

Рекуррентные нейронные сети, Трансформер

План

- Рекуррентные нейронные сети (RNN)
- Long short-term memory (LSTM)
- Модификации рекуррентных сетей

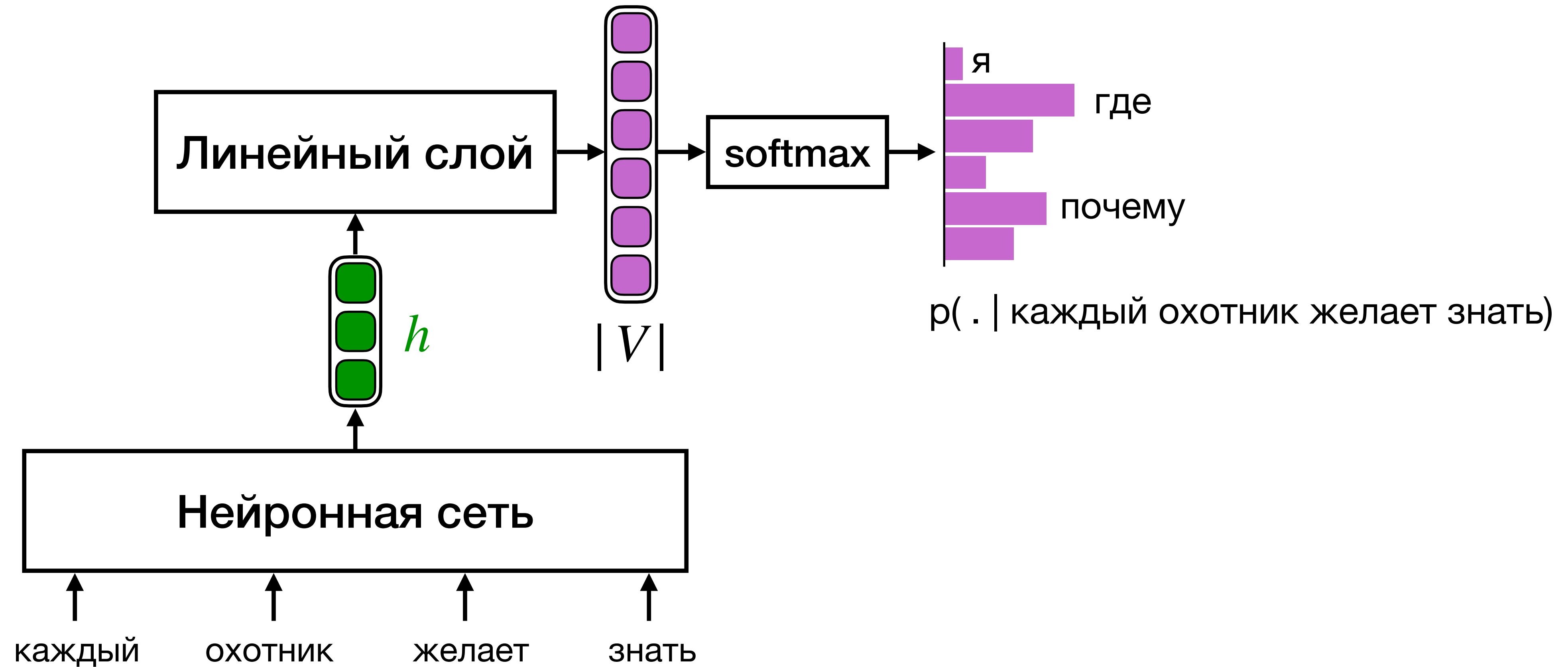
N-грамм модель генерации

Следующий токен зависит только от **n** предыдущих.

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

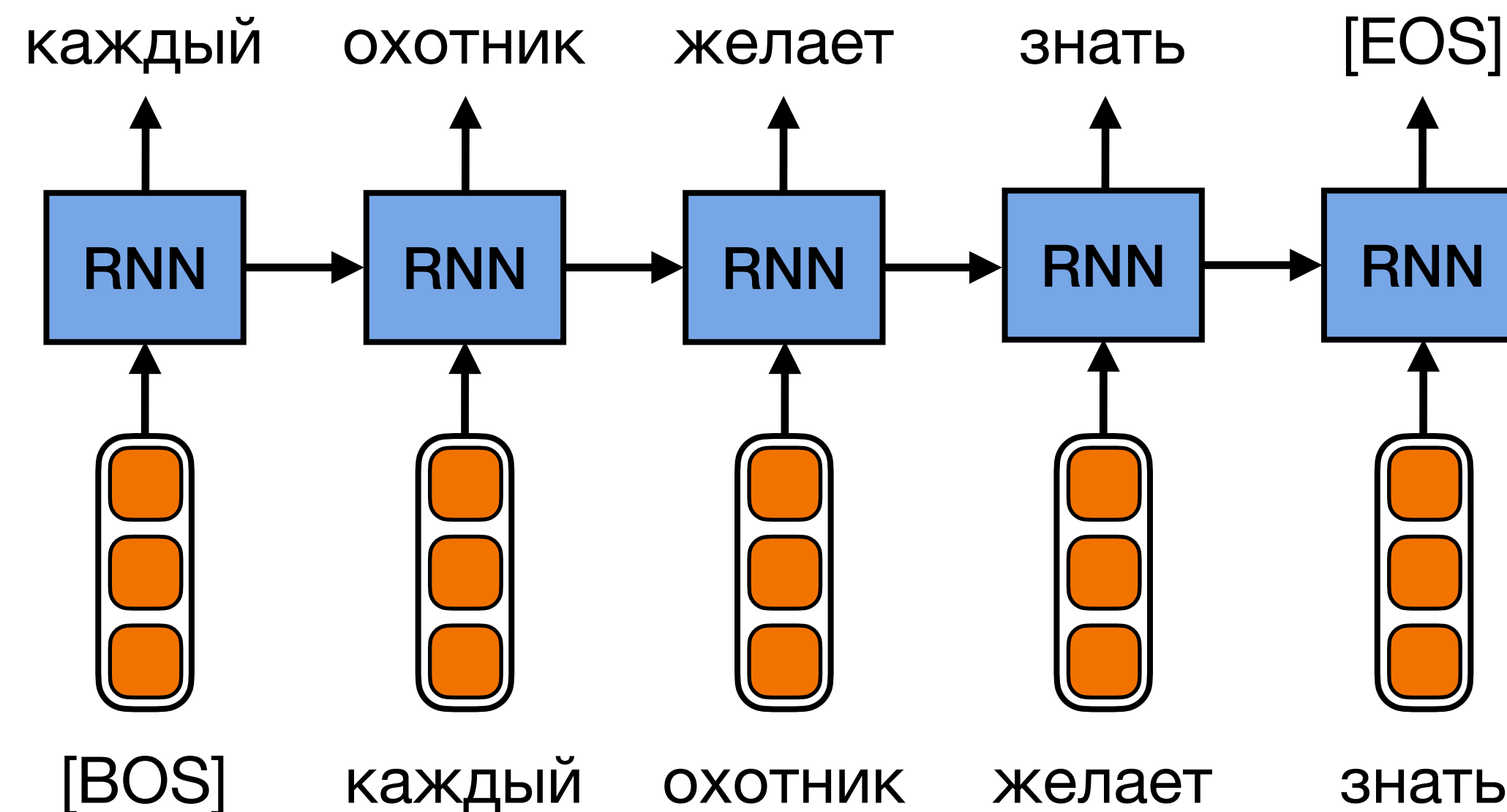
Не можем учитывать даже близкие зависимости
Плохое качество генерации

Нейронные сети для генерации текста



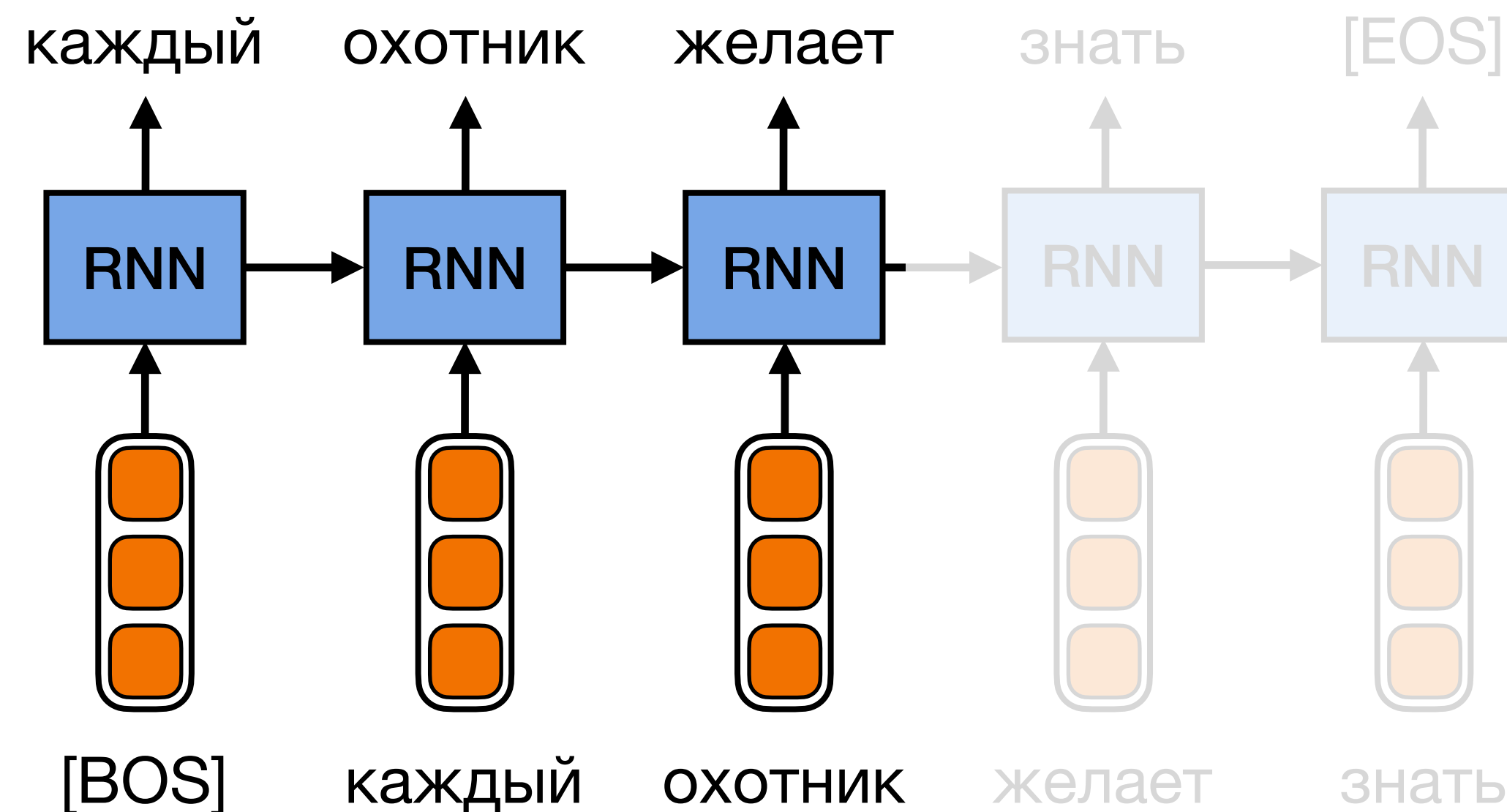
Recurrent Neural Networks (RNN)

- Разработаны для работы с последовательными данными.
- Каждый блок предсказывает следующий токен.



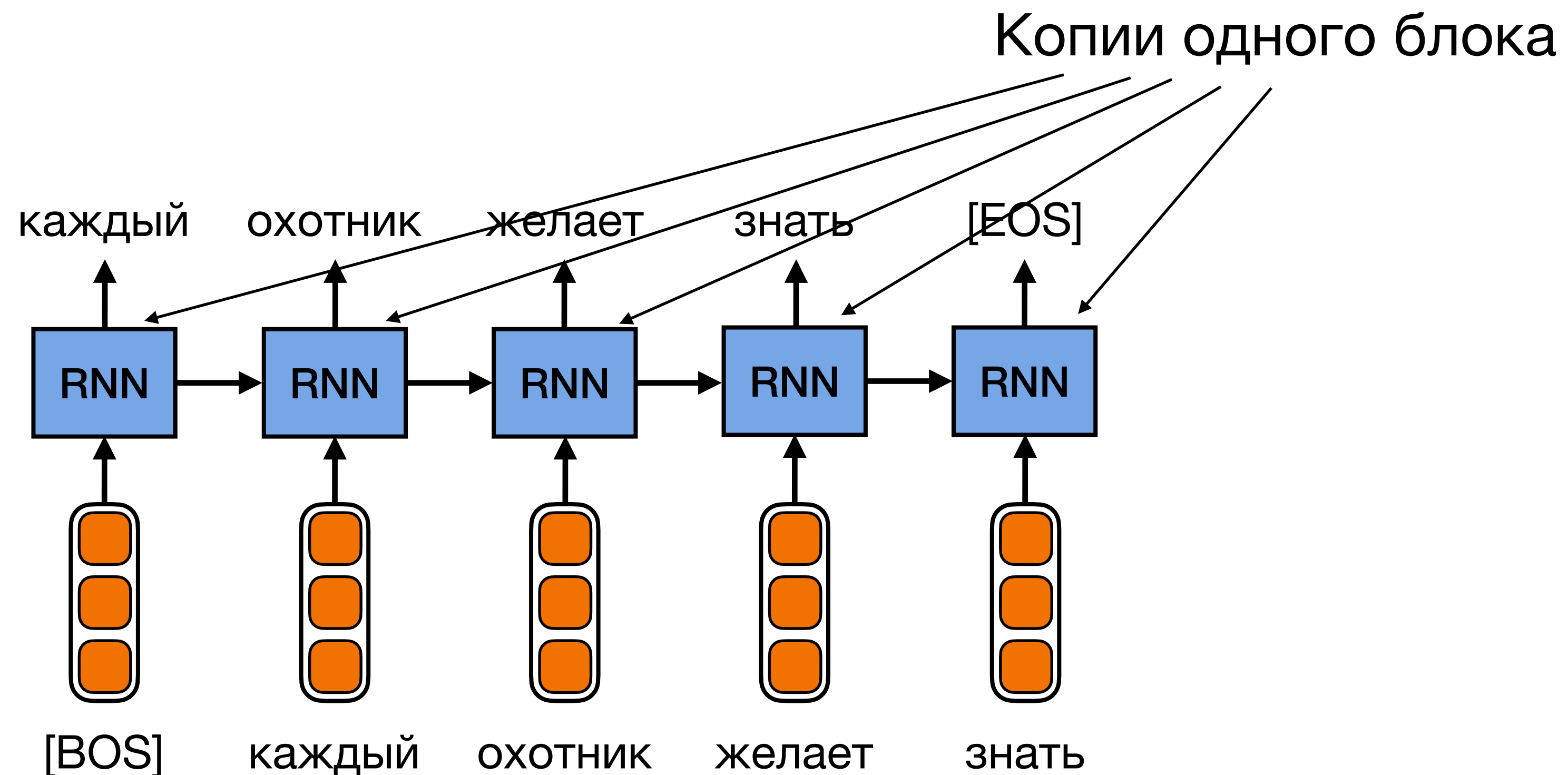
Recurrent Neural Networks (RNN)

- Разработаны для работы с последовательными данными.
- Каждый блок предсказывает следующий токен.
- Процесс генерации интуитивно понятен.

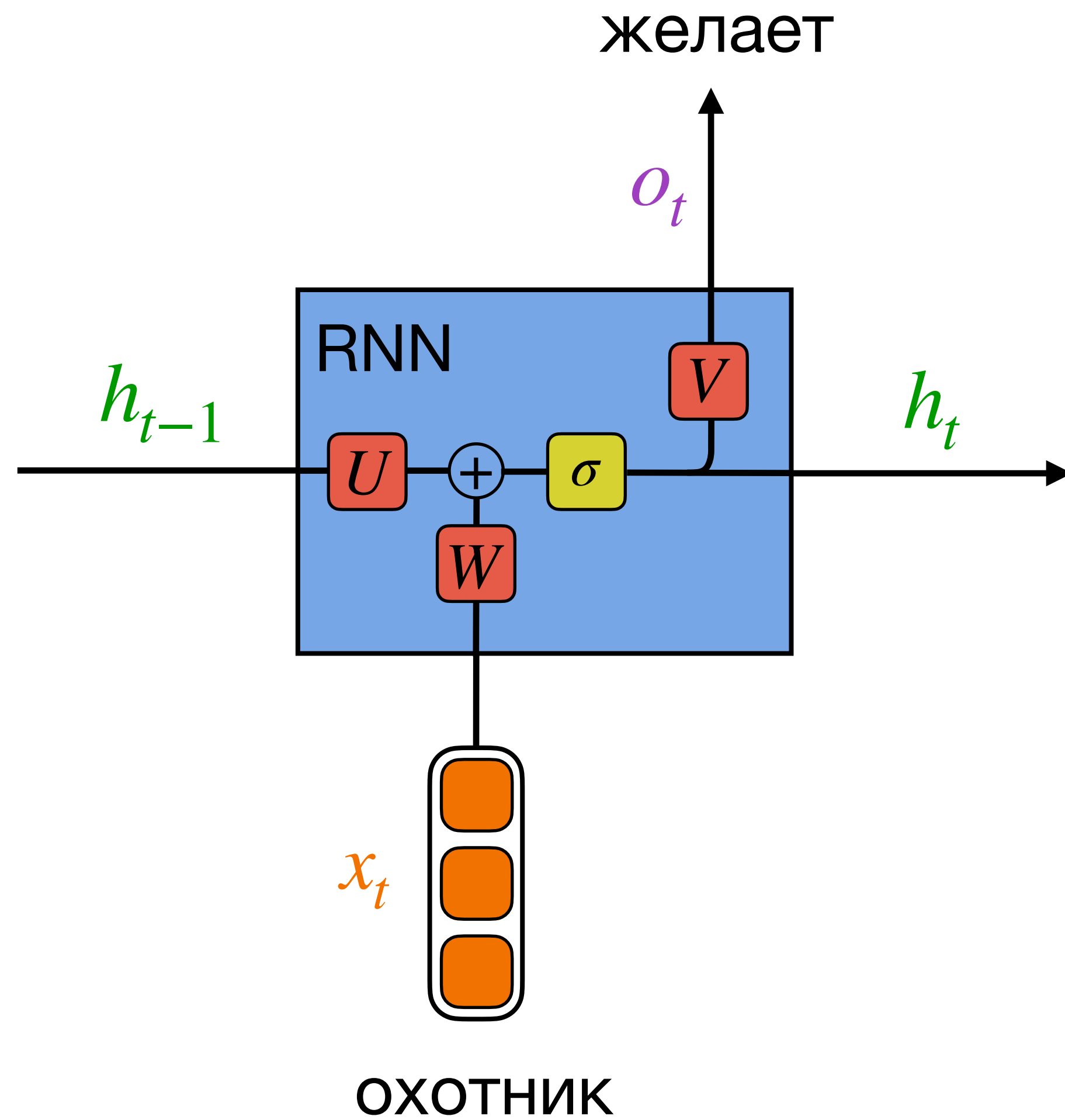


Recurrent Neural Networks (RNN)

- Разработаны для работы с последовательными данными.
- Каждый блок предсказывает следующий токен.



