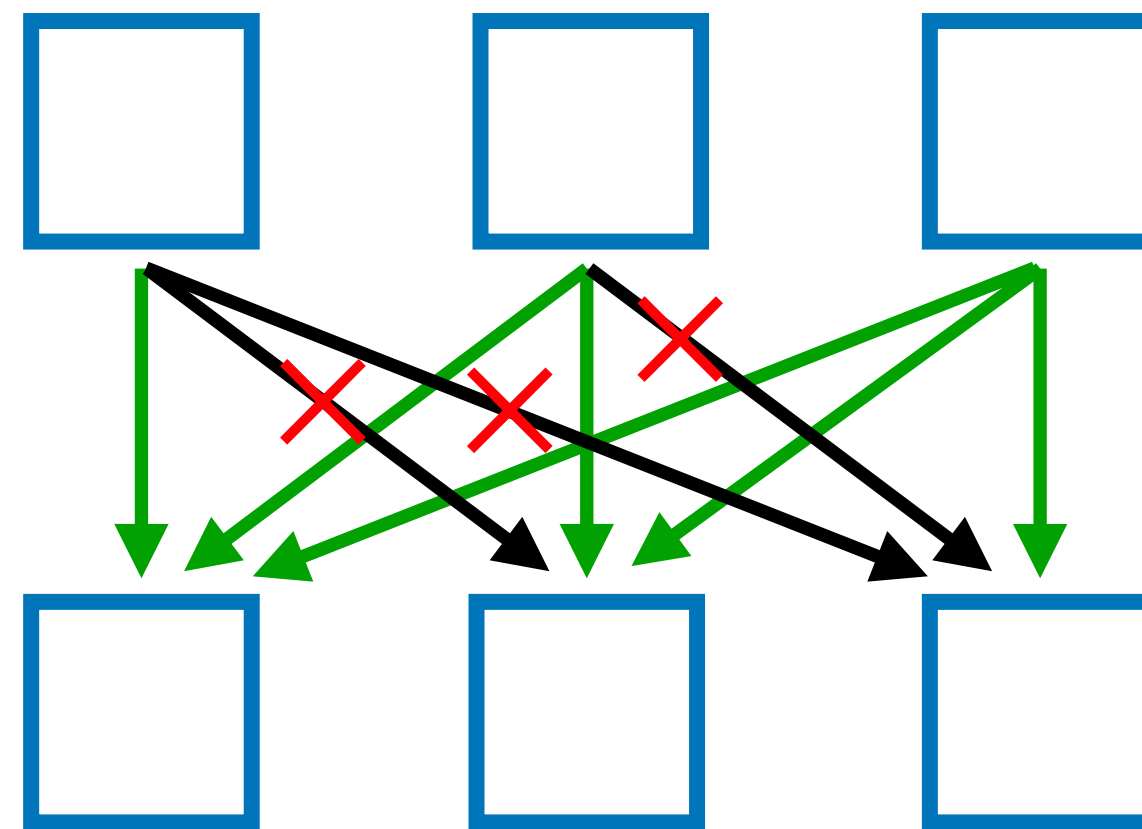
Decoder нужен для генерации
ВЫХОДНОГО ТЕКСТА

1. Masked Multi-Head Self-attention
2. Cross Attention



Masked Self-attention

- При обучении генерации мы должны запретить декодеру смотреть на токены, идущие после текущего
- Декодер учитывает другие токены только в слое **self-attention**
- => Необходимо его модифицировать



Каждый охотник желает

Masked Self-attention

- Добавляем маску в подсчет внимания
- Маска явно запрещает смотреть вперед

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}} + M\right) V$$

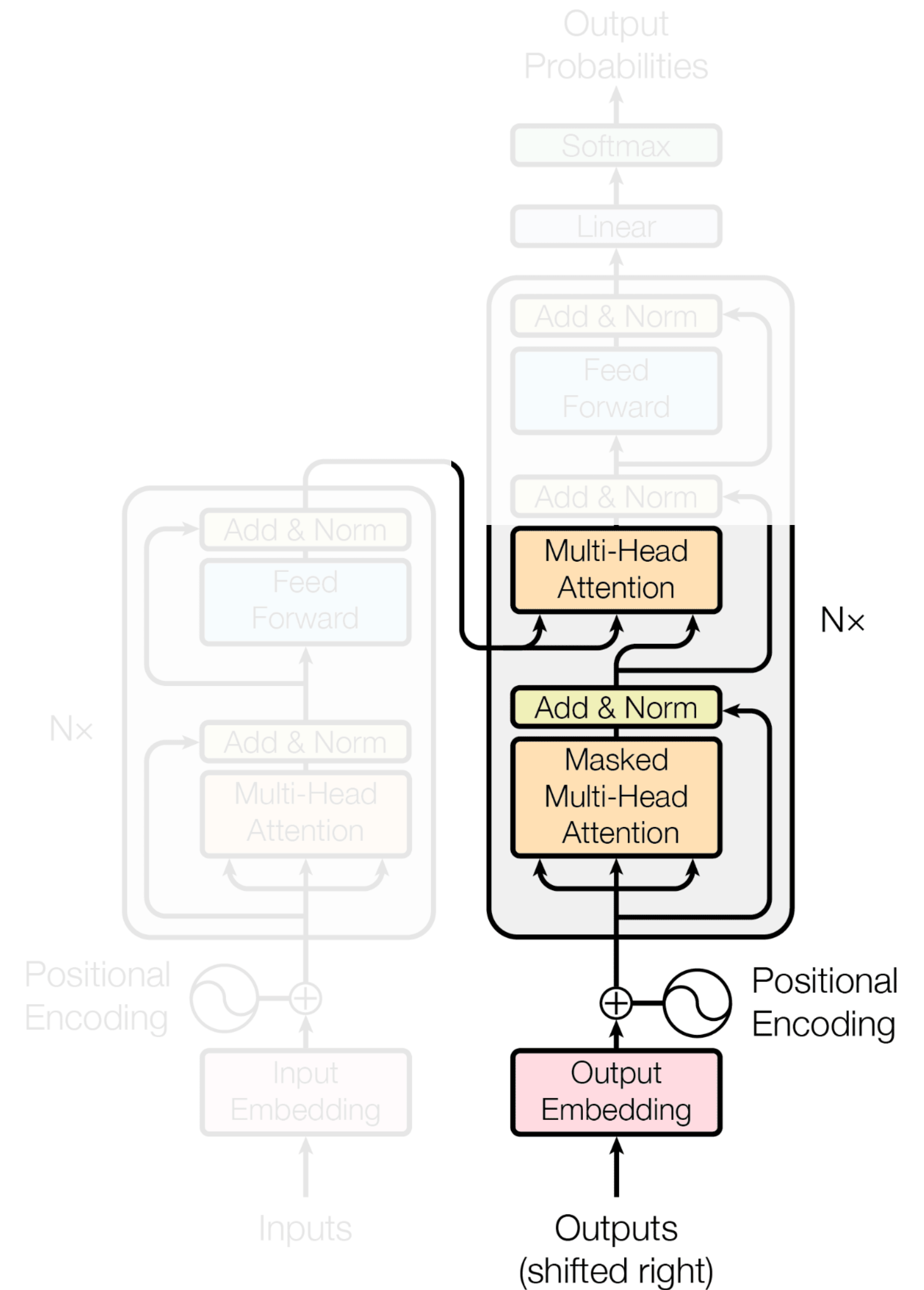
$$M_{ij} = \begin{cases} 0, & i \leq j \\ -\infty, & i > j \end{cases}$$

$M =$

0	$-\infty$	$-\infty$	$-\infty$
0	0	$-\infty$	$-\infty$
0	0	0	$-\infty$
0	0	0	0

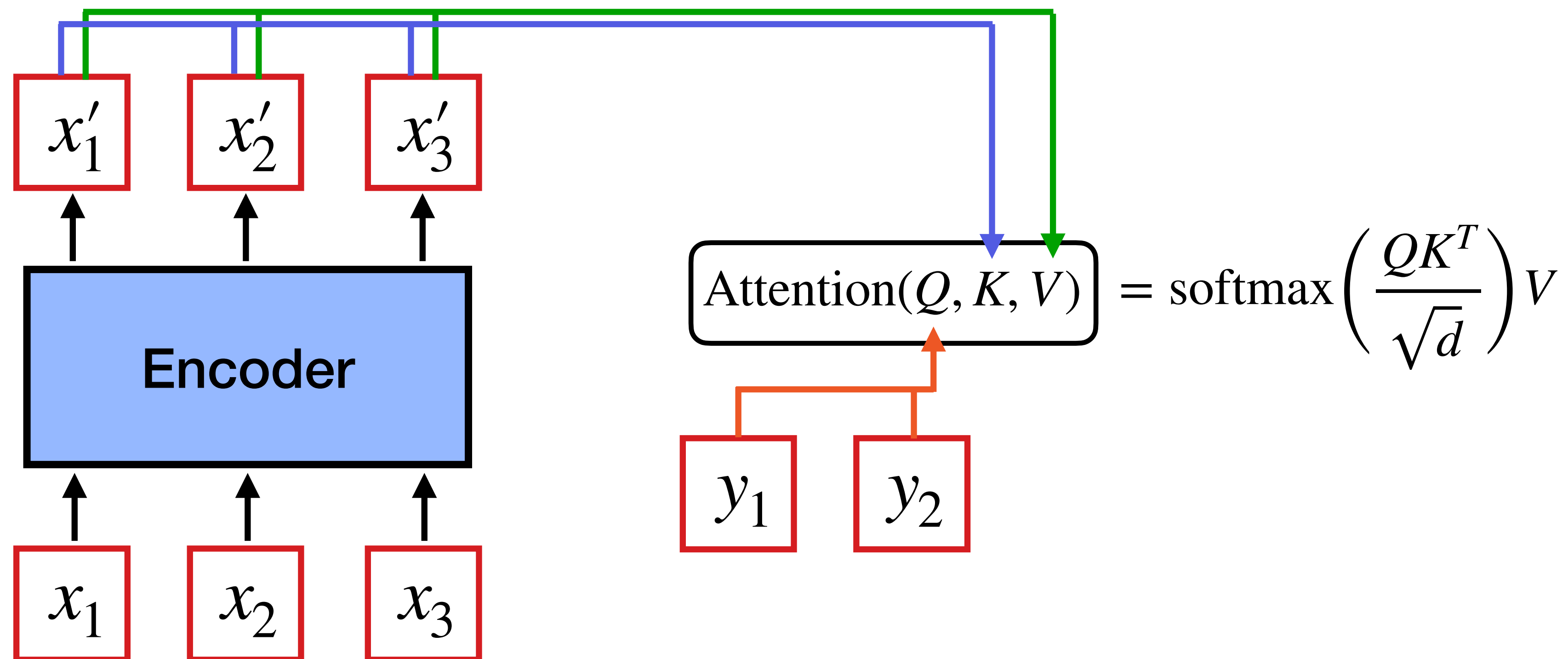
Decoder

1. Masked Multi-Head Self-attention
2. **Cross Attention**



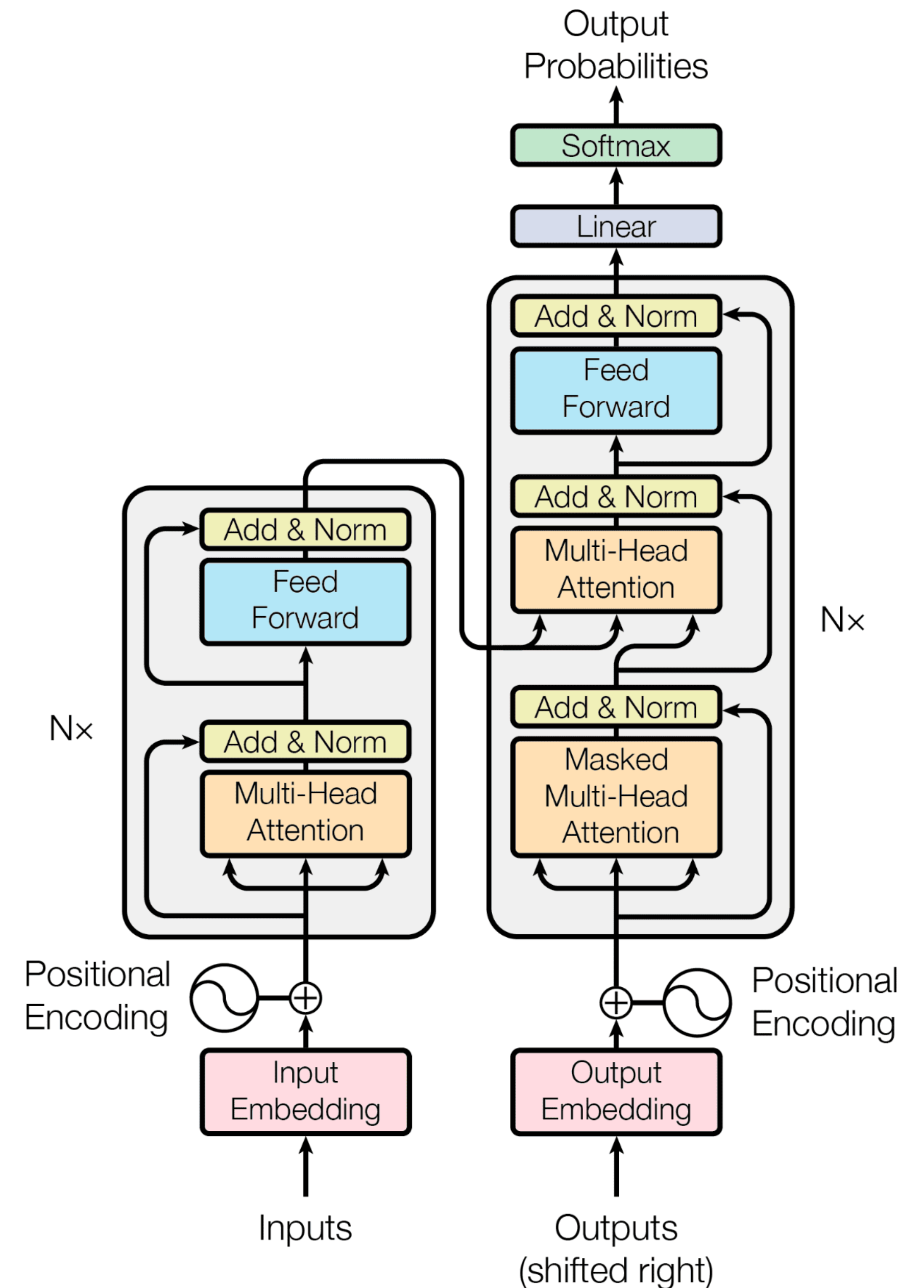
Cross Attention

- Используется для передачи информации от Encoder в Decoder
- Decoder "запрашивает" информацию, передавая **query** в attention
- Encoder передает **key** и **value**



Итого

- Трансформер создан специально для задач Seq2seq
- Блок **Encoder** состоит из:
 - Multi-Head Attention
 - Feed Forward Network
- После каждого слоя идет skip-connection и нормализация
- Блок **Decoder** использует
 - **Masked** Multi-Head Attention, чтобы не смотреть в будущее
 - **Cross-Attention**, чтобы брать информацию из Encoder
 - На выходе декодер предсказывает следующий токен