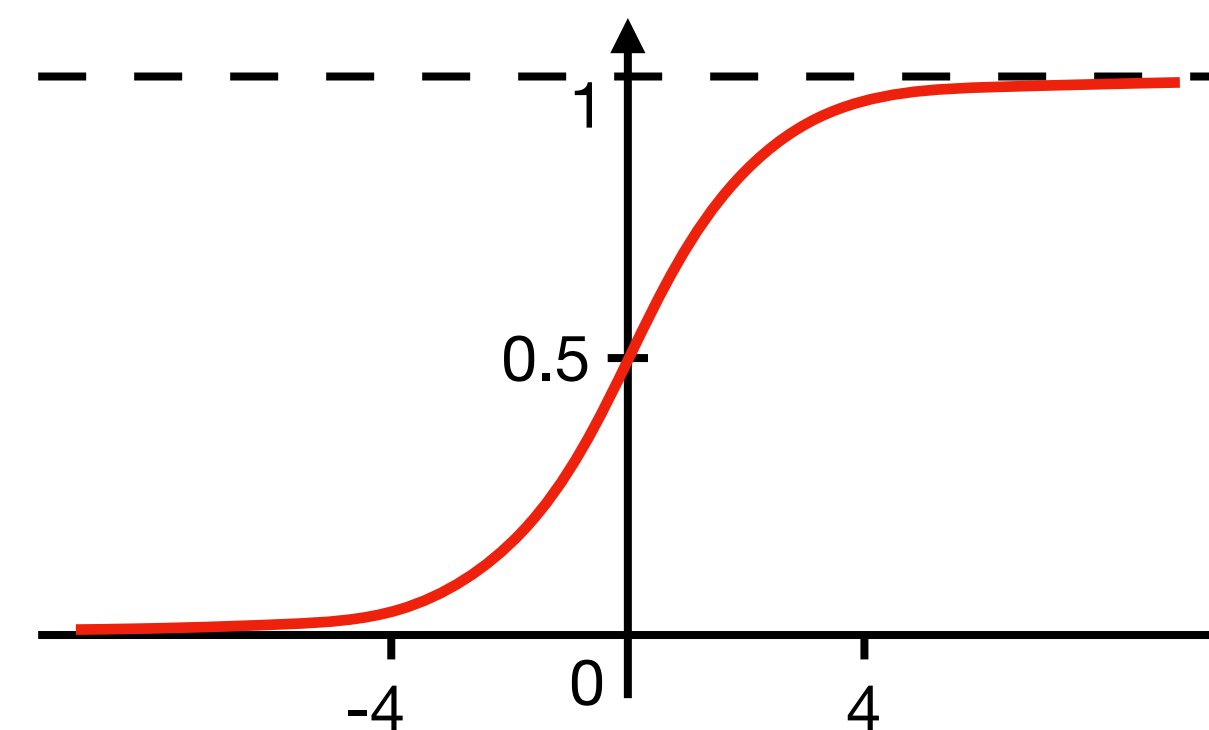
Блок RNN



$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

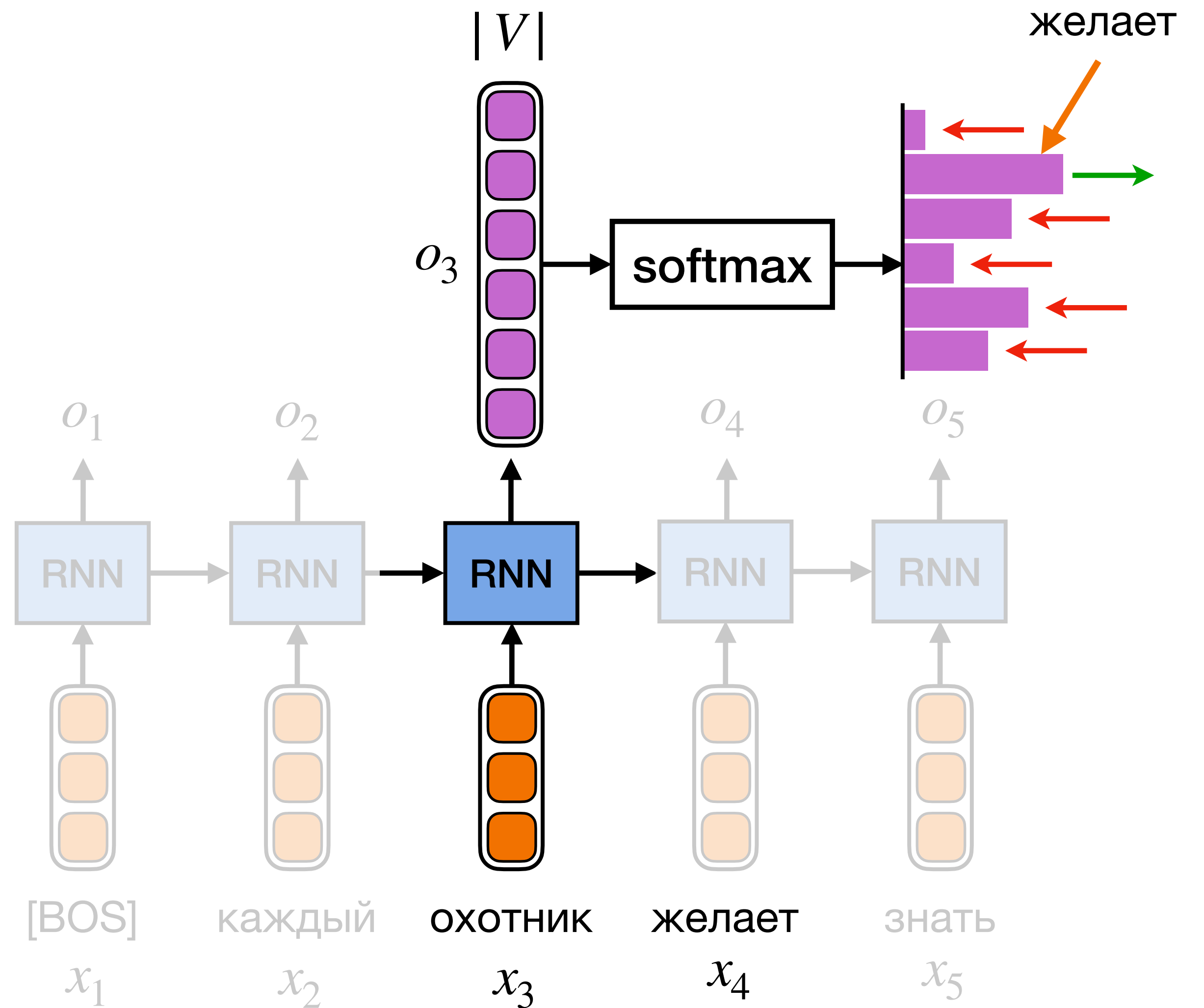
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



RNN: Обучение

$$p(x_1, \dots, x_m) = \prod_{t=1}^m p(x_t | x_{<t}) \rightarrow \max$$

$$p(\cdot | x_{<t}) = \text{softmax}(o_t)$$



RNN: Обучение

$$p(x_1, \dots, x_m) = \prod_{t=1}^m p(x_t | x_{<t}) \rightarrow \max$$

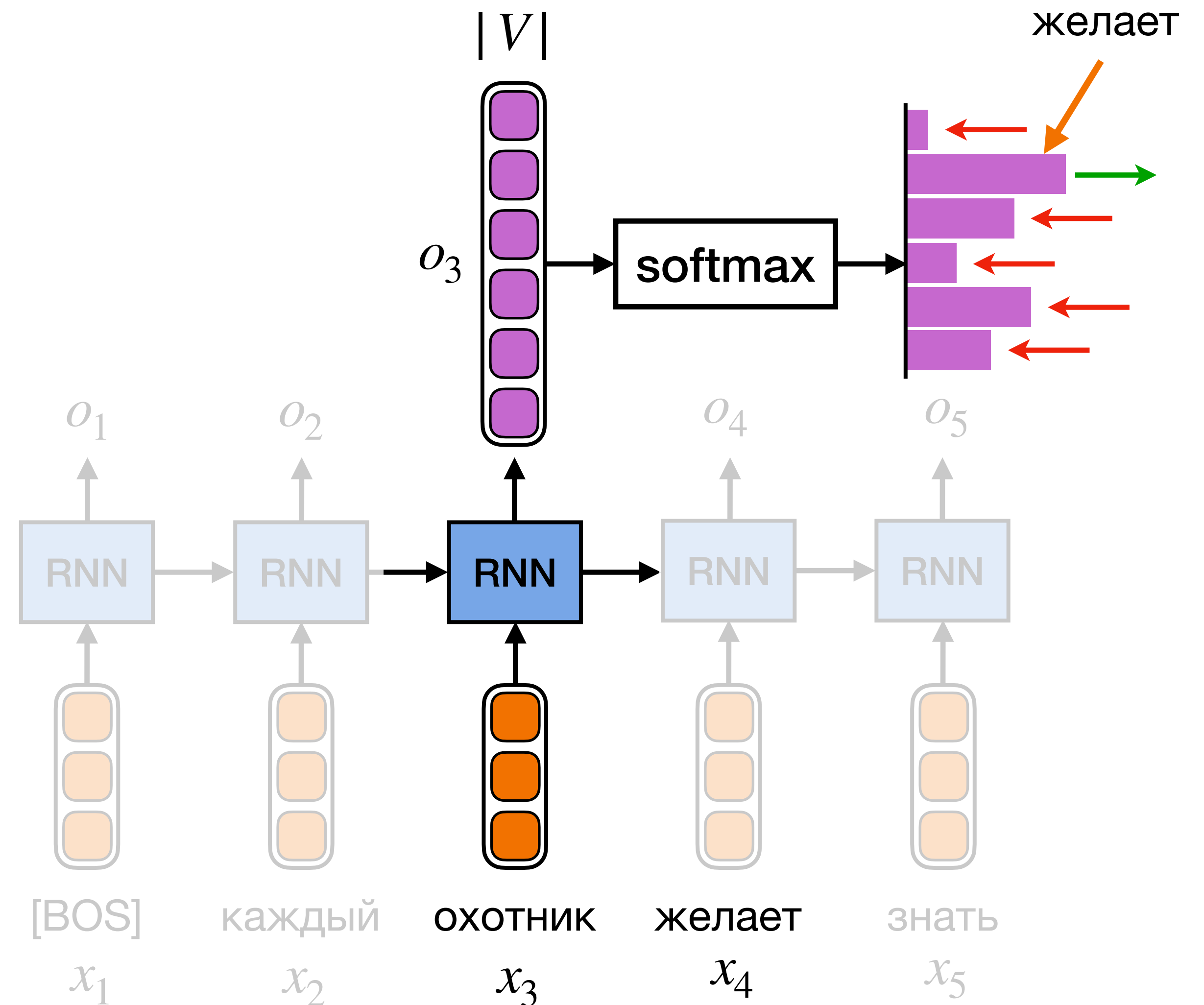
$$p(\cdot | x_{<t}) = \text{softmax}(o_t)$$

Накладывая логарифм и отрицание, получаем кросс-энтропию.

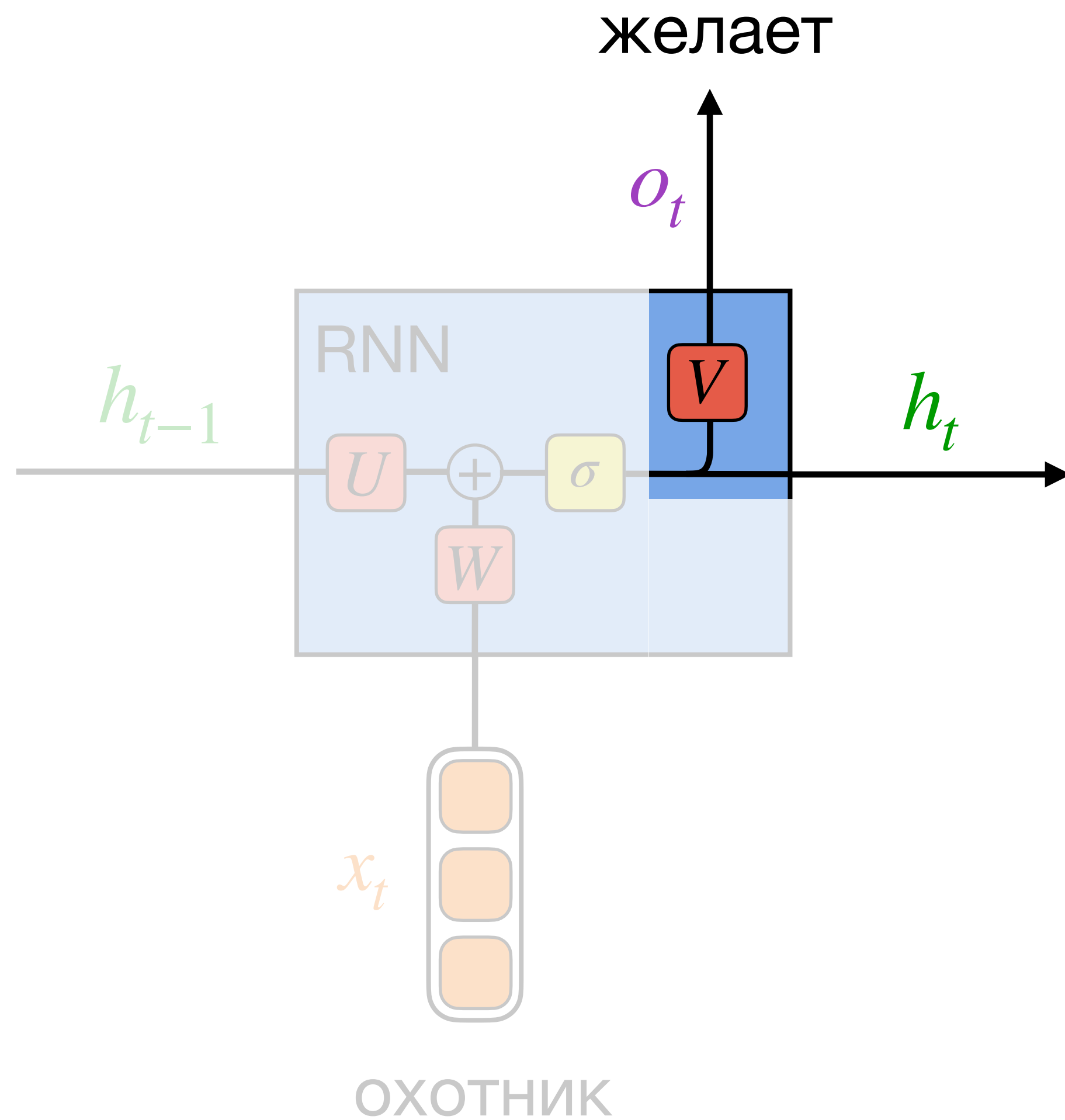
$$L(x) = - \sum_{t=1}^m \log p(x_t | x_{<t}) \rightarrow \min$$

Ошибка для всего корпуса.

$$L(X) = - \frac{1}{|X|} \sum_{x \in X} \sum_{t=1}^m \log p(x_t | x_{<t}) \rightarrow \min$$



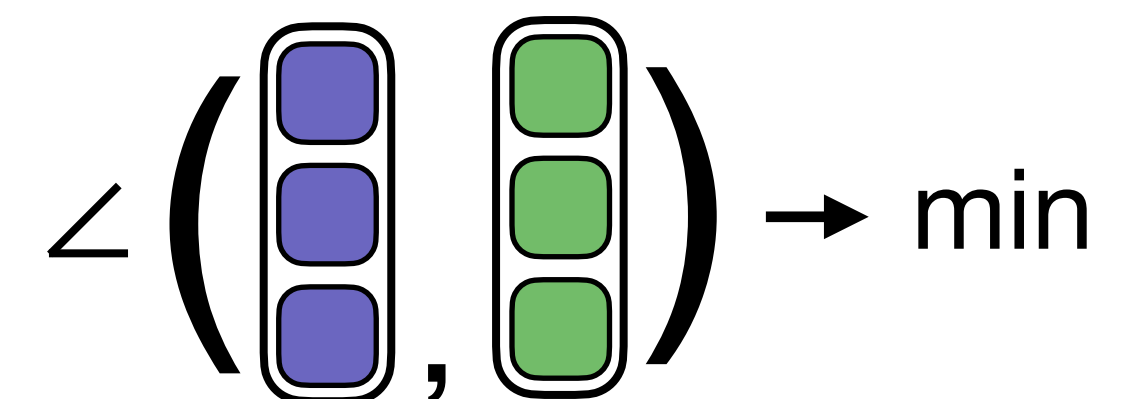
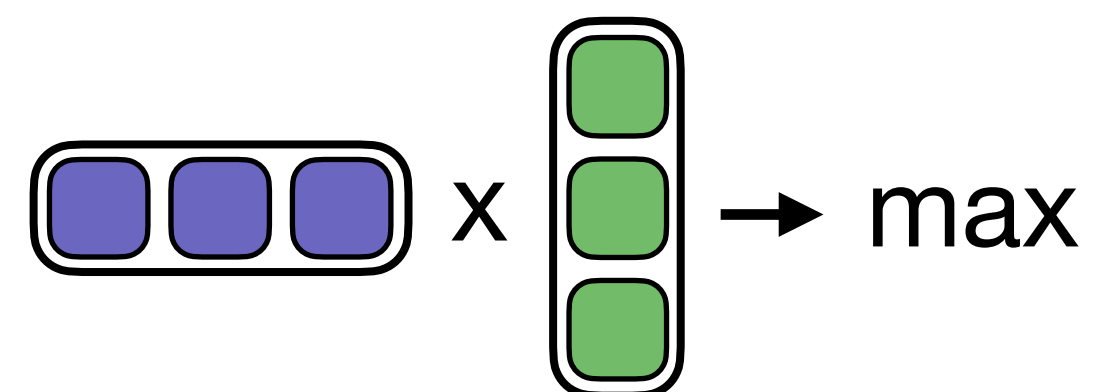
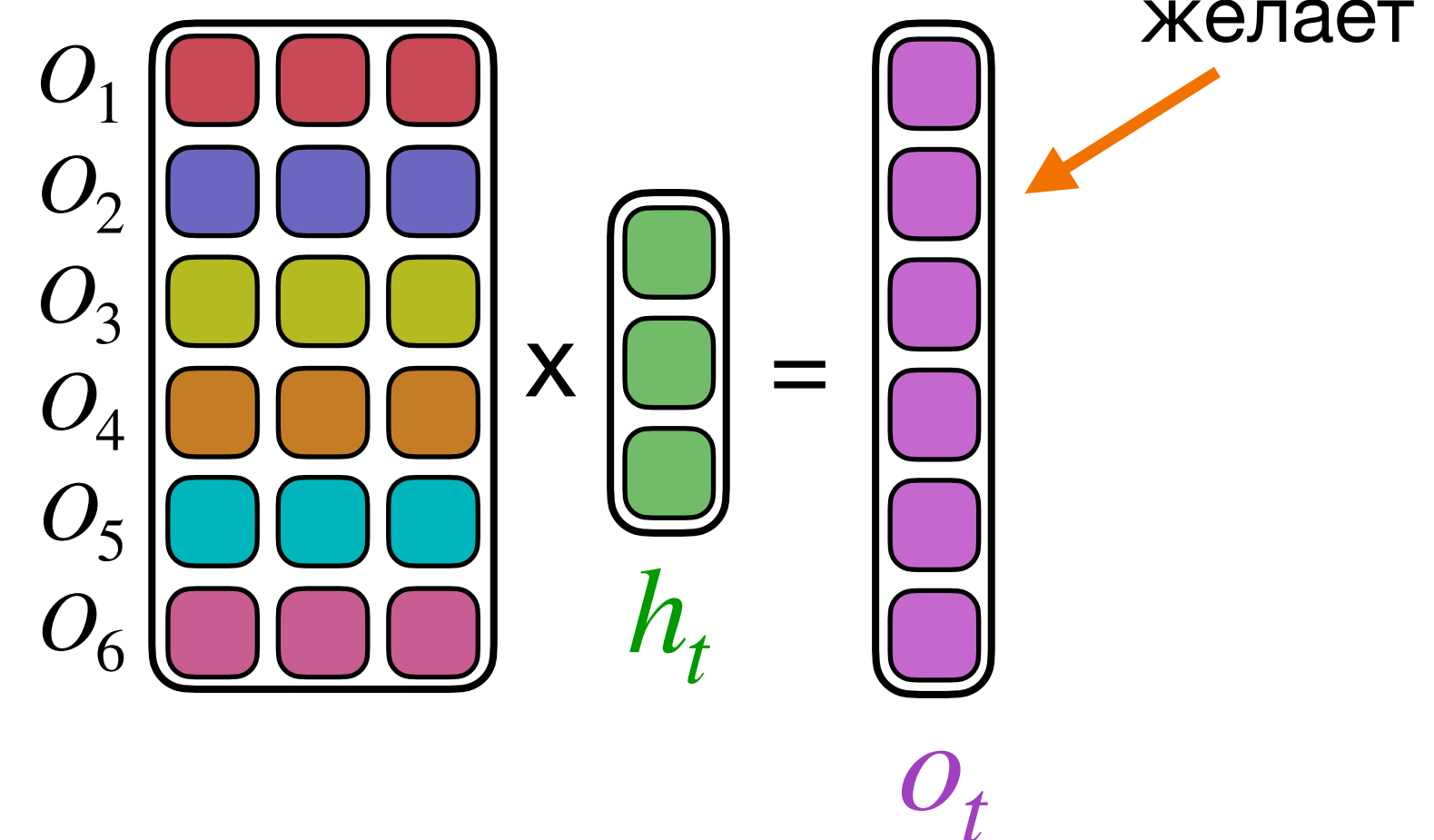
Как еще об этом можно думать?



$$o_t = Vh_t + b_v$$

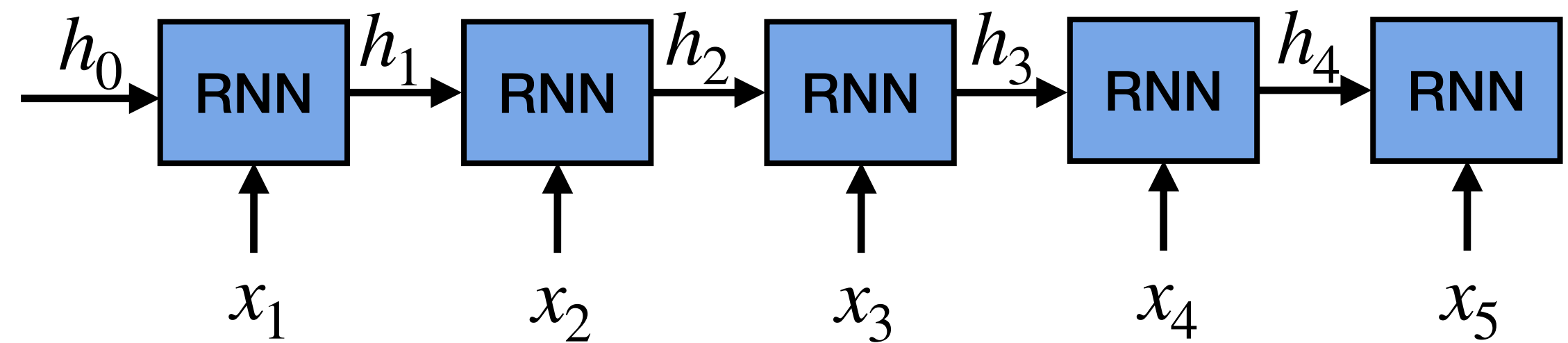
Выходные эмбединги

ТОКЕНОВ



Градиенты RNN

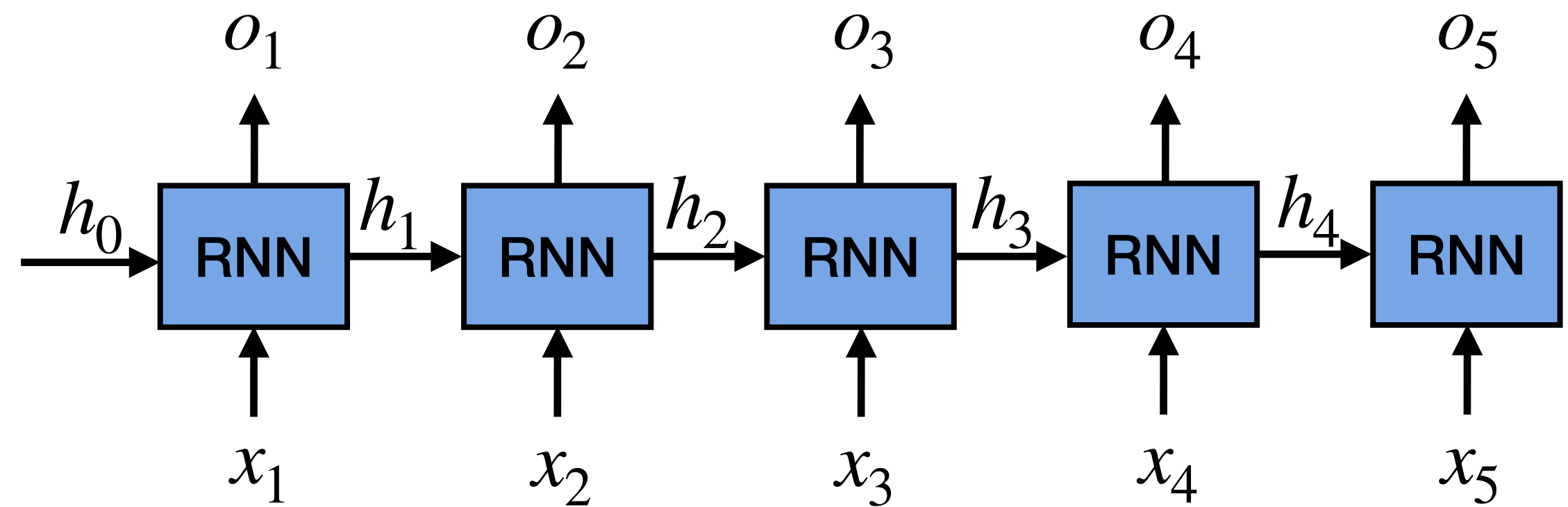
$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$



Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

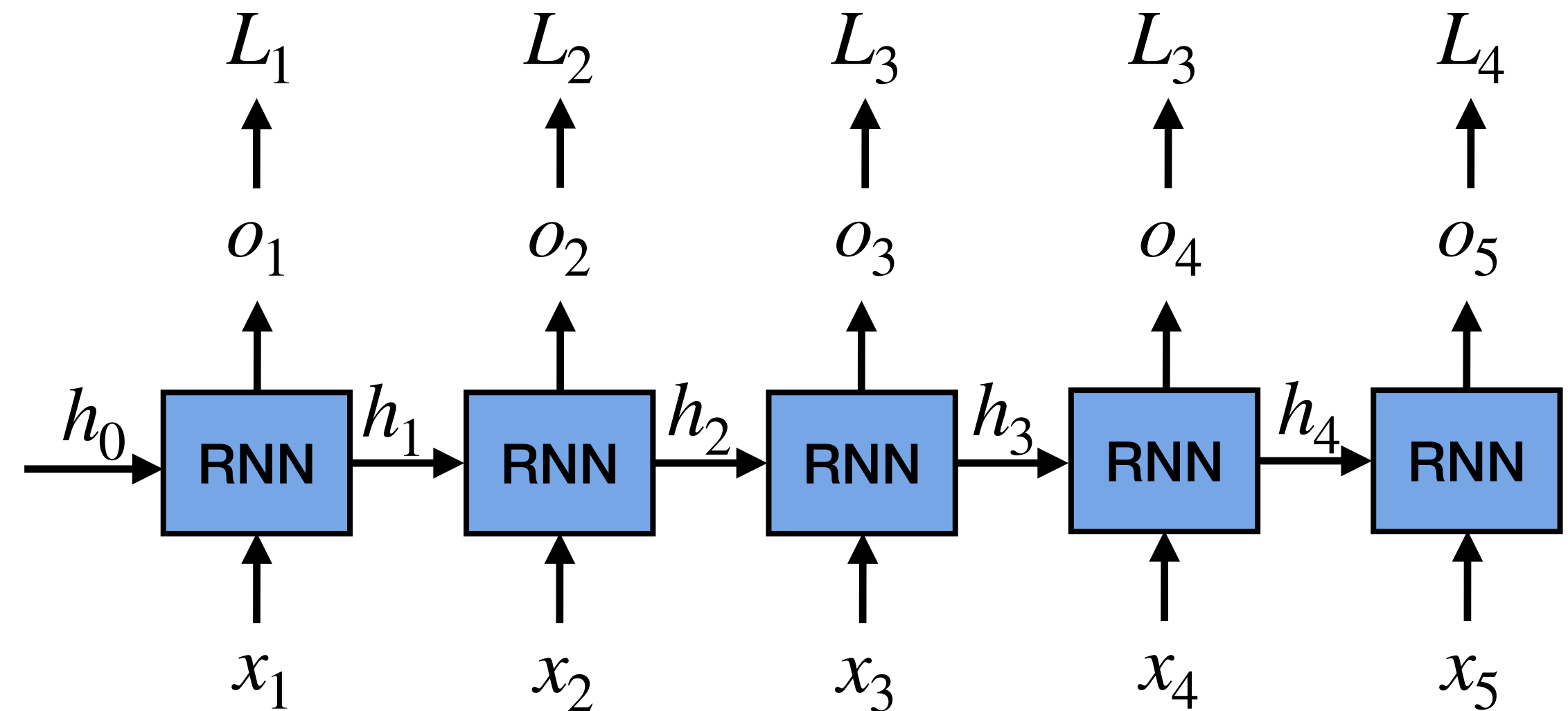


Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$



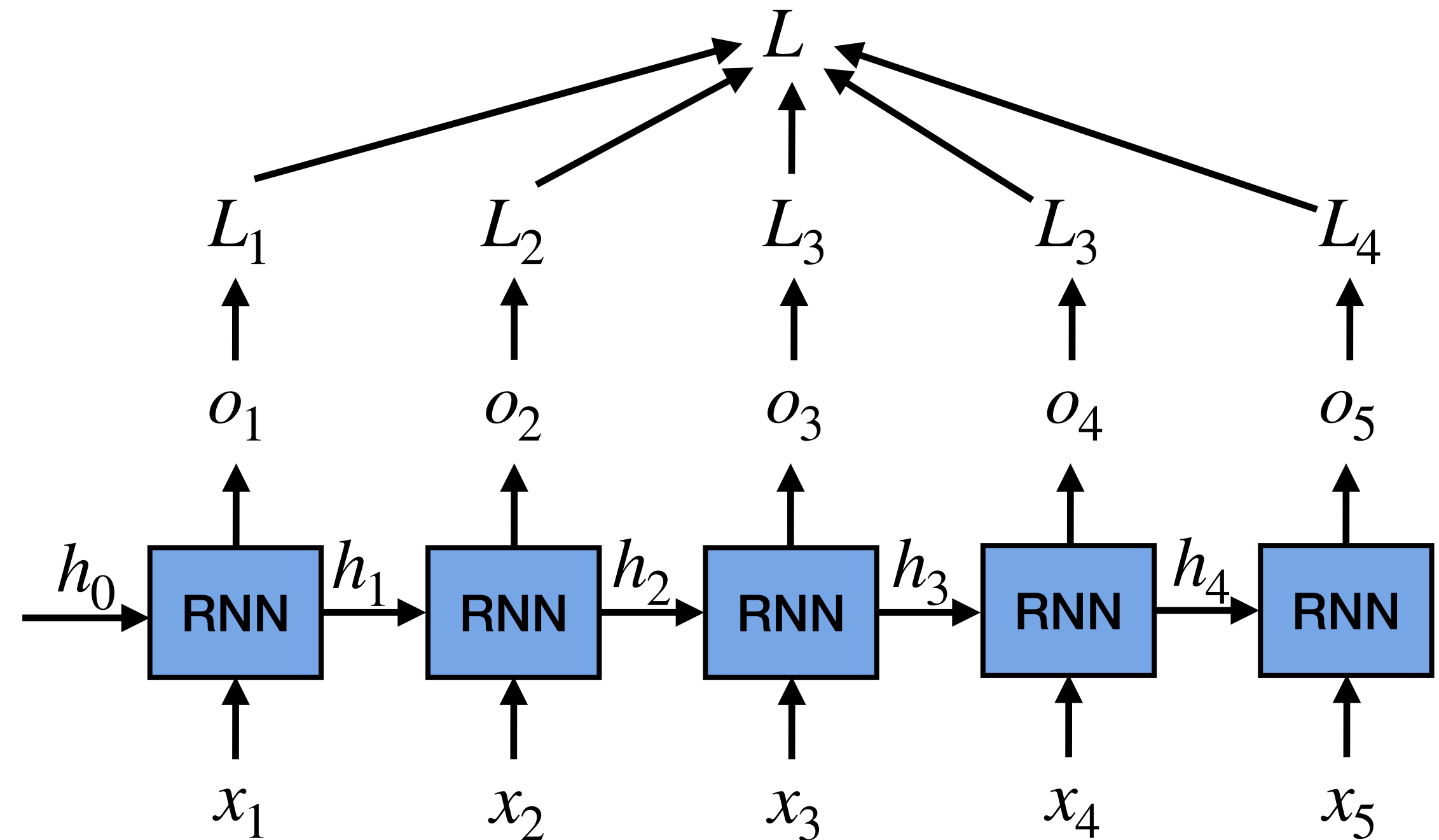
Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$



Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

chain rule

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

$$\frac{dh_t}{dU} = \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{dU}$$

Переходим от производных к частным производным

Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

$$\begin{aligned} \frac{dh_t}{dU} &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \boxed{\frac{dh_{t-1}}{dU}} = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \boxed{\left(\frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} \right)} \end{aligned}$$

Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

$$\begin{aligned} \frac{dh_t}{dU} &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{dU} = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \left(\frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} \right) = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} \end{aligned}$$

Раскрываем скобки

Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

$$\begin{aligned} \frac{dh_t}{dU} &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{dU} = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \left(\frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} \right) = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} = \\ &= \sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \end{aligned}$$

Собираем все в одну сумму

Градиенты RNN

$$h_t = \sigma(Wx_t + Uh_{t-1} + b_h)$$

$$o_t = Vh_t + b_v$$

$$L_t = -\log p(x_t | x_{<t}) = -\log \text{softmax}(o_t)_{x_t}$$

$$L = \sum_{t=1}^m L_t$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \frac{dh_t}{dU}$$

$$\begin{aligned} \frac{dh_t}{dU} &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{dU} = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \left(\frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} \right) = \\ &= \frac{\partial h_t}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial U} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{dh_{t-2}}{dU} = \\ &= \sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \end{aligned}$$

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$

Взрыв градиентов

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$

Серия умножений производных

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| > 1$$

- Происходит **взрыв градиента** $\frac{dL}{dU}$
- Модель расходится, NaN в весах

Взрыв градиентов

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$

Серия умножений производных

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| > 1$$

- Происходит **взрыв градиента** $\frac{dL}{dU}$
- Модель расходится, NaN в весах

Решения:

- Регуляризация
- Уменьшение learning rate
- Gradient clipping

Взрыв градиентов

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$

Серия умножений производных

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| > 1$$

- Происходит **взрыв градиента** $\frac{dL}{dU}$
- Модель расходится, NaN в весах

Решения:

- Регуляризация
- Уменьшение learning rate
- Gradient clipping

$$1. g \leftarrow \min \left(1, \frac{\text{max norm}}{\|g\|} \right) \cdot g$$

Правильный способ

$$2. g \leftarrow \text{clip}(g, -C, C)$$

Ленивый способ (меняет направление градиента)

Затухание градиентов

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| < 1$$

- Происходит **затухание градиента**
- Модель перестает учиться
- Модель не улавливает далекие зависимости!

Затухание градиентов

$$\frac{dL}{dU} = \sum_{t=1}^m \frac{dL_t}{do_t} \frac{do_t}{dh_t} \left[\sum_{k=1}^t \left(\prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right) \frac{\partial h_k}{\partial U} \right]$$
$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| < 1$$

- Происходит **затухание градиента**
- Модель перестает учиться
- Модель не улавливает далекие зависимости!

Затухание градиентов – частая проблема RNN.

Ее нельзя починить трюками.

Затухание градиентов: почему возникает?

$$h_j = \sigma(\underbrace{Wx_j + Uh_{j-1} + b_h}_{z_j})$$

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial z_j} \frac{\partial z_j}{\partial h_{j-1}}$$

Затухание градиентов: почему возникает?

$$h_j = \sigma(\underbrace{Wx_j + Uh_{j-1} + b_h}_{z_j})$$

Поэлементное умножение

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial z_j} \frac{\partial z_j}{\partial h_{j-1}} = \left(\sigma(z_j) \odot (1 - \sigma(z_j)) \right) U$$

Затухание градиентов: почему возникает?

$$h_j = \sigma(\underbrace{Wx_j + Uh_{j-1} + b_h}_{z_j})$$

Поэлементное умножение

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial z_j} \frac{\partial z_j}{\partial h_{j-1}} = \left(\sigma(z_j) \odot (1 - \sigma(z_j)) \right) U$$

Посмотрим на спектральную норму

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \underbrace{\| \sigma(z_j) \odot (1 - \sigma(z_j)) \|}_{< 1} \cdot \|U\|$$

т. к. $\sigma(z) \in [0, 1]$

Затухание градиентов: почему возникает?

$$h_j = \sigma(\underbrace{Wx_j + Uh_{j-1} + b_h}_{z_j})$$

Поэлементное умножение

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial h_{j-1}} = \frac{\partial \sigma(z_j)}{\partial z_j} \frac{\partial z_j}{\partial h_{j-1}} = \left(\sigma(z_j) \odot (1 - \sigma(z_j)) \right) U$$

Посмотрим на спектральную норму

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \underbrace{\| \sigma(z_j) \odot (1 - \sigma(z_j)) \|}_{< 1} \cdot \|U\|$$

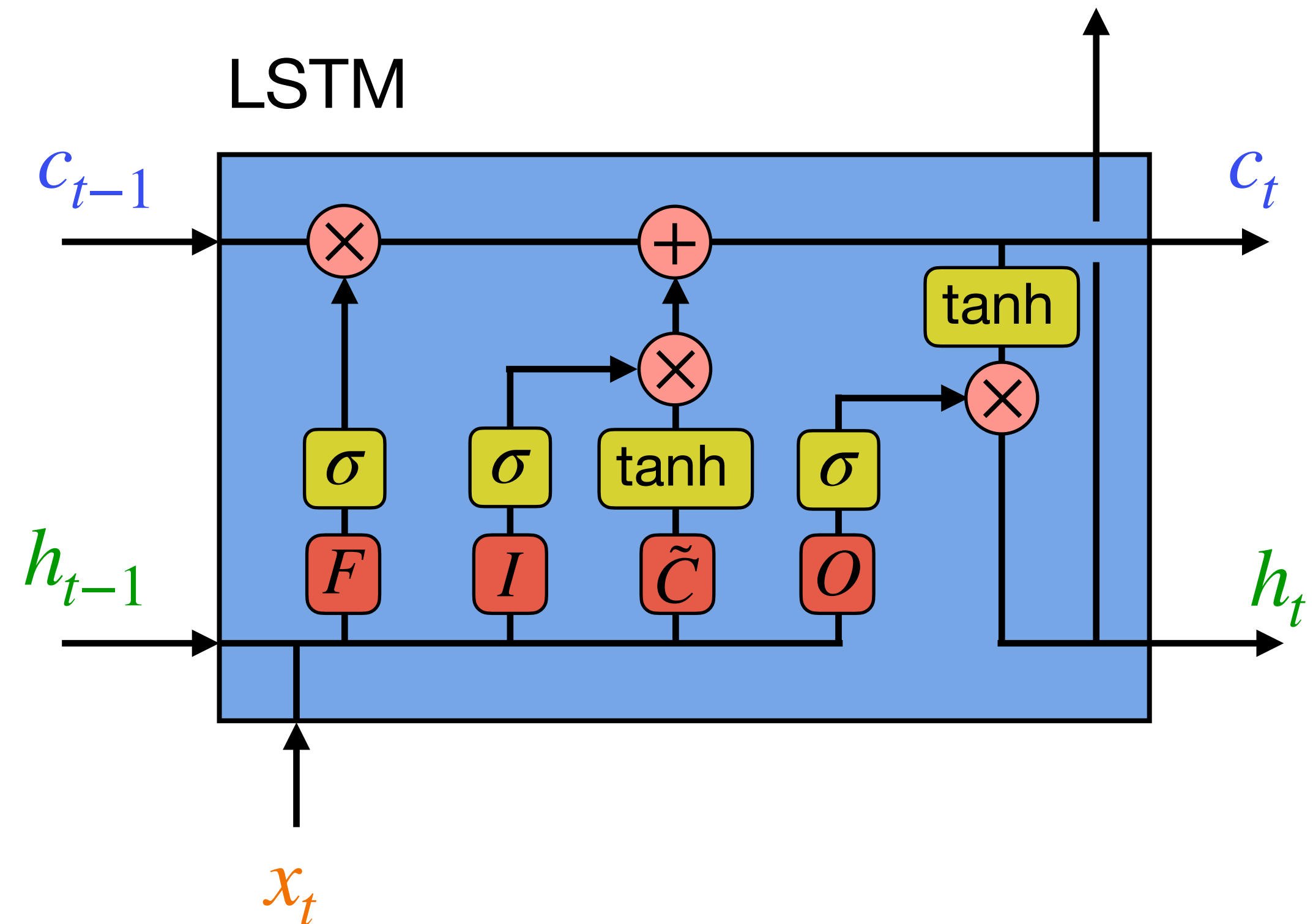
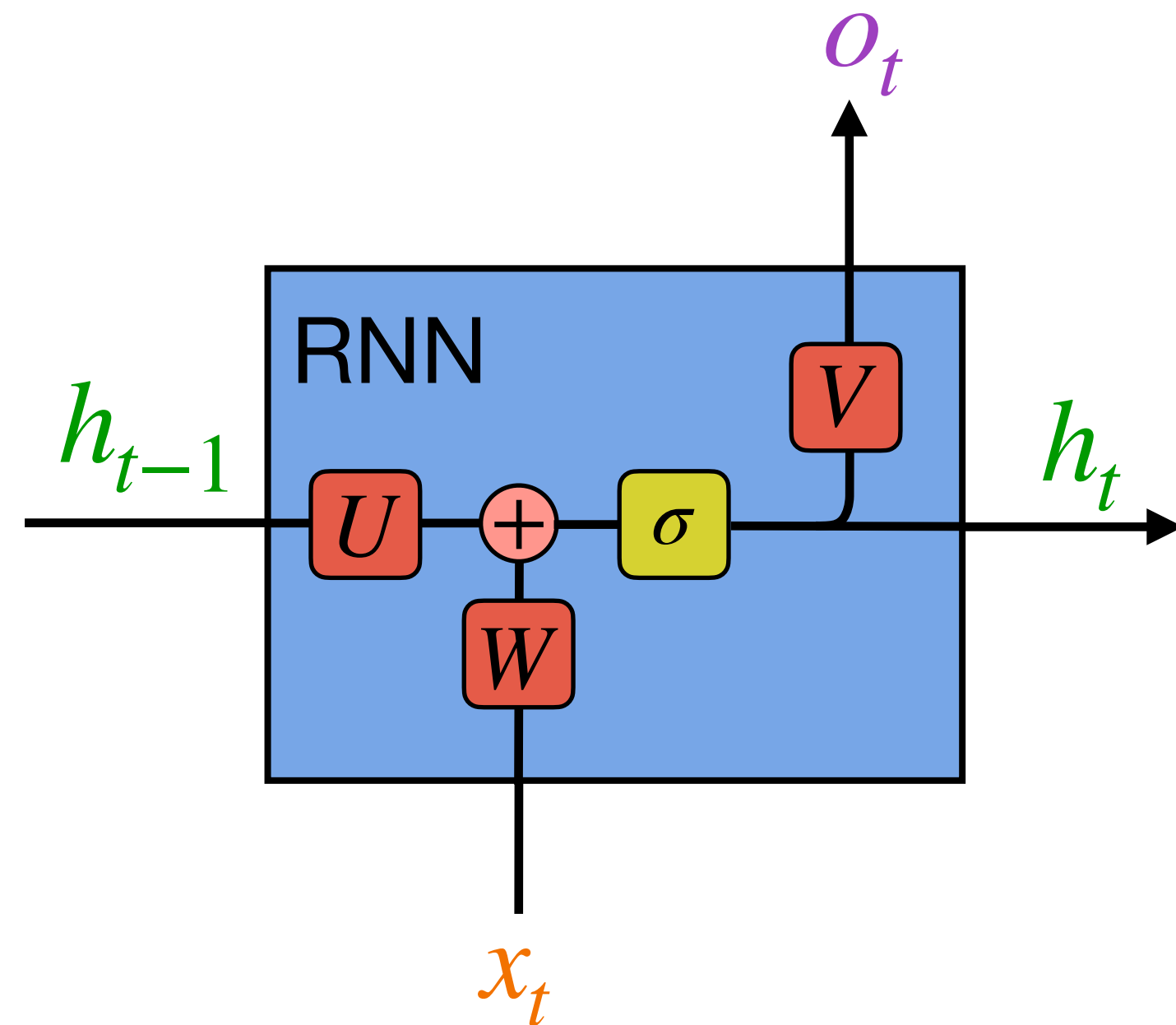
т. к. $\sigma(z) \in [0, 1]$