Методы семплирования токенов

Модель выдает распределение вероятностей токенов



Как выбрать токен из этого распределения?

Жадное семплирование

Greedy Sampling


Можно всегда выбирать токен с максимальной вероятностью

$$y_t = \operatorname{argmax}_{y \in V} p(y | y_{<t})$$

Плюсы:

- Максимизируем вероятность текста

Минусы:

- Теряем разнообразие текста
 - Плохо, если генерация **безусловная**
 - Не страшно, если **условная** (seq2seq)
- 

Beam Search

Лучевой поиск

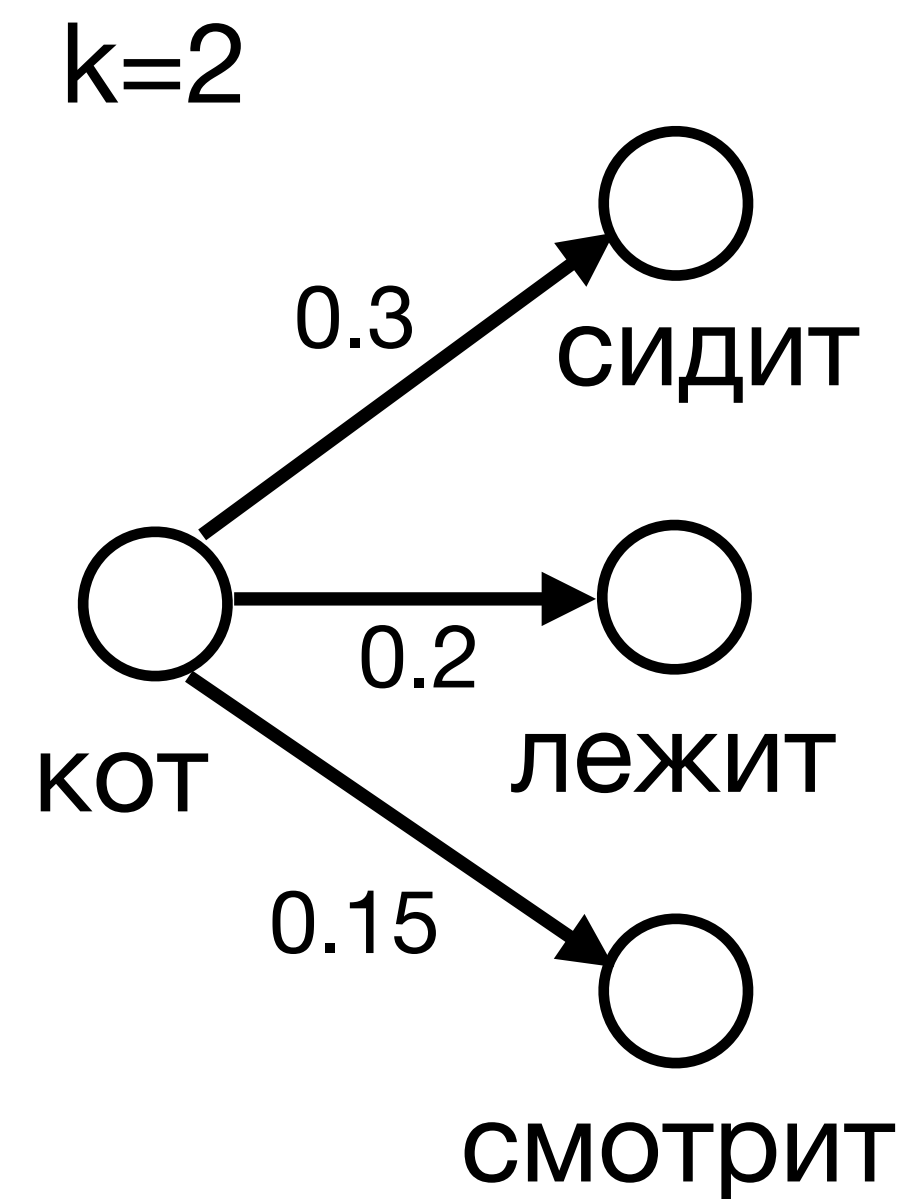
- Попробуем максимизировать вероятность текста еще больше
- При жадном семплировании мы **не учитываем** влияние токена на следующие за ним
- Будем строить предсказания на несколько токенов вперед, а затем выбирать токен с наибольшей совокупной вероятностью

Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

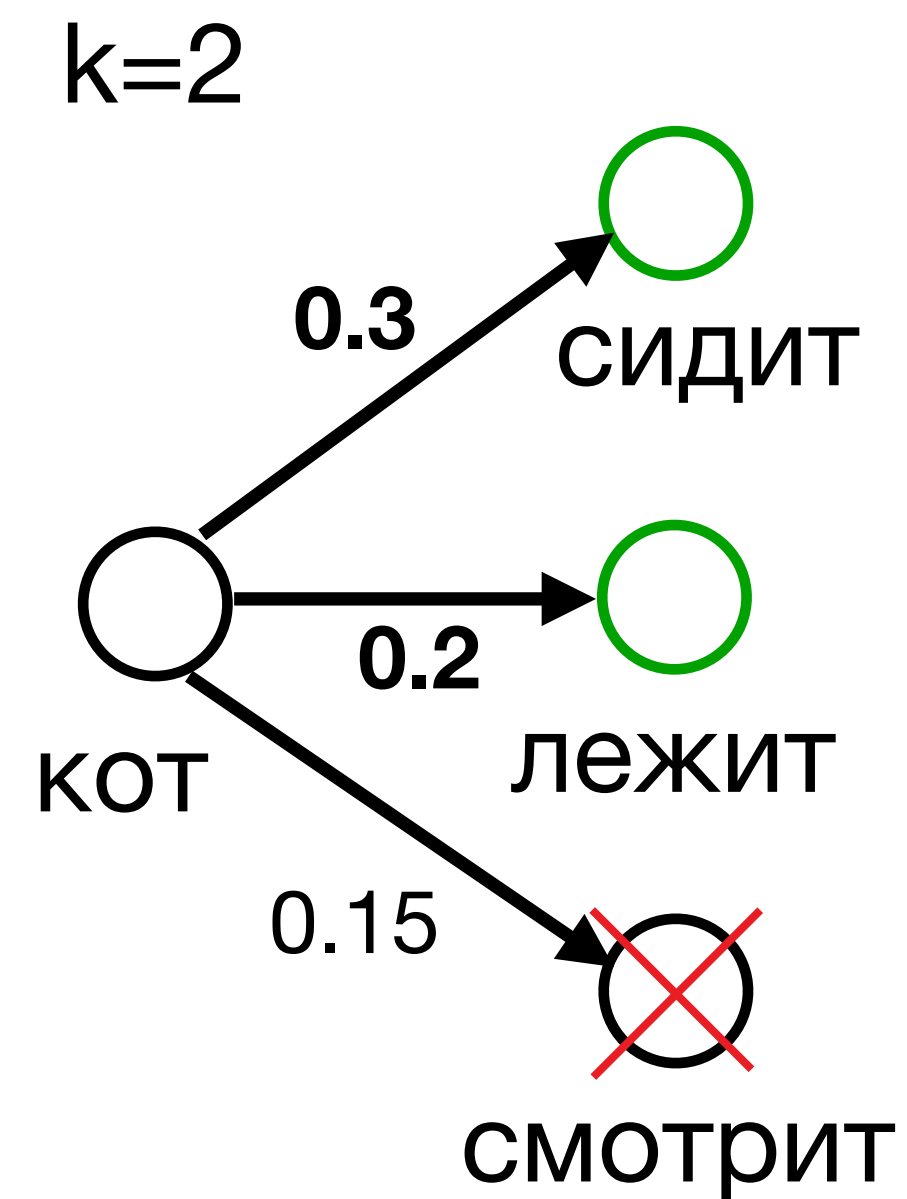


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

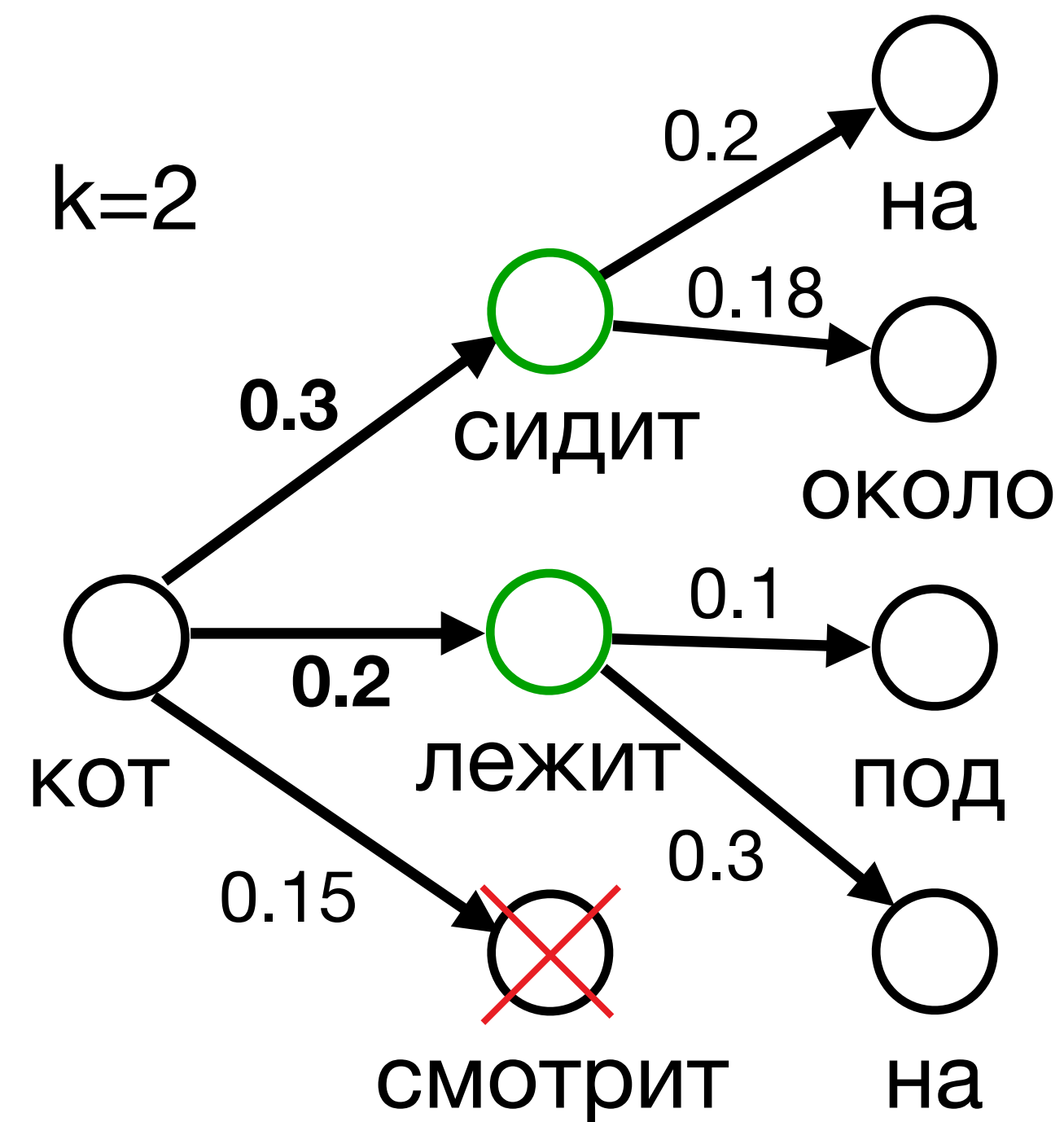


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