Если U – ортогональная ($UU^T = I$), то

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \| \sigma(z_j) \odot (1 - \sigma(z_j)) \| < 1$$

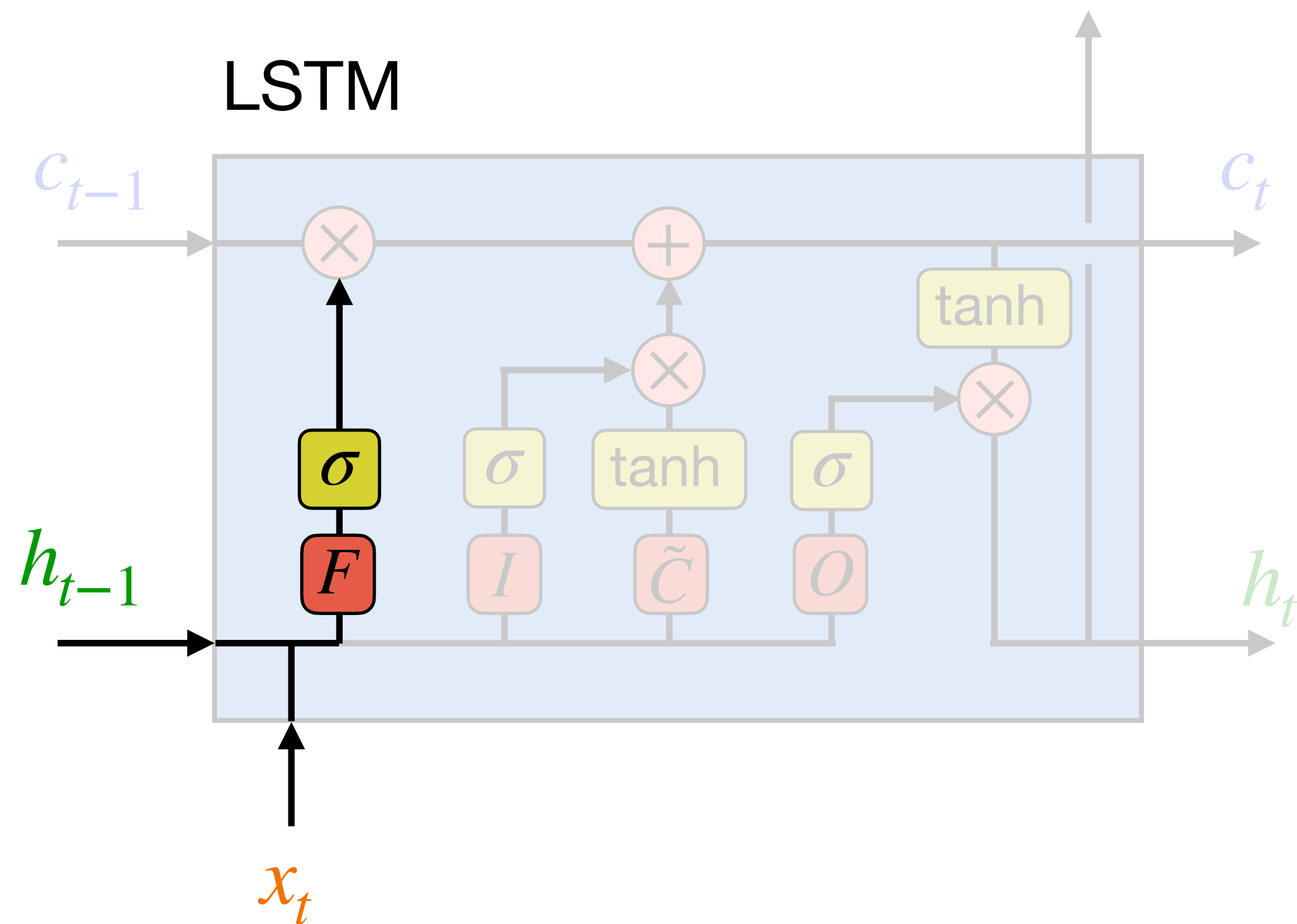
Long shot-term memory (LSTM)

- В LSTM добавляется вектор памяти c_t
- Благодаря ему модель не забывает старую информацию
- Так же добавляется 3 фильтра



LSTM: фильтр забывания

Контролирует, какую информацию надо забыть, а какую оставить.



$$f_t = \sigma(W_f x_{t-1} + U_f h_{t-1} + b_f)$$

$$f_t \in [0,1]$$

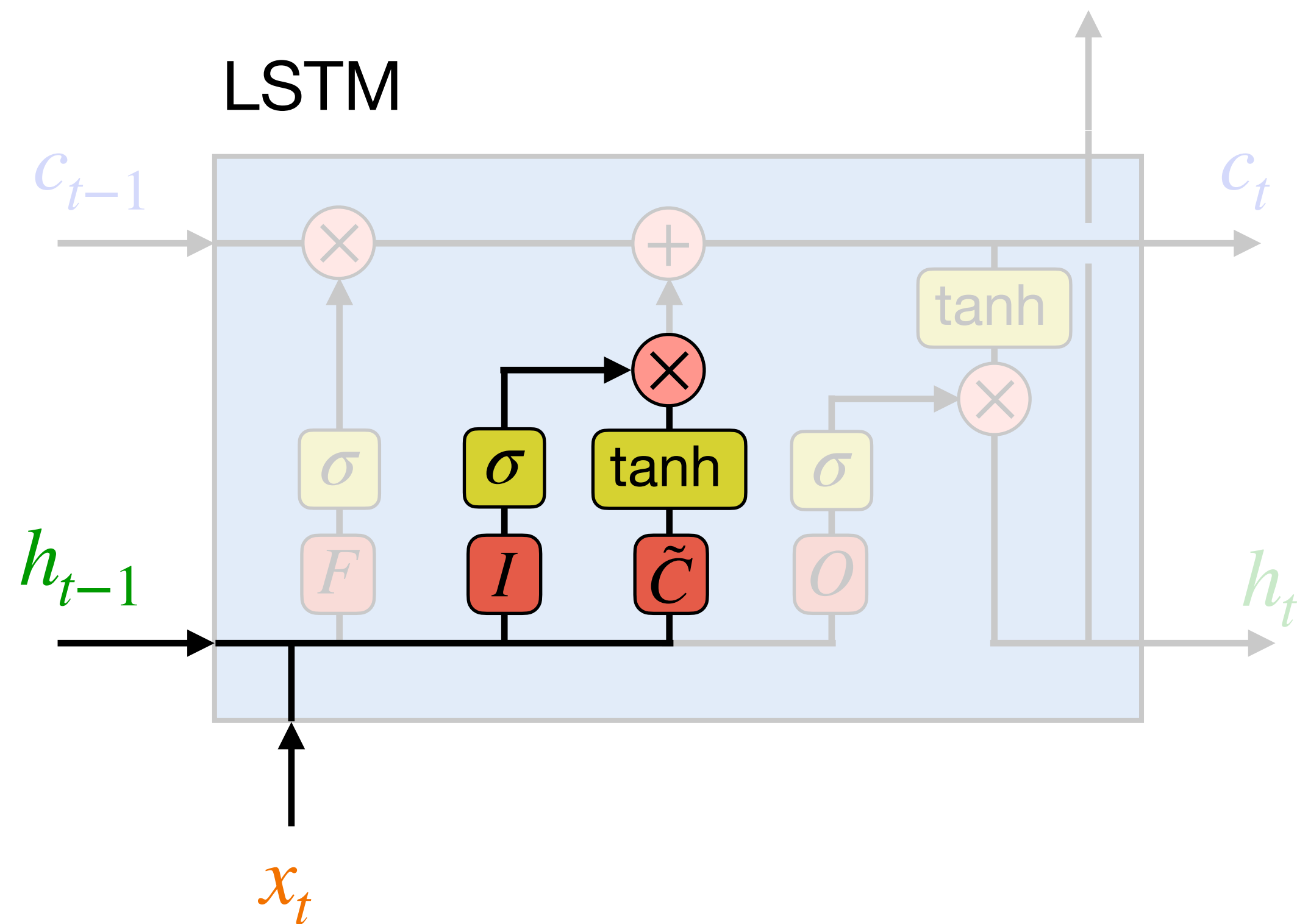
Еда была **вкусная** и мы **не разочаровались**.

x_3 – слово-маркер, можно забыть все до него.

x_7 – негативный маркер, но до него идет "**не**".
Знаем об этом из h_7 .

LSTM: фильтр входа

Контролирует, какую информацию надо добавить в вектор памяти c_t

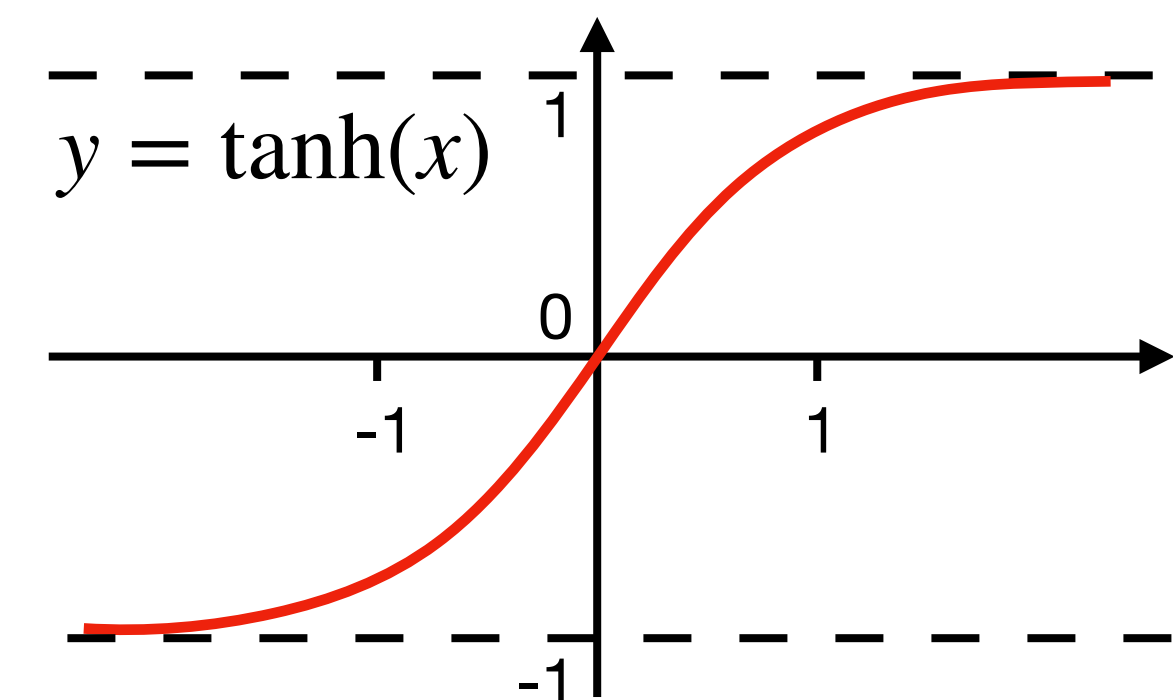


Из-за i_t можем не добавлять ничего

$$i_t = \sigma(W_i x_{t-1} + U_i h_{t-1} + b_i)$$

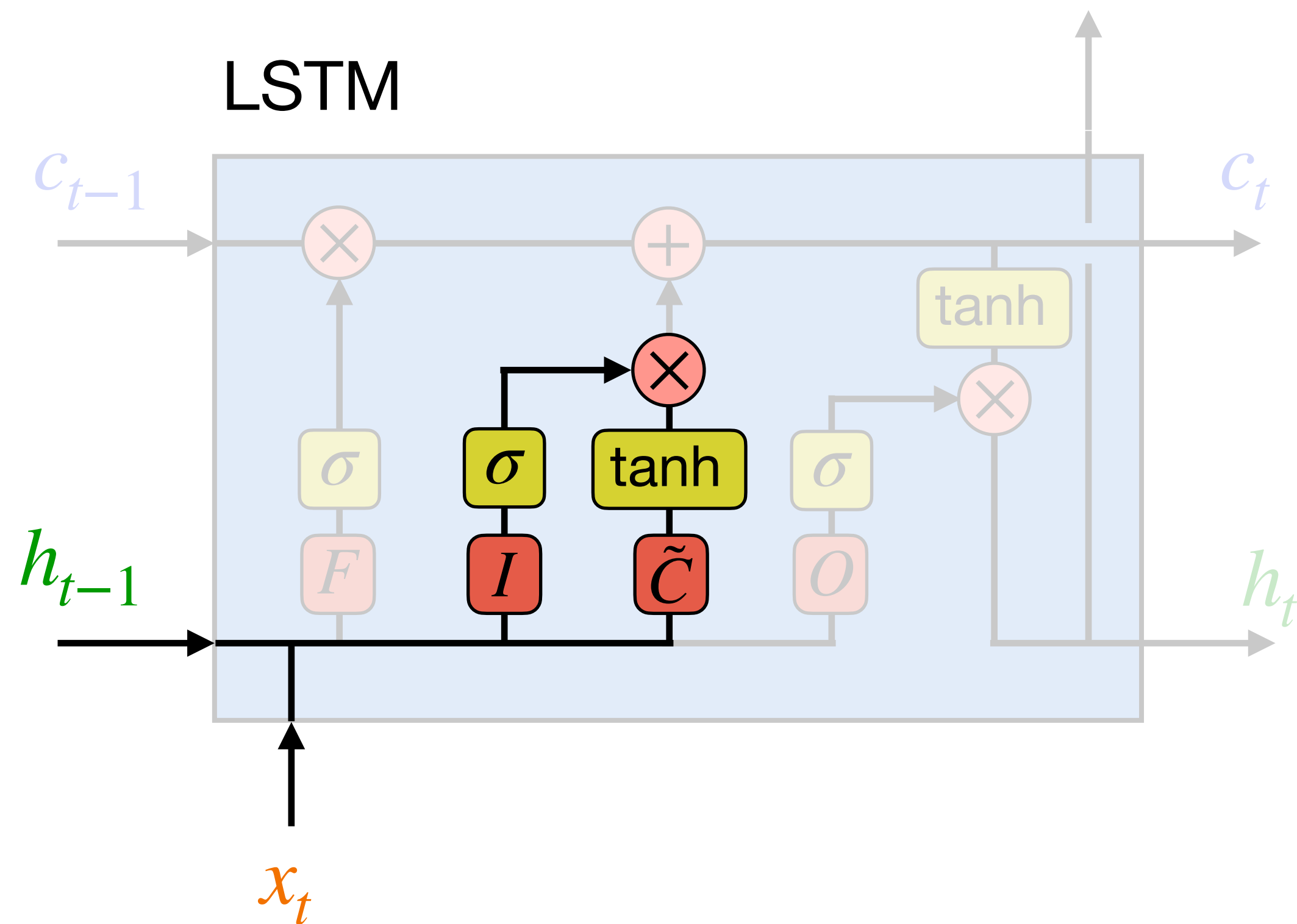
$$i_t \in [0, 1]$$

$$\tilde{c}_t = \tanh(W_c x_{t-1} + U_c h_{t-1} + b_c)$$



LSTM: фильтр входа

Контролирует, какую информацию надо добавить в вектор памяти c_t



Из-за i_t можем не добавлять ничего.

$$i_t = \sigma(W_i x_{t-1} + U_i h_{t-1} + b_i)$$

$$i_t \in [0, 1]$$

$$\tilde{c}_t = \tanh(W_c x_{t-1} + U_c h_{t-1} + b_c)$$

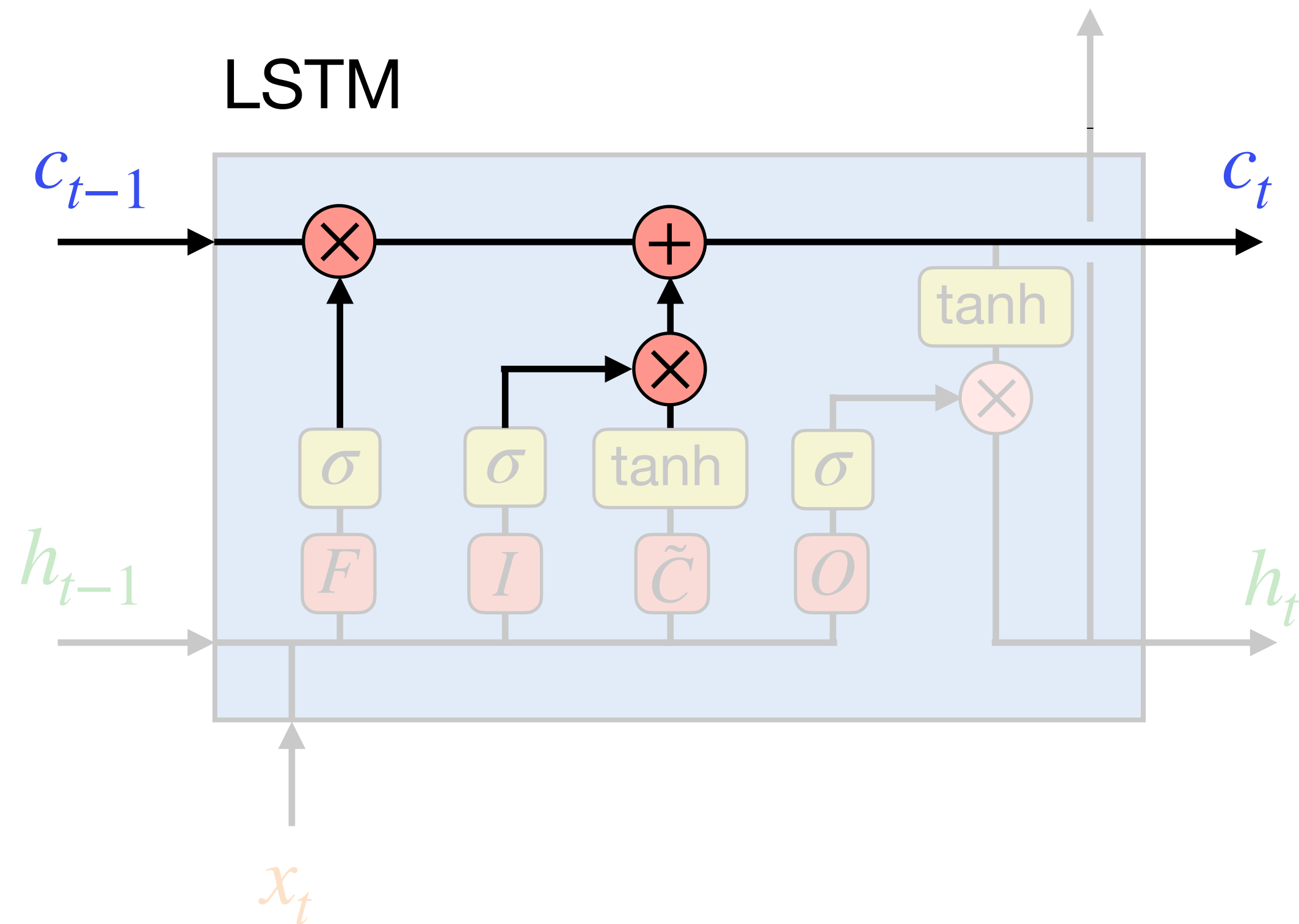
Еда была **вкусная** и **мы** не разочаровались.

x_3 – слово-маркер.
Запоминаем, что класс
положительный.

x_5 – не влияет на класс.
Ничего не добавляем.

LSTM: обновление памяти

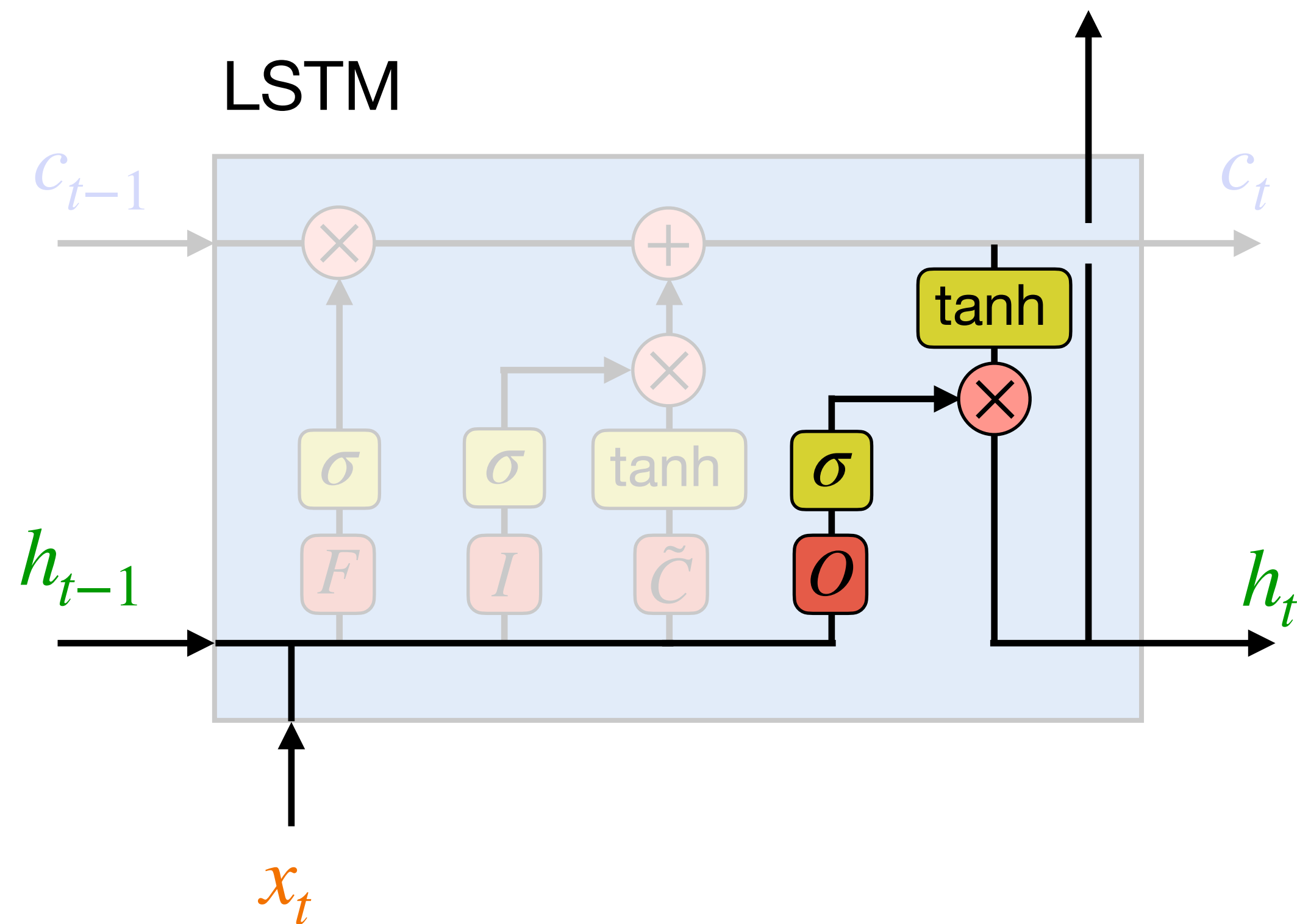
Удаляем ненужную информацию и добавляем новую.



$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

LSTM: фильтр выхода

Контролирует выход текущего шага.

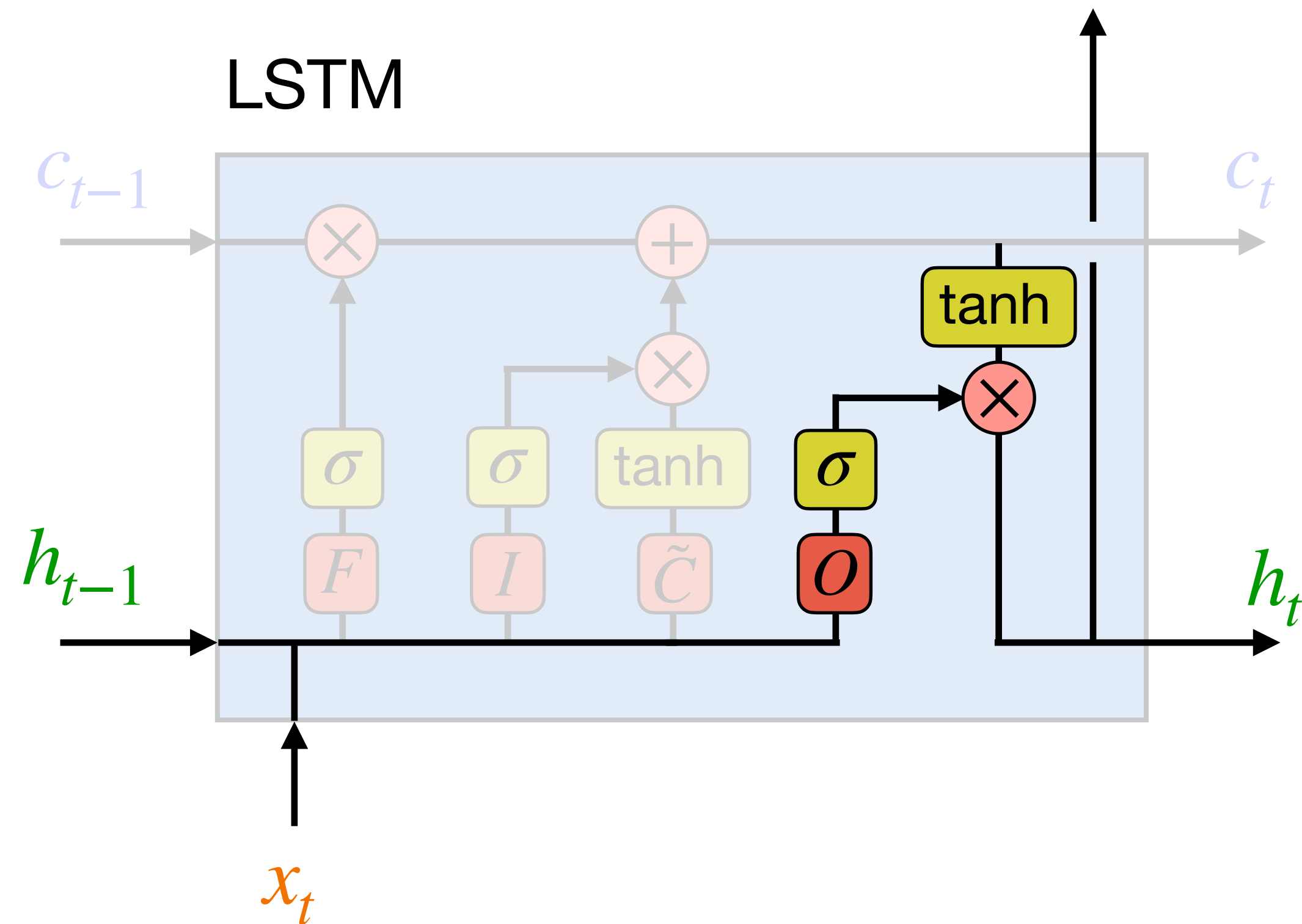


$$o_t = \sigma(W_o x_{t-1} + U_o h_{t-1} + b_o)$$

$$h_t = o_t \odot \tanh(c_t)$$

LSTM: фильтр выхода

Контролирует выход текущего шага.



$$o_t = \sigma(W_o x_{t-1} + U_o h_{t-1} + b_o)$$

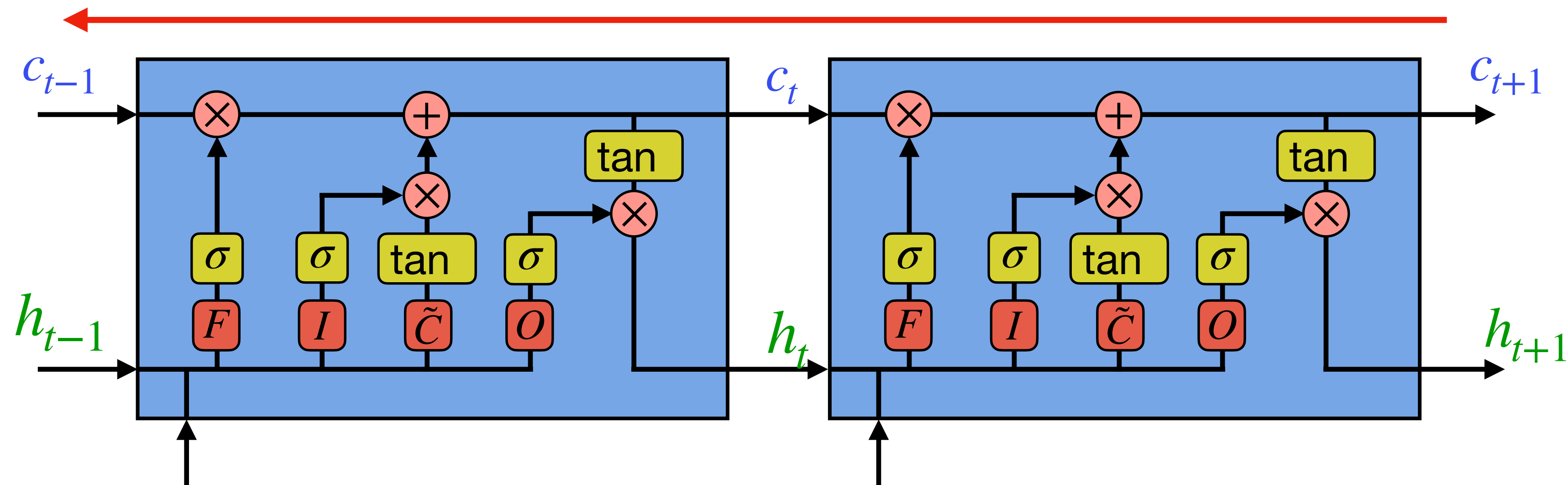
$$h_t = o_t \odot \tanh(c_t)$$

Учитель ведет урок, посвященный рекуррентным моделям. _

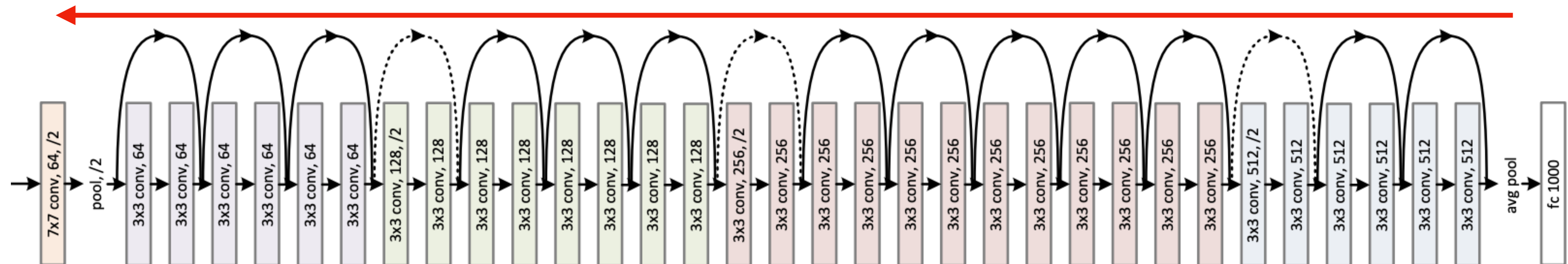
Начало нового предложения.
Надо вспомнить, что речь идет об учителе и уроке.

Градиенты не затухают

Градиенты свободно протекают через весь блок

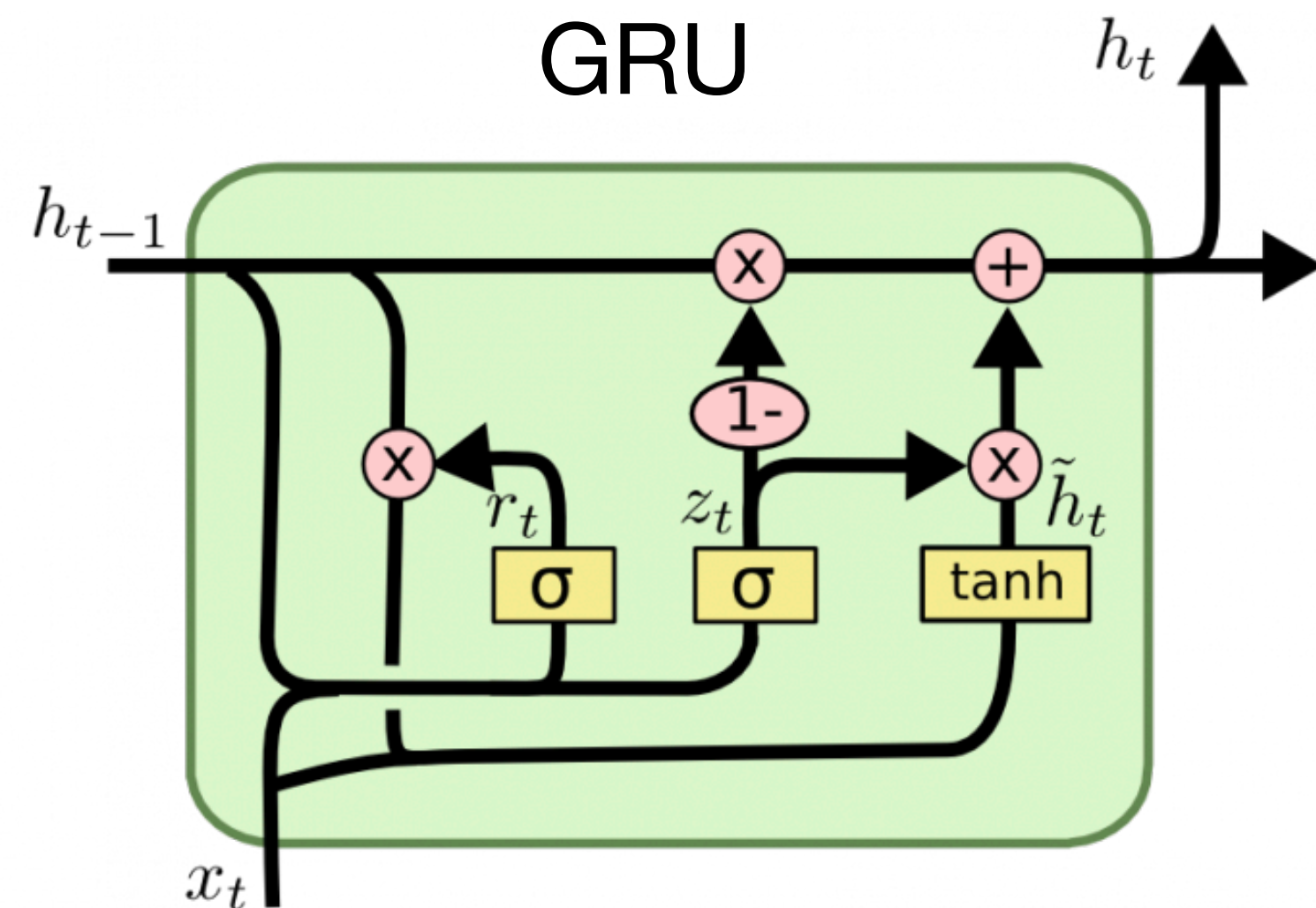


То же самое происходит в ResNet



Gated recurrent units (GRU)

Наиболее успешная уменьшенная вариация LSTM



$$z_t = \sigma(W_z x_{t-1} + U_z h_{t-1} + b_z)$$

$$r_t = \sigma(W_r x_{t-1} + U_r h_{t-1} + b_r)$$

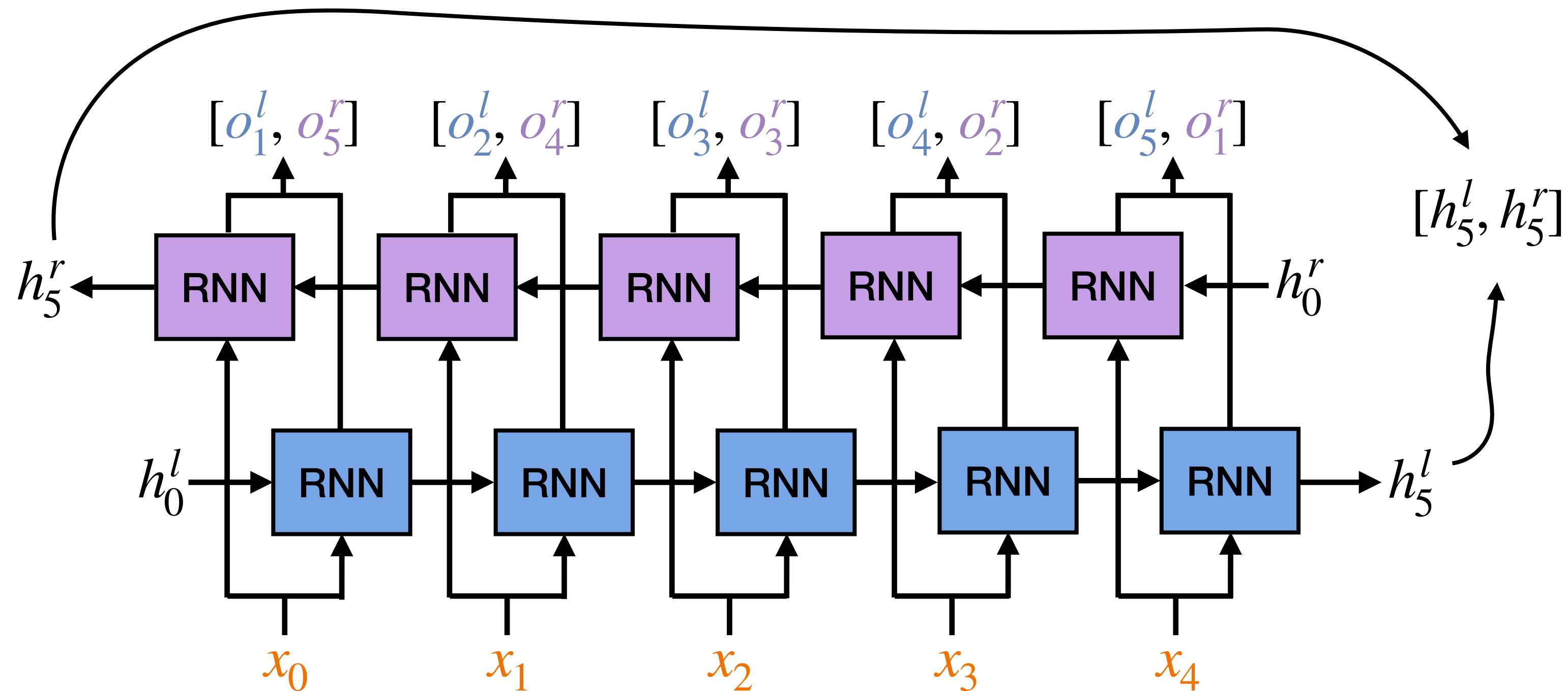
$$\tilde{h}_t = \tanh(W_h x_{t-1} + U_h (r_t \odot h_{t-1}) + b_h)$$