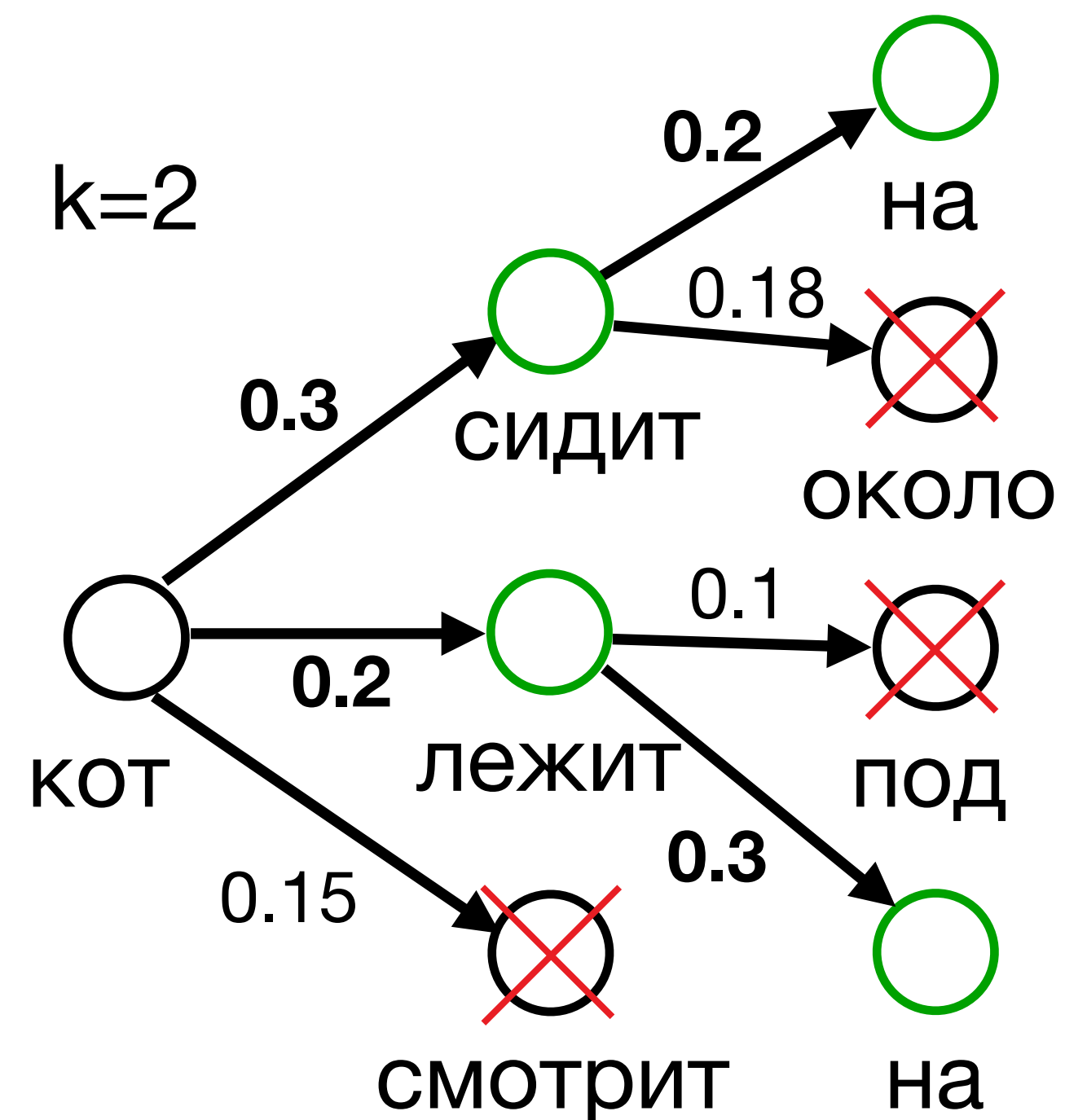


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

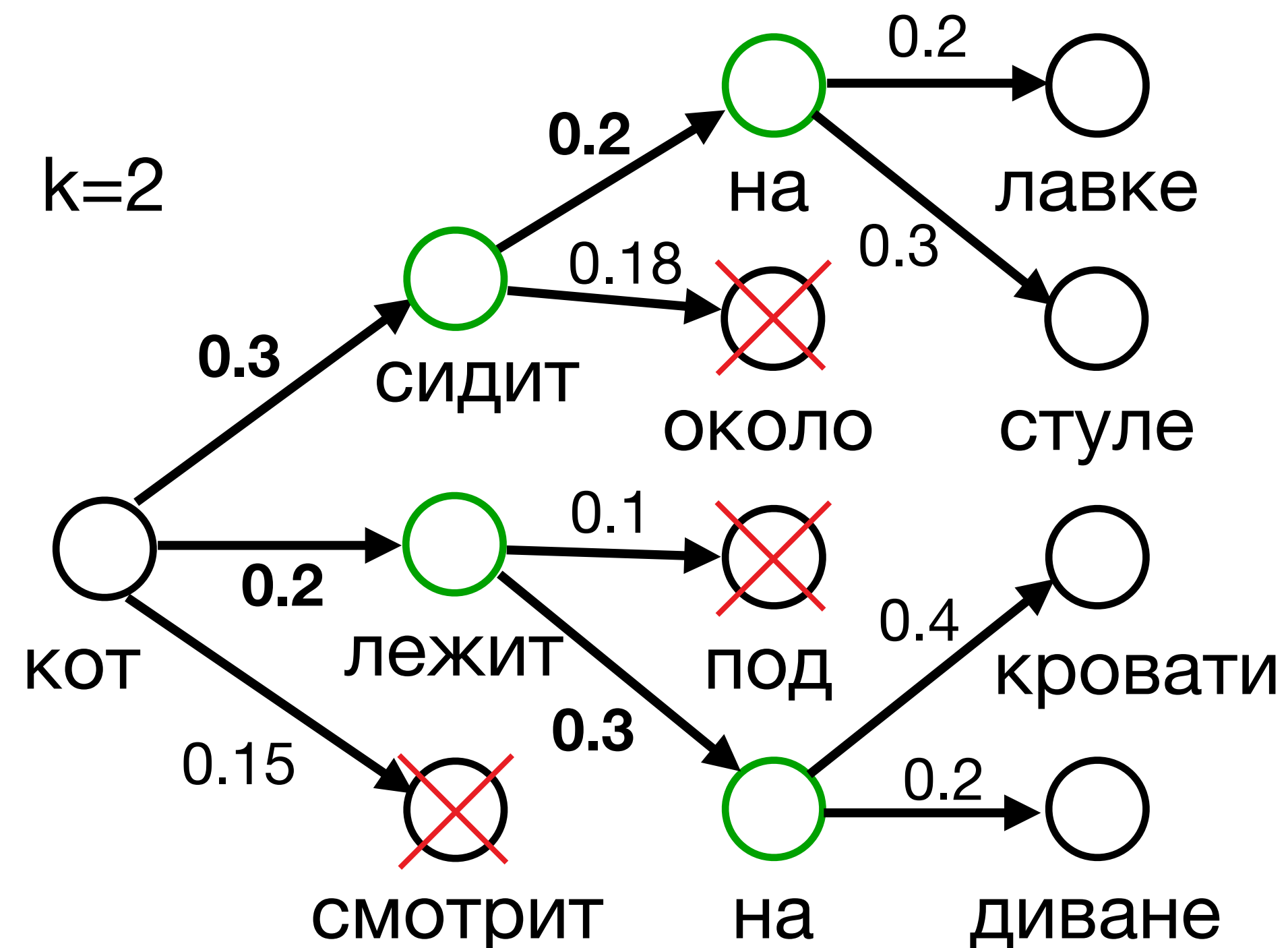


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных

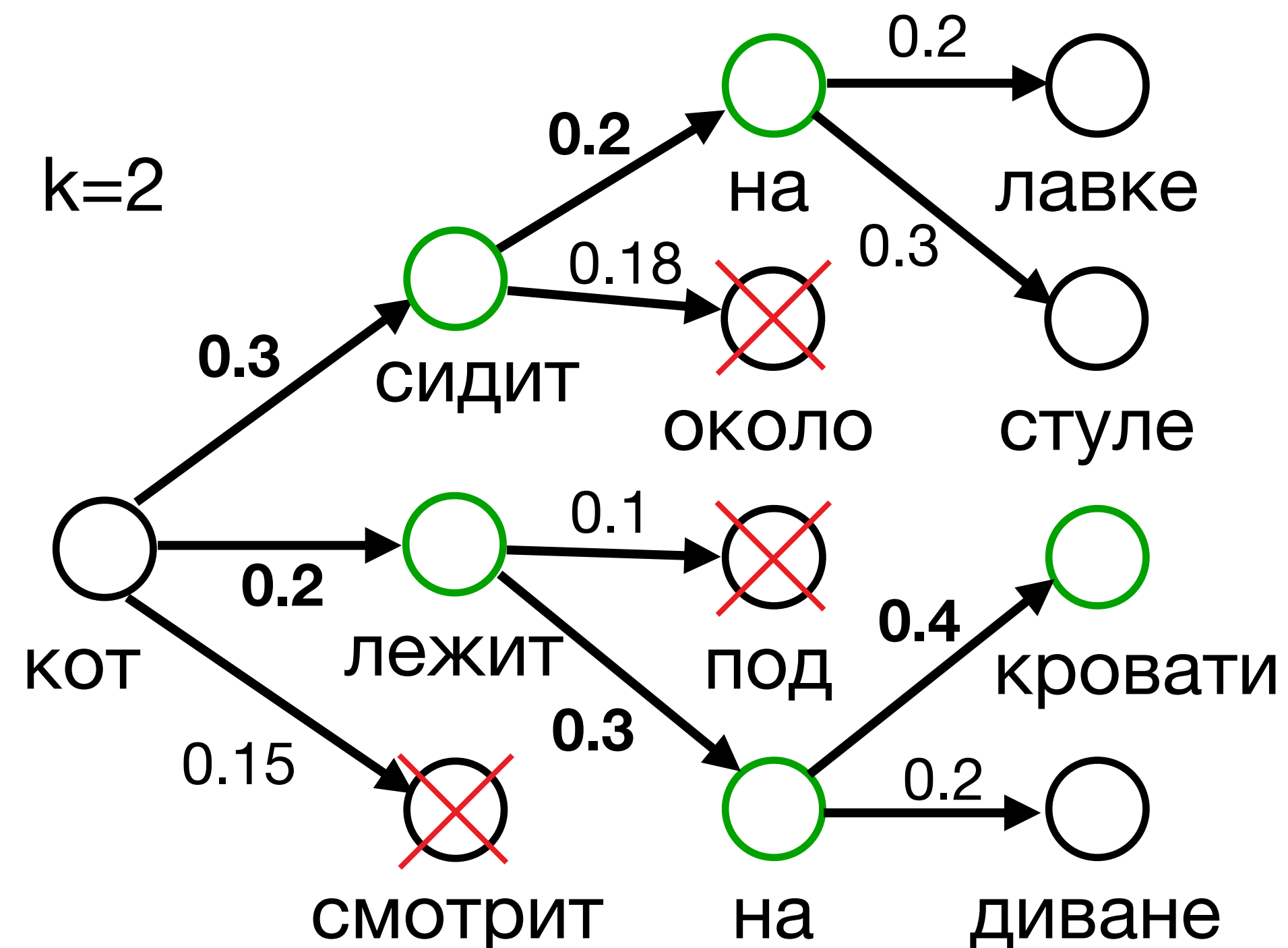