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

Фильтры забывания и входа объединены для уменьшения числа параметров

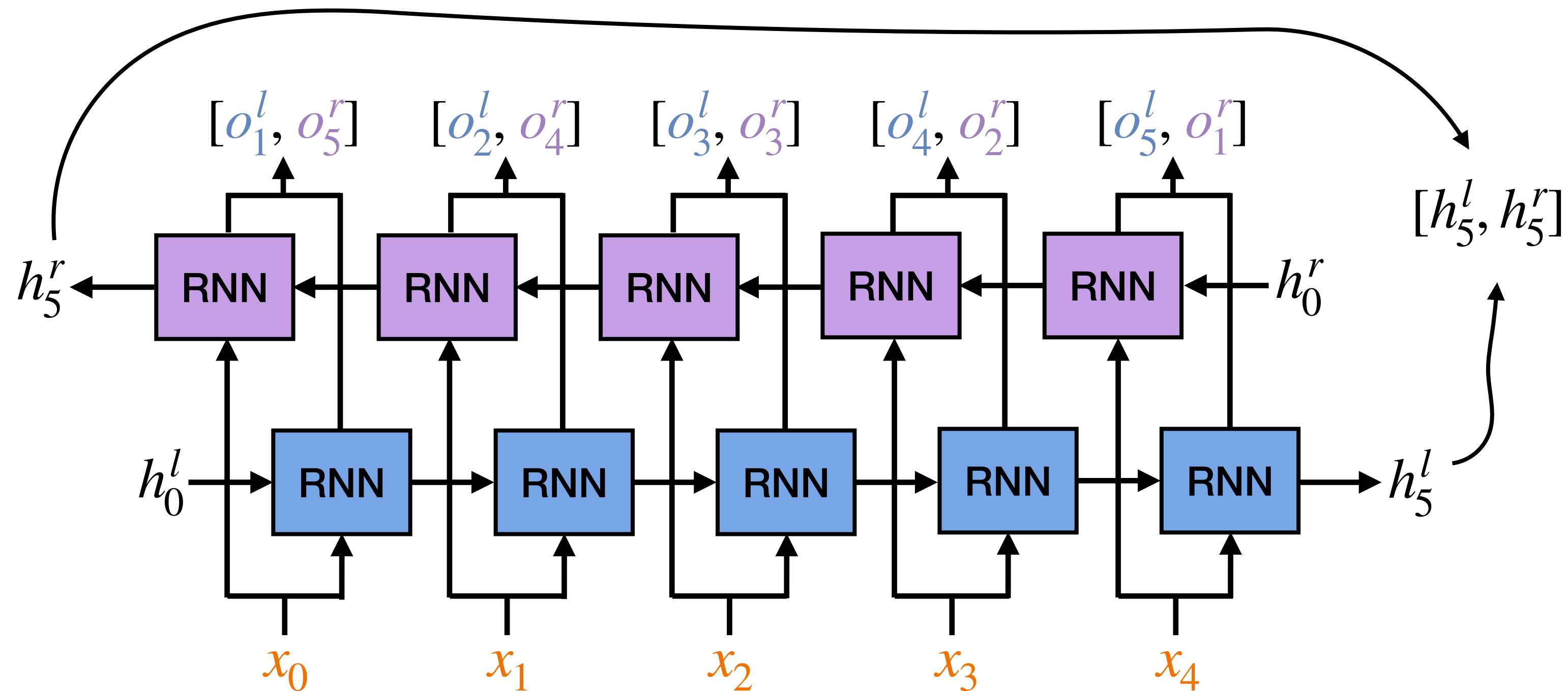
Двунаправленные рекуррентные сети (biRNN)

- Читает текст **слева направо** и **справа налево**.



Двунаправленные рекуррентные сети (biRNN)

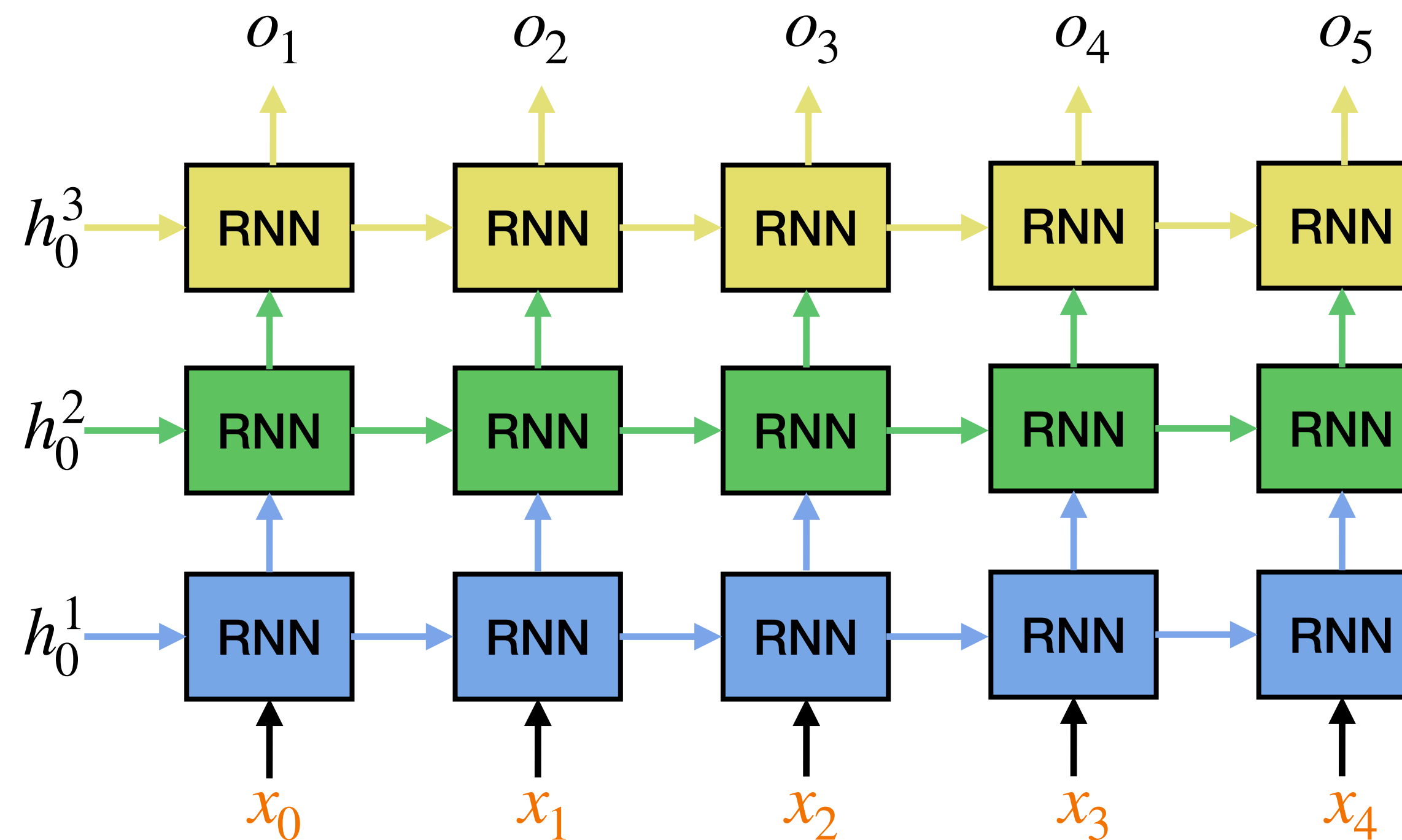
- Читает текст **слева направо** и **справа налево**.



- Имеет в два раза больше параметров.
- Бесполезна для задачи генерации.

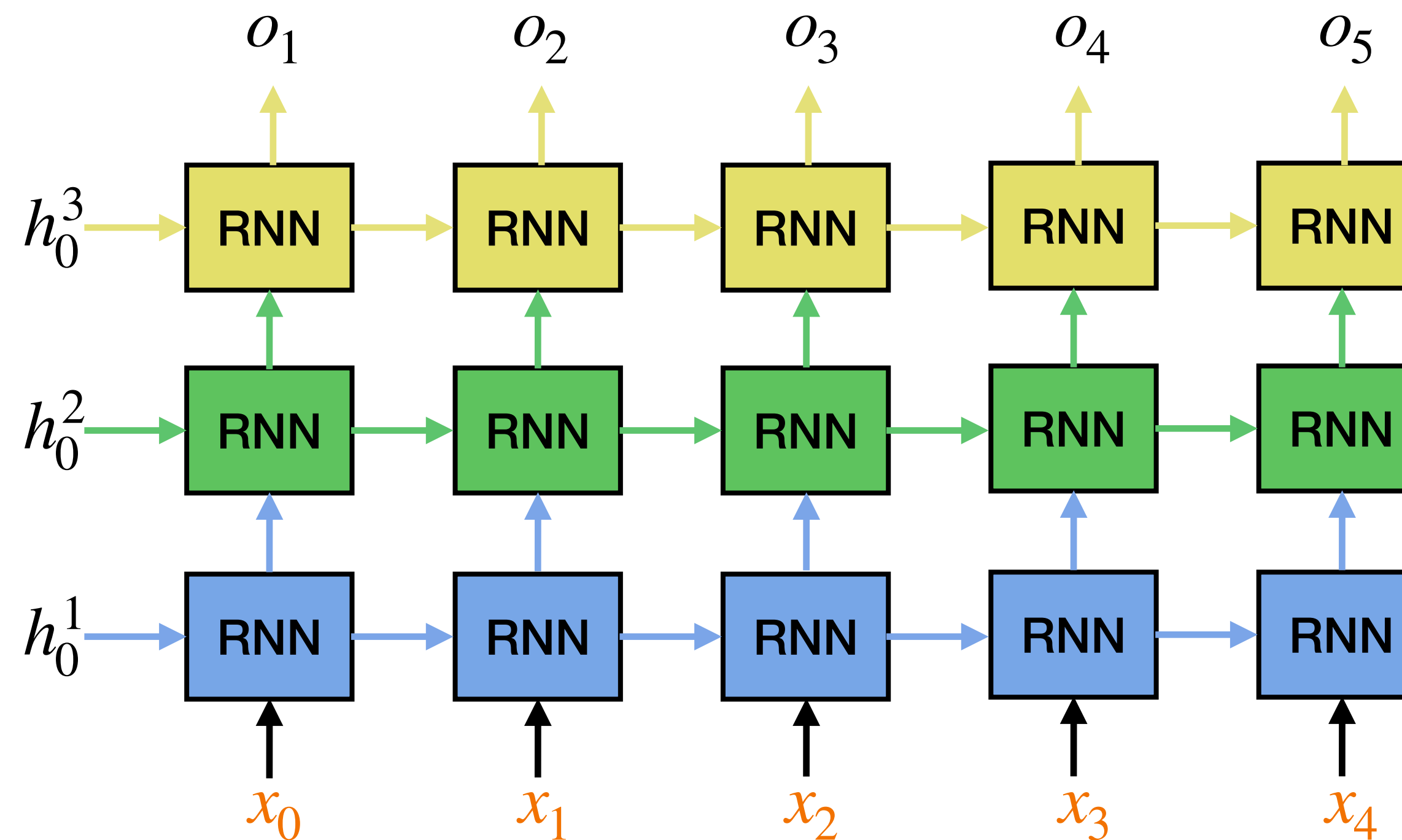
Многослойные рекуррентные сети (biRNN)

- Выходы текущего слоя являются входами следующего.
- Извлекаются более сложные признаки.



Многослойные рекуррентные сети (biRNN)

- Выходы текущего слоя являются входами следующего.
- Извлекаются более сложные признаки.



- Число параметров увеличивается в число слоев раз.
- Обычно ограничиваются **двумя** слоями.

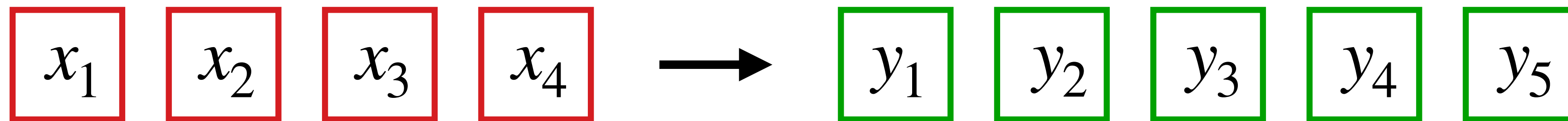
Трансформер

План

- Задача Seq2seq или что не так с RNN
- Механизм внимания
- Трансформер
- Метрики качества Seq2seq
- Методы выбора токенов из их распределения

Задача 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:

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

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

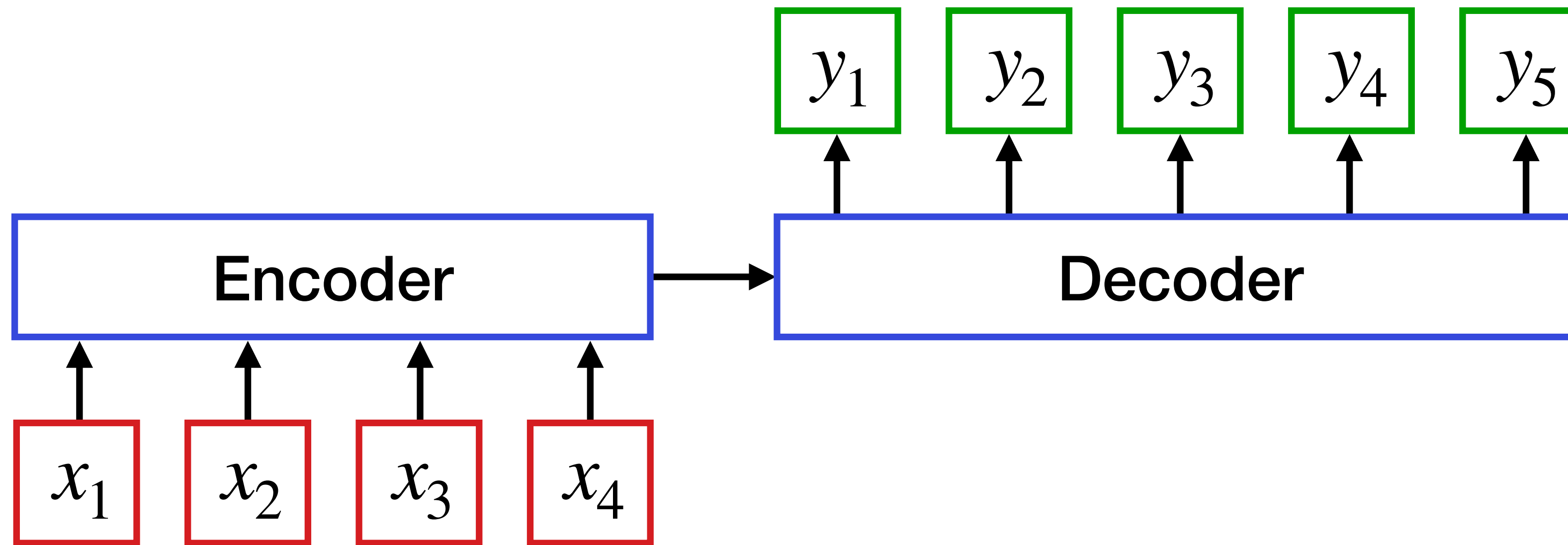
y – новая (выходная) последовательность

учимся приближать



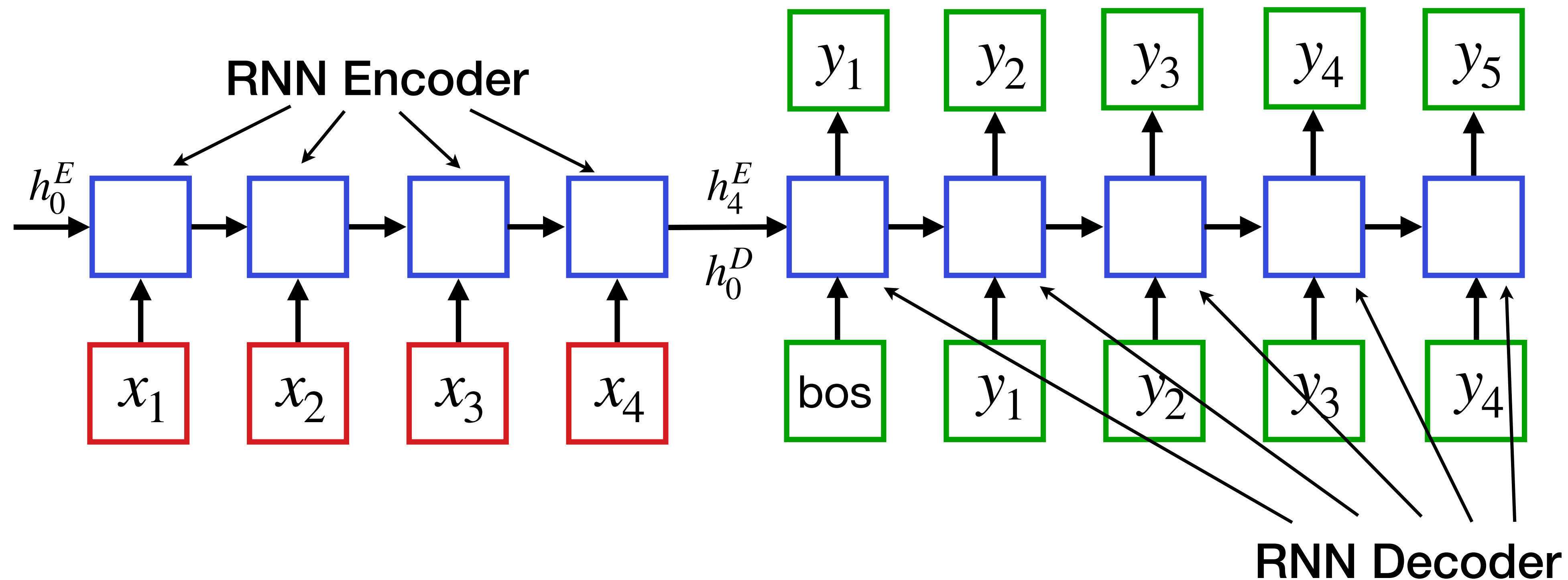
Стандартный подход

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



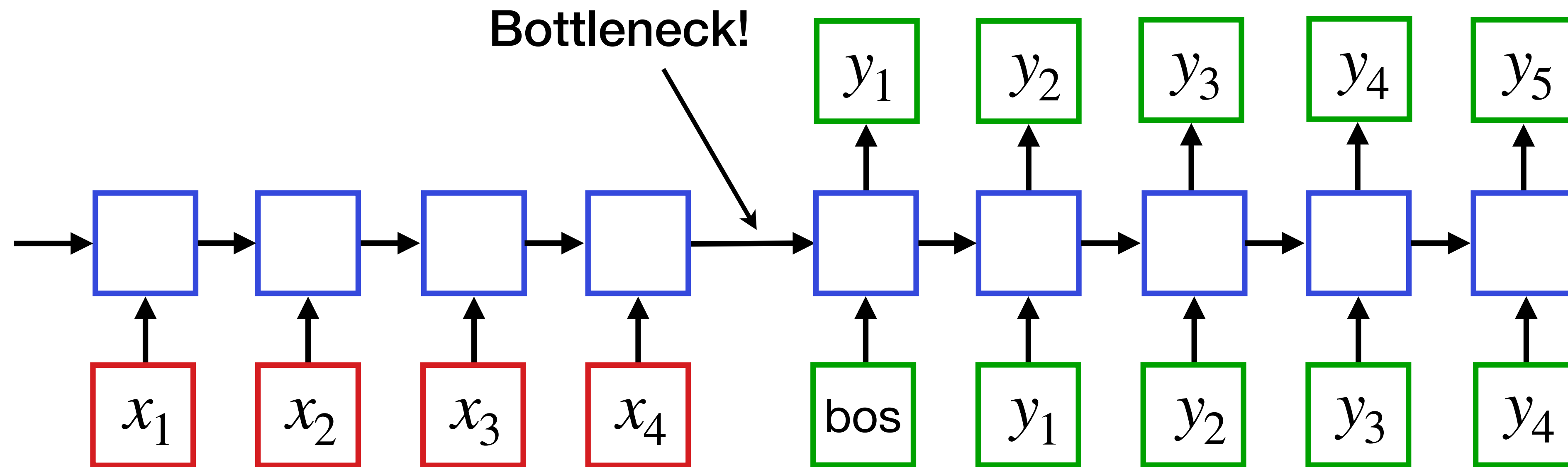
RNN подход

- Одной RNN моделью кодируем текст и получаем вектор состояния h^E
- Второй RNN моделью генерируем последовательность