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

- На каждом шаге генерируем k продолжений для каждой траектории
- Оставляем только k самых вероятных



Траектория с "лежит" имеет максимальную вероятность

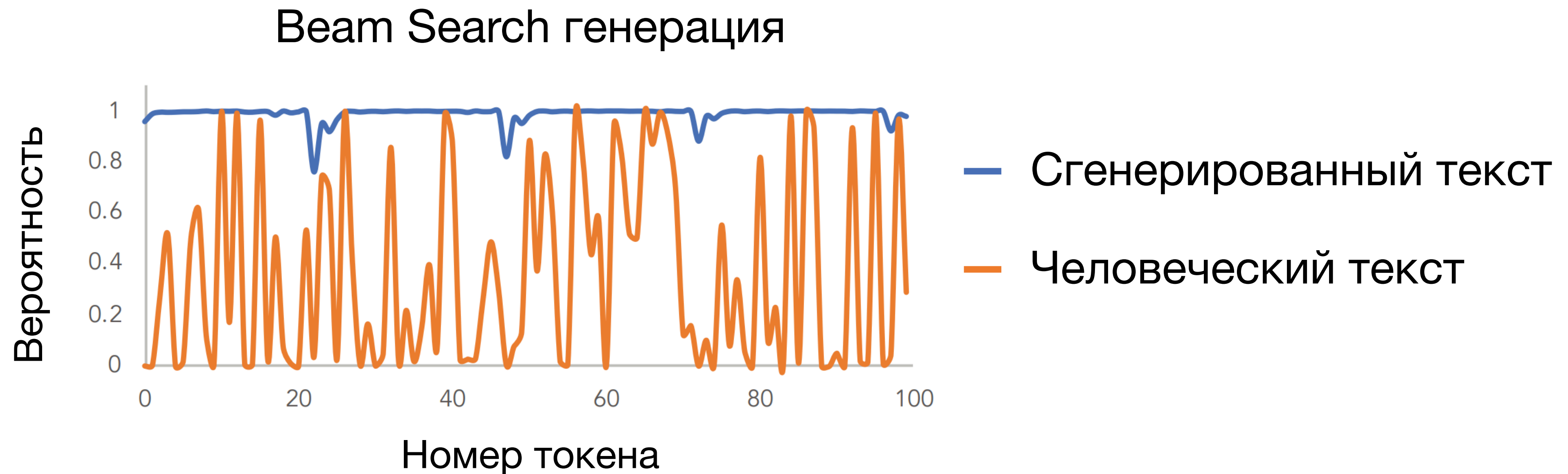
=> выбираем "лежит"