RNN подход: проблема

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



Механизм внимания

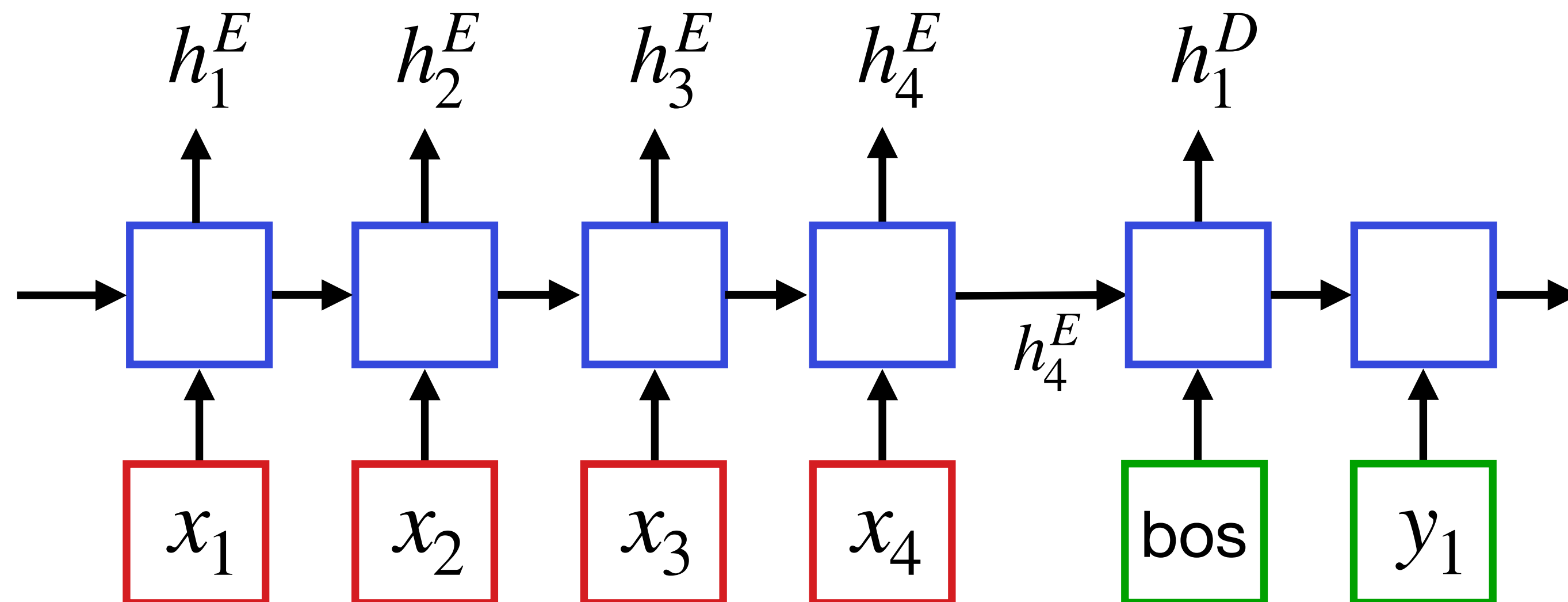
Идея:

Позволим декодеру "смотреть" на произвольные выходы кодировщика

Механизм внимания

Идея:

Позволим декодеру "смотреть" на произвольные выходы кодировщика

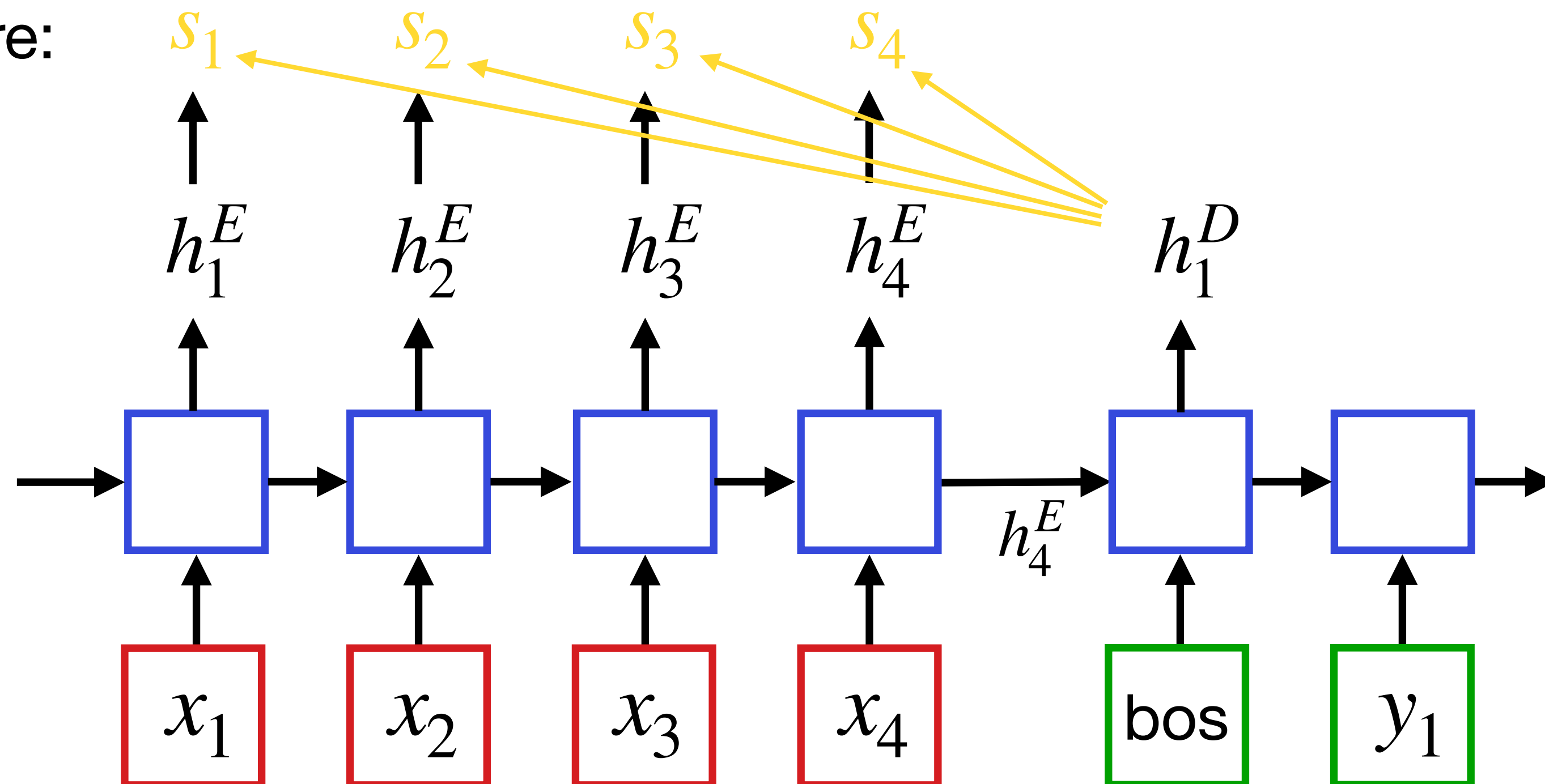


Механизм внимания

Идея:

Позволим декодеру "смотреть" на произвольные выходы кодировщика

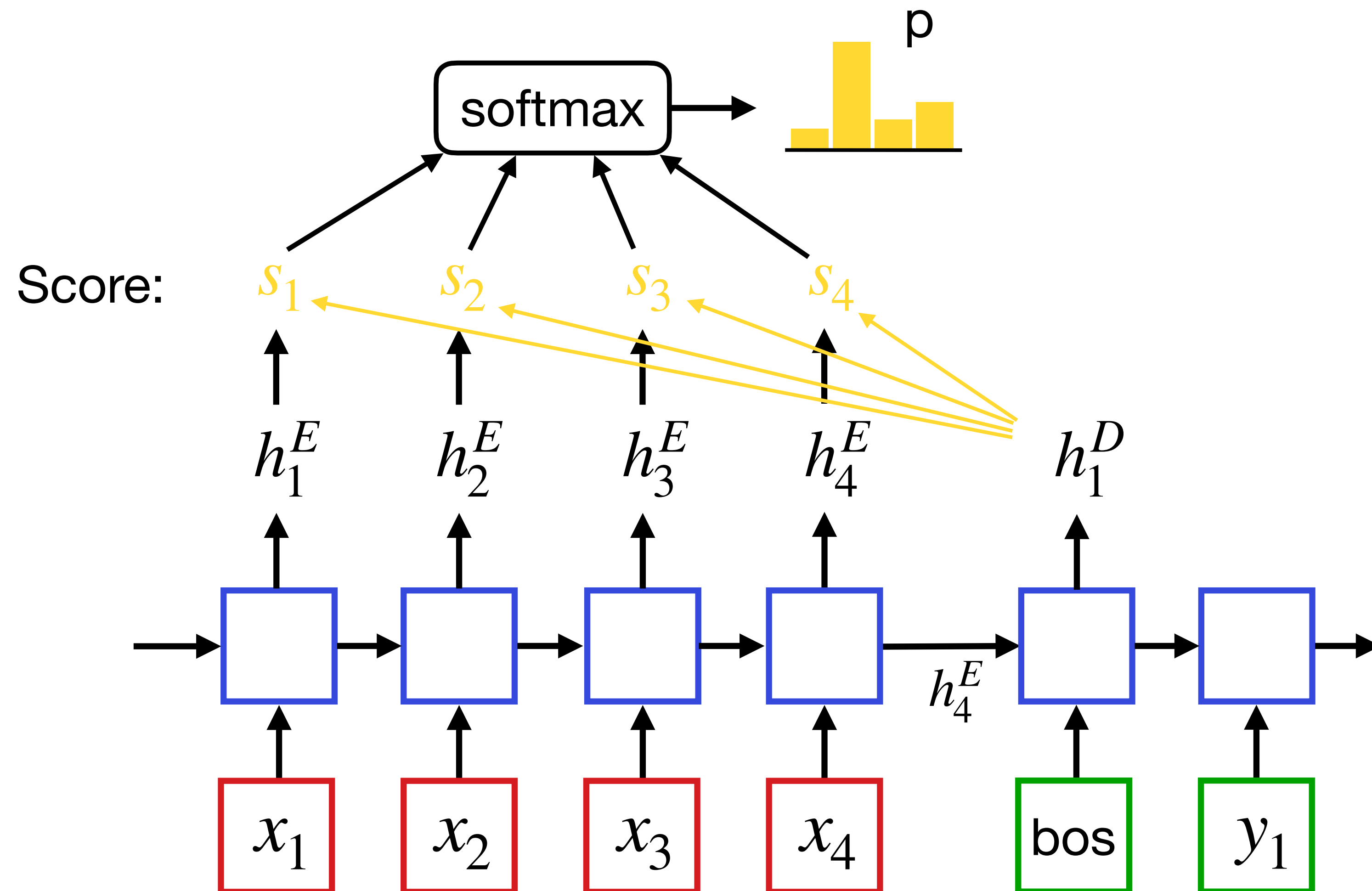
Score:



Механизм внимания

Идея:

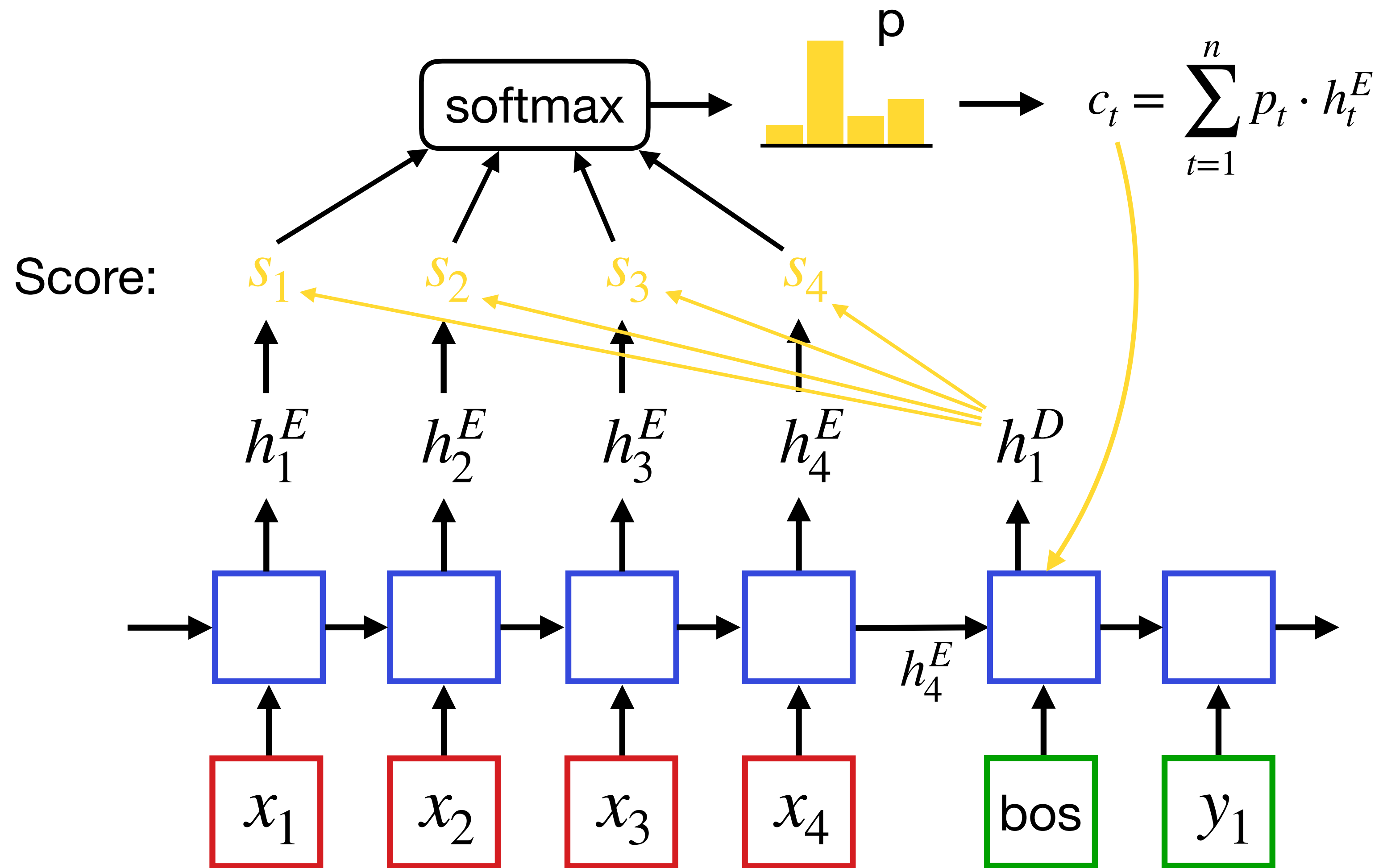
Позволим декодеру "смотреть" на произвольные выходы кодировщика



Механизм внимания

Идея:

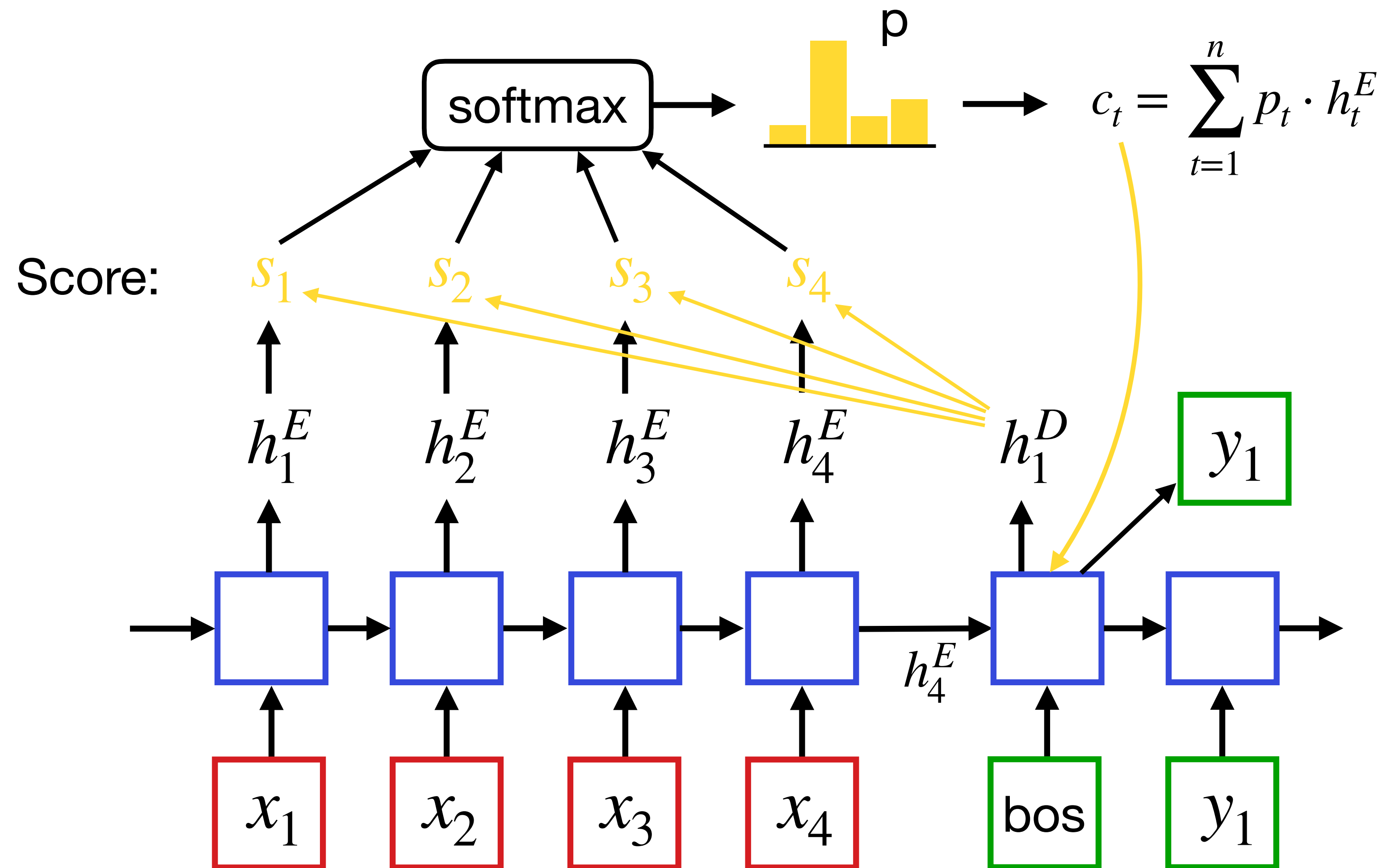
Позволим декодеру "смотреть" на произвольные выходы кодировщика