Beam Search vs Жадное семплирование

- Beam Search работает лучше для всех seq2seq задач
- Beam Search работает гораздо медленнее (зависит от k)
- Популярные значения k – 3 или 5

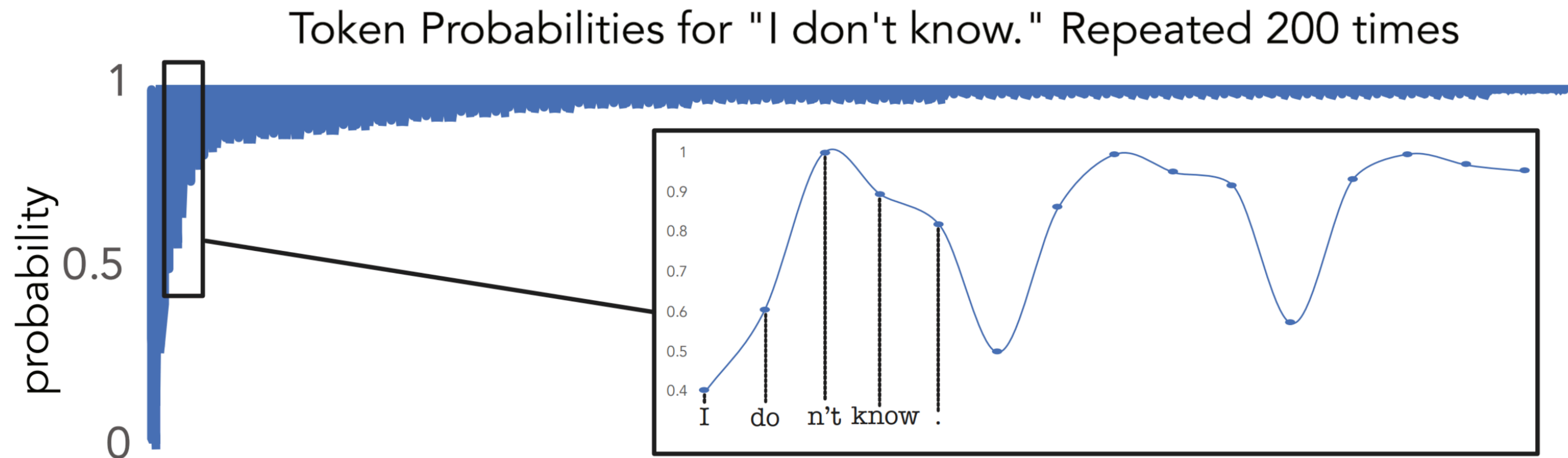
Безусловная генерация

Оба метода уменьшают разнообразие безусловной генерации!



Безусловная генерация

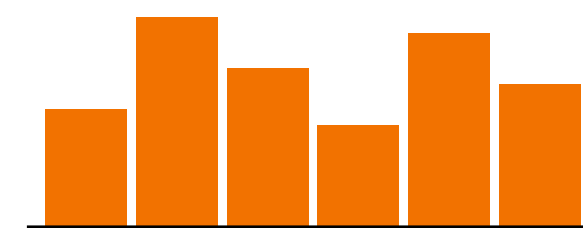
Оба метода поощряют повторения!



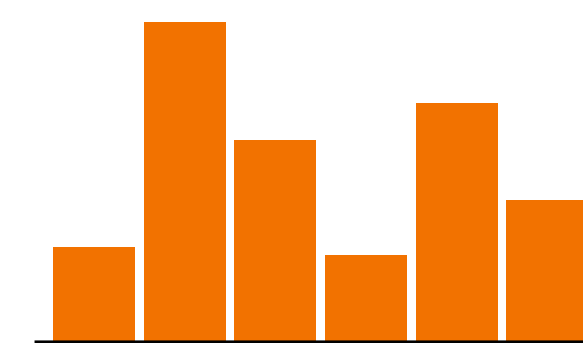
Семплирование с температурой

- При обычном семплировании с вероятностями вероятностная масса случайных токенов слишком велика
- Сделаем распределение более вырожденным, добавив температуру $\tau \in [0,1]$

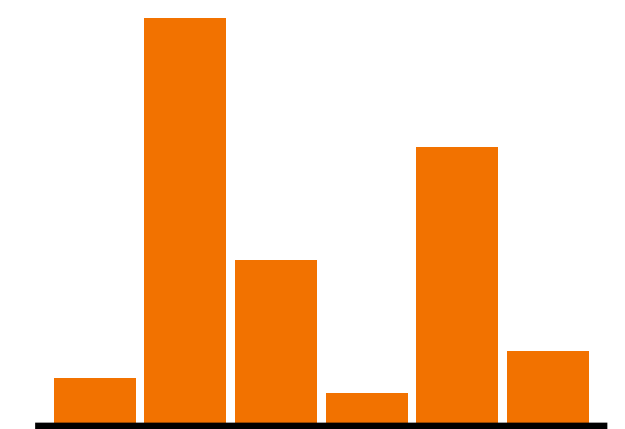
$$p_{\tau}(y | y_{<t}) = \frac{\exp \frac{p(y | y_{<t})}{\tau}}{\sum_{w \in V} \exp \frac{p(w | y_{<t})}{\tau}}$$



$p_2(y_t | y_{<t})$



$p_1(y_t | y_{<t})$

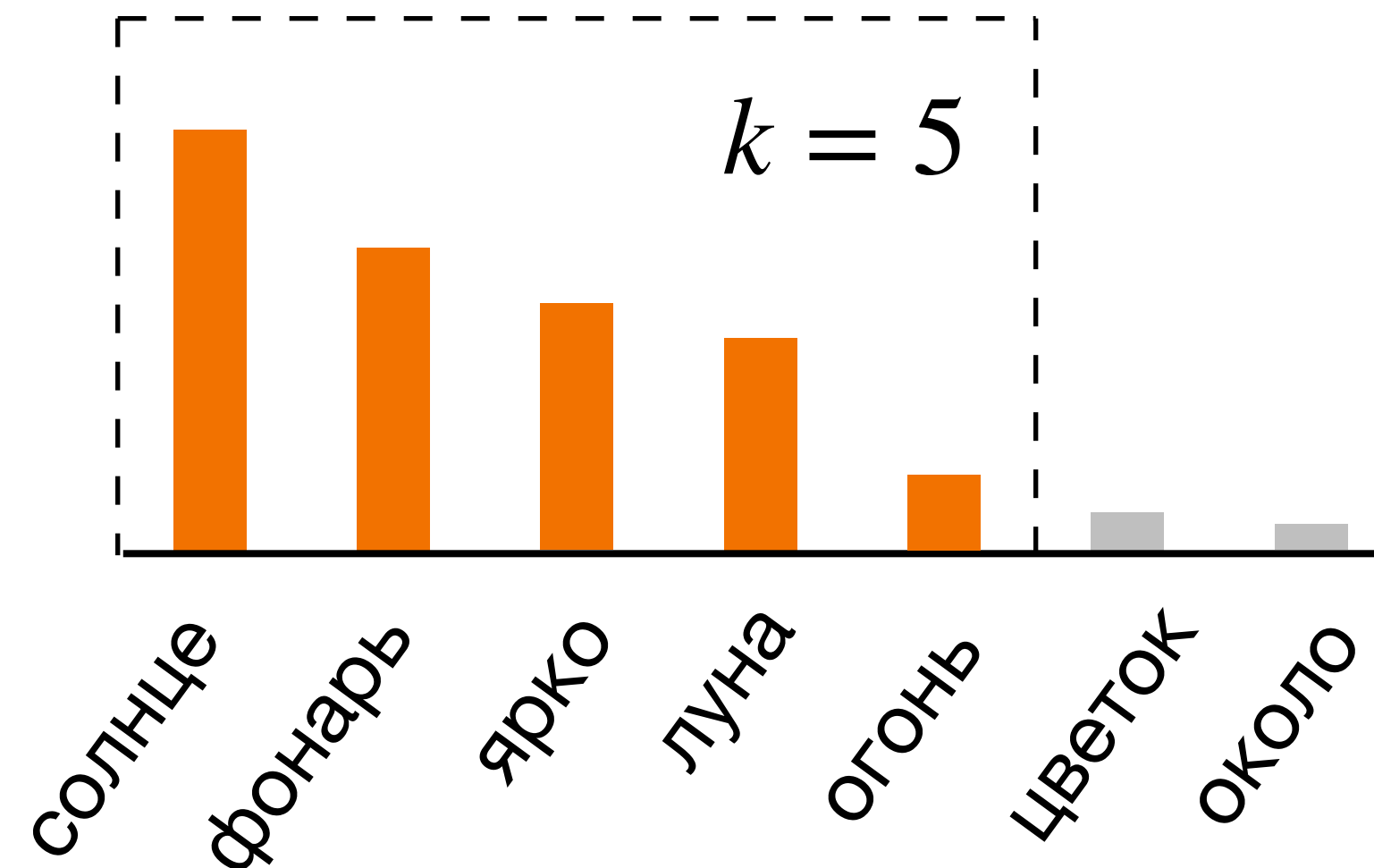


$p_{0.5}(y_t | y_{<t})$

Тор-k семплирование

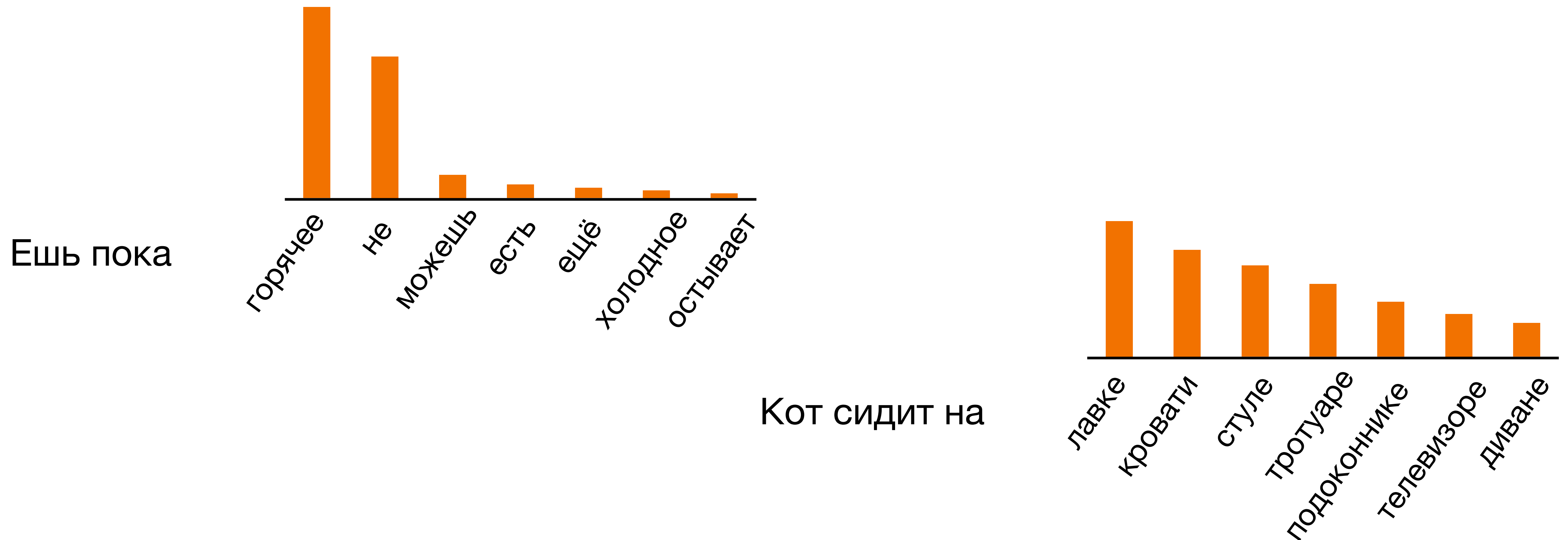
- Даже с температурой остается ненулевая генерация выдать случайный токен
- Оставим только k самых вероятных токенов и будем семплировать из них

На улице светит



Top-k семплирование

- Не во всех ситуациях самых вероятных токенов одинаковое число
- Из-за этого невозможно подобрать идеальное k



Тор-р семплирование

Nucleus sampling

- Будем выбирать из минимального числа токенов, суммарная вероятность которых больше p .

$$\sum_{w \in V^{(p)}} p(w | y_{<t}) \geq p$$

$$|V^{(p)}| \rightarrow \min$$

Популярное значение для p – 0.9 или 0.95