Механизм внимания

Идея:

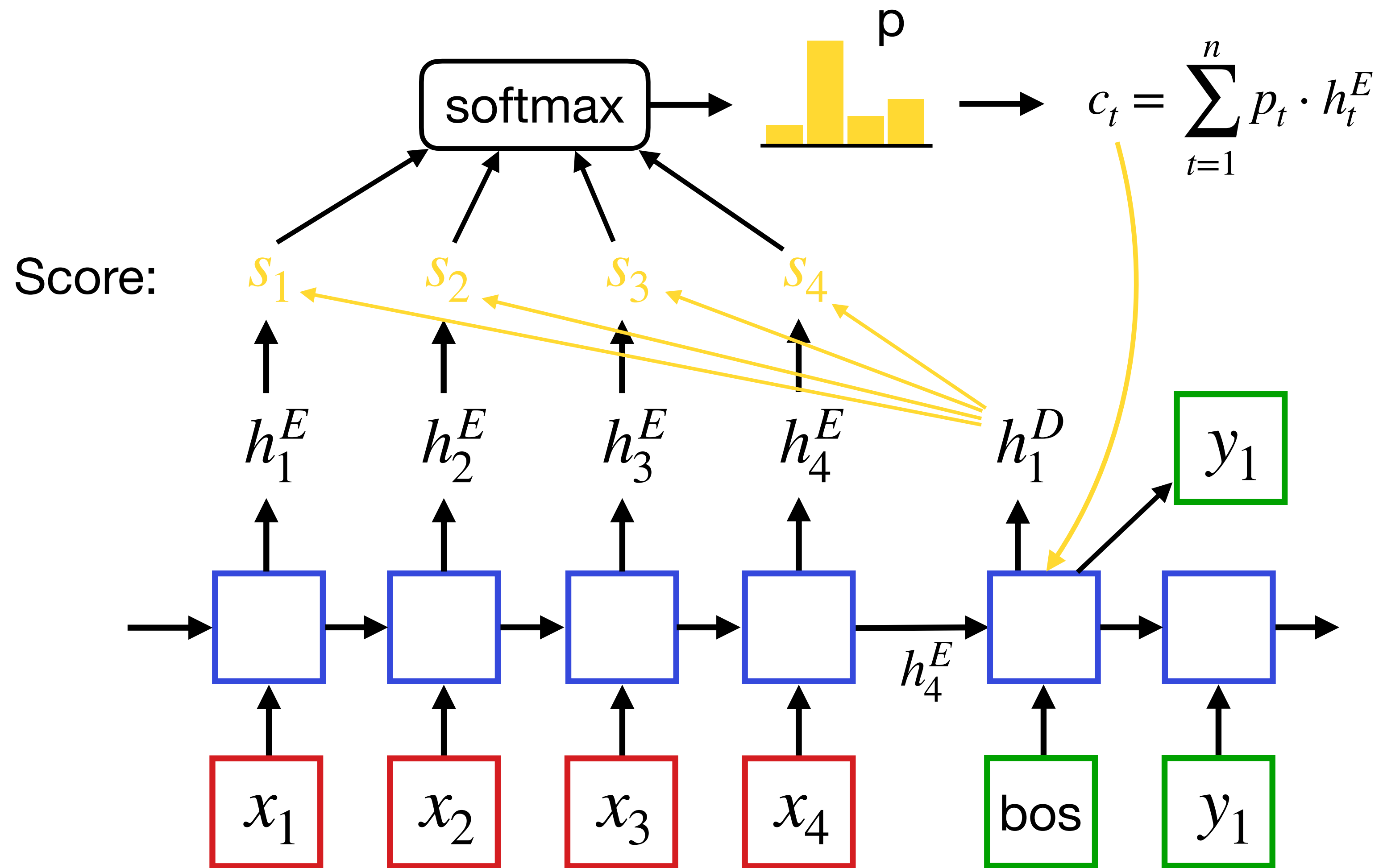
Позволим декодеру "смотреть" на произвольные выходы кодировщика



Механизм внимания

Идея:

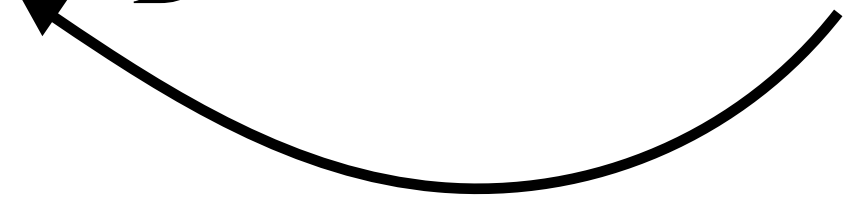
Позволим декодеру "смотреть" на произвольные выходы кодировщика



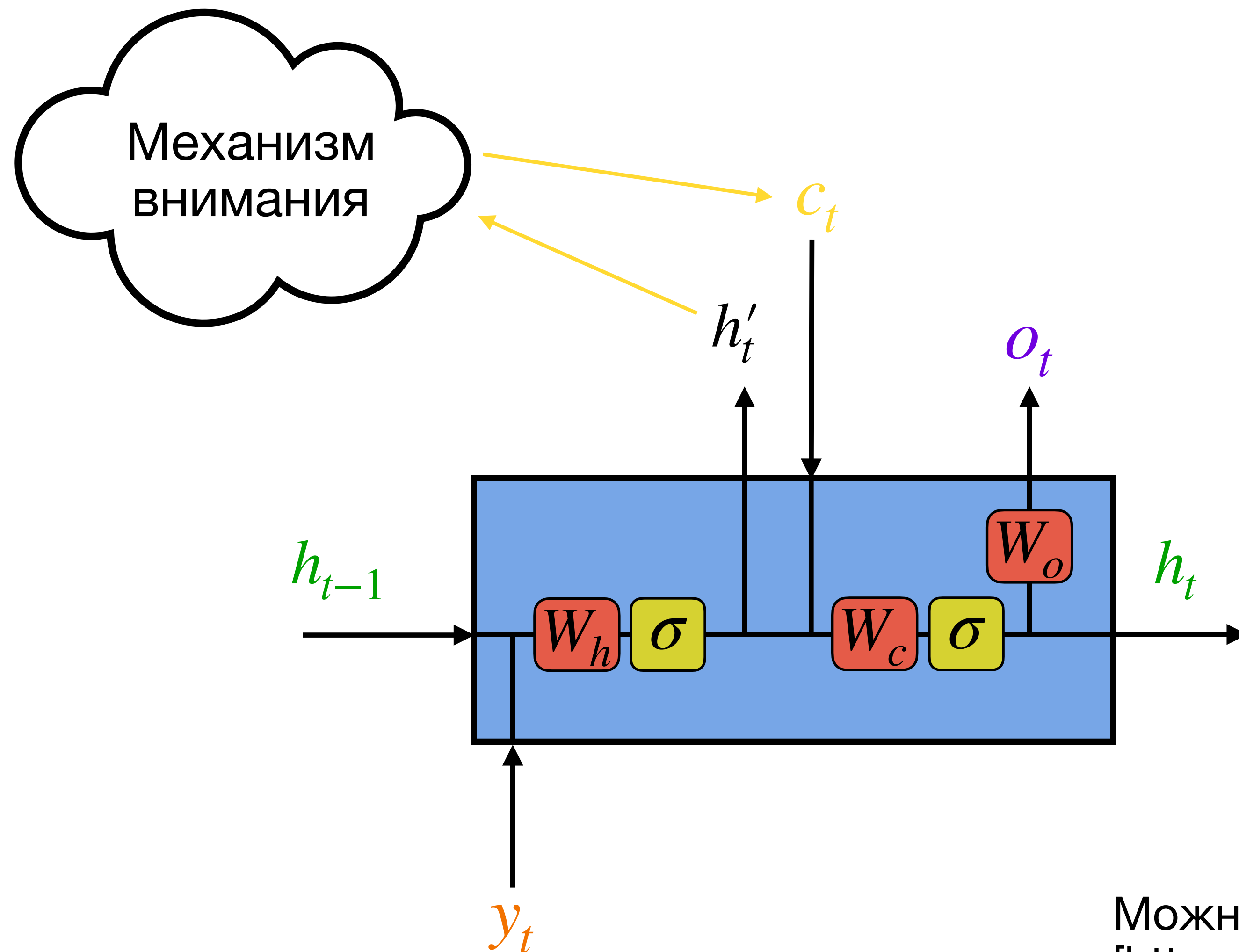
- Что такое Score?
- Как обрабатывается c_t ?

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



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

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

$$o_t = W_o h_t + b_o$$

Можно комбинировать и по-другому:

[<https://arxiv.org/abs/1409.0473>, Bahdanau et al, 2014]

[<https://arxiv.org/abs/1508.04025>, Luong et al, 2015]

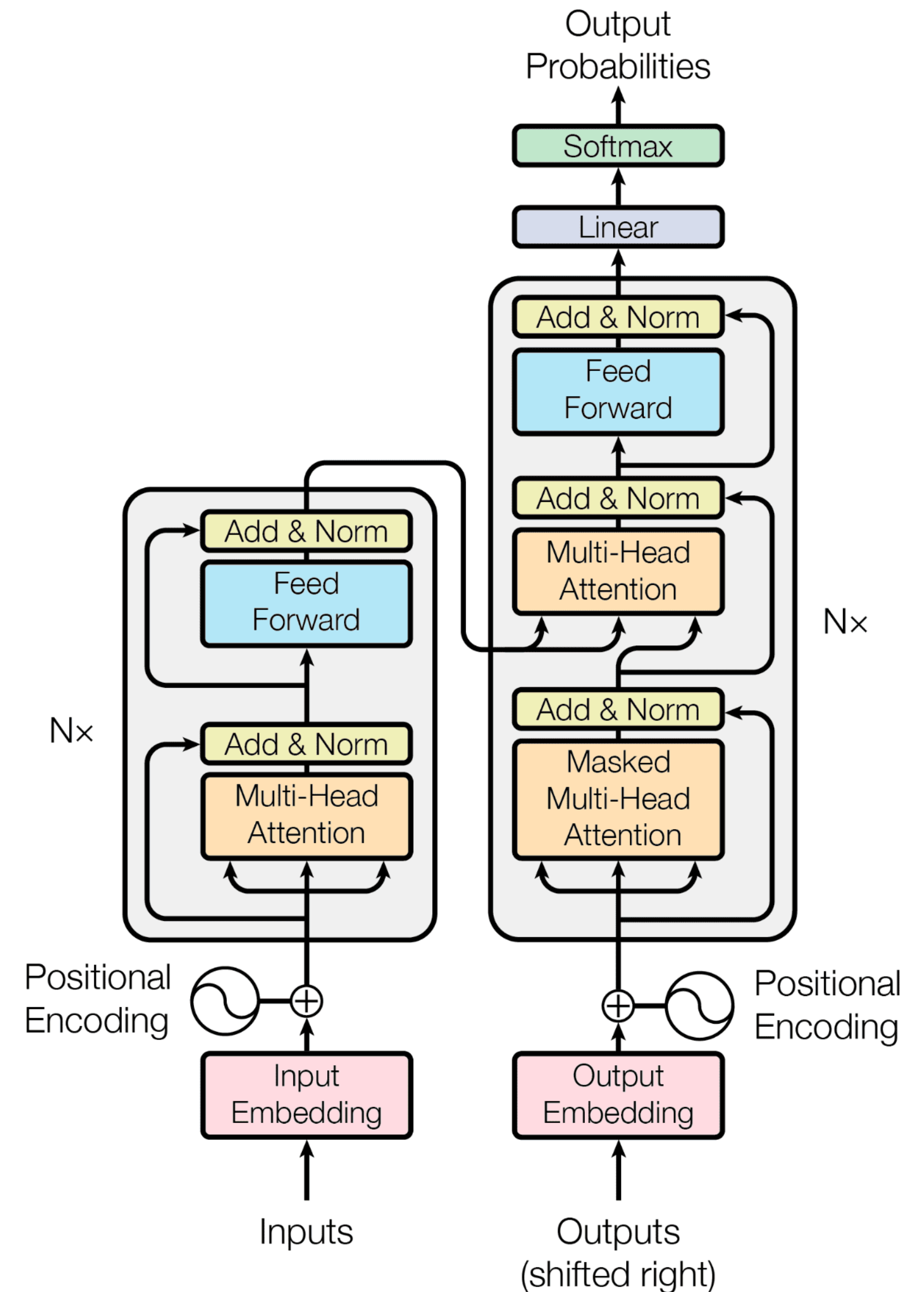
Transformer

Attention is all you need

- Самая популярная архитектура с 2017 года
- 130к+ цитат у оригинальной статьи

Компоненты:

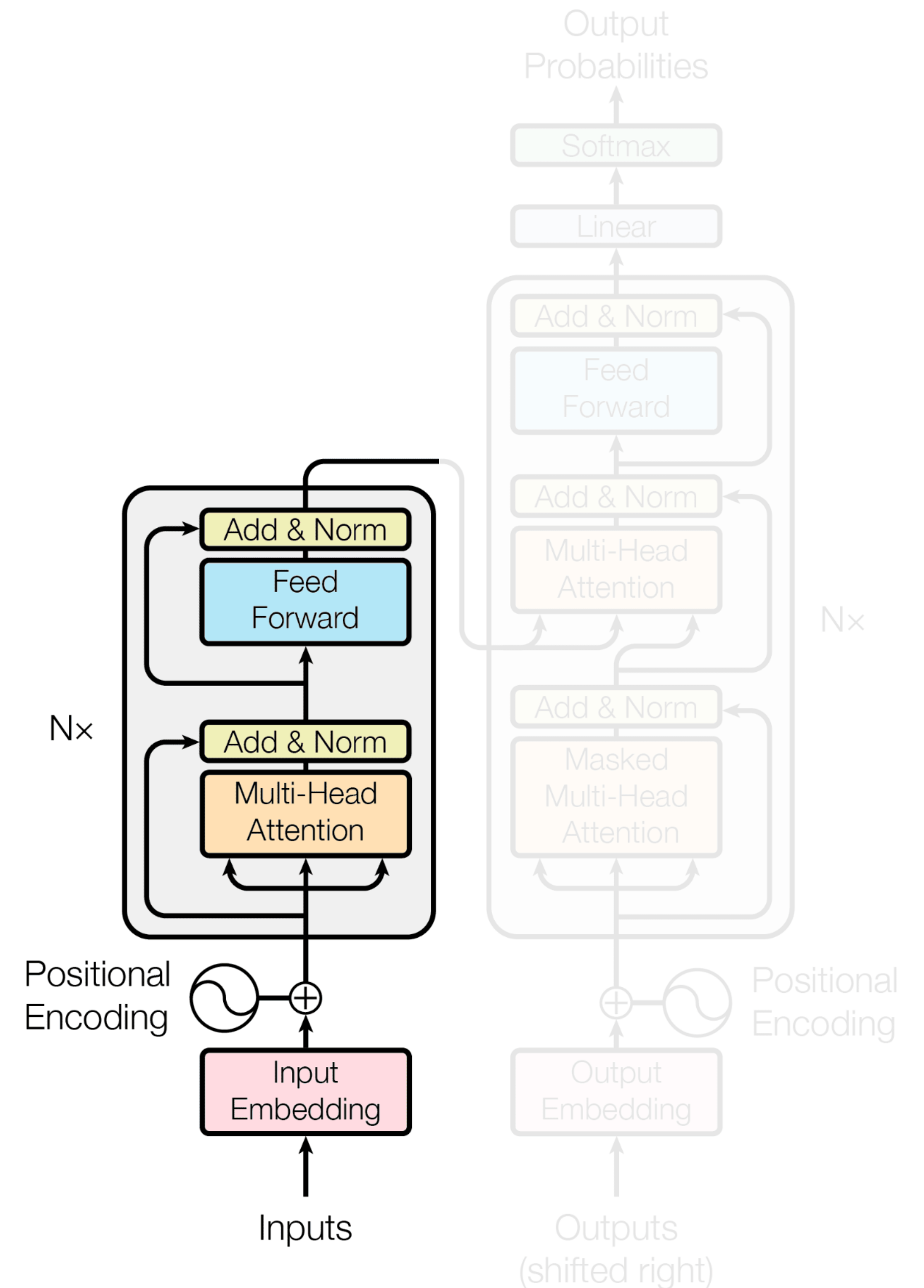
1. Multi-Head Self-attention
2. Feed Forward Network
3. Masked Multi-Head Self-attention
4. Positional Encodings



Encoder

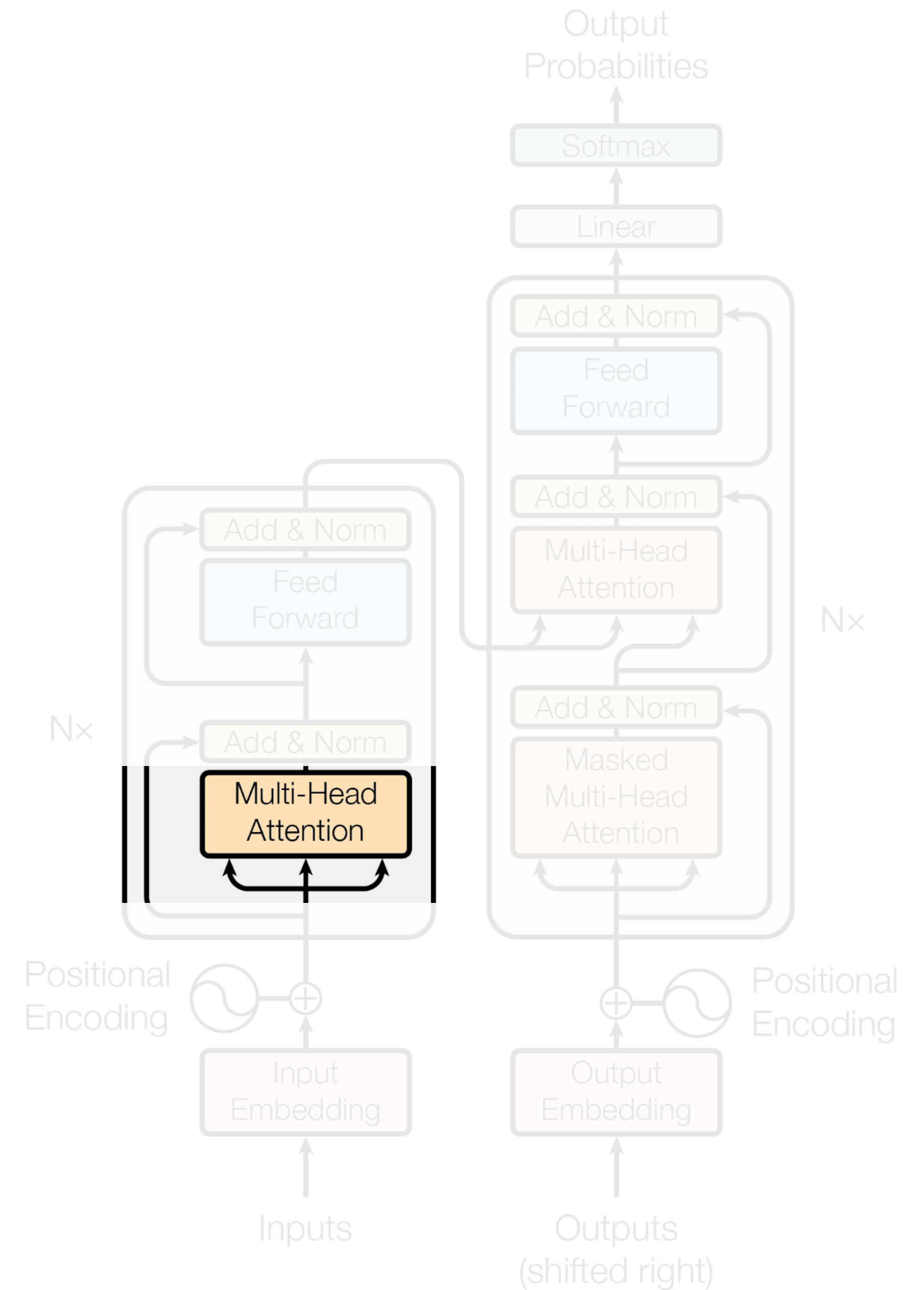
Encoder нужен для извлечения признаков из последовательности

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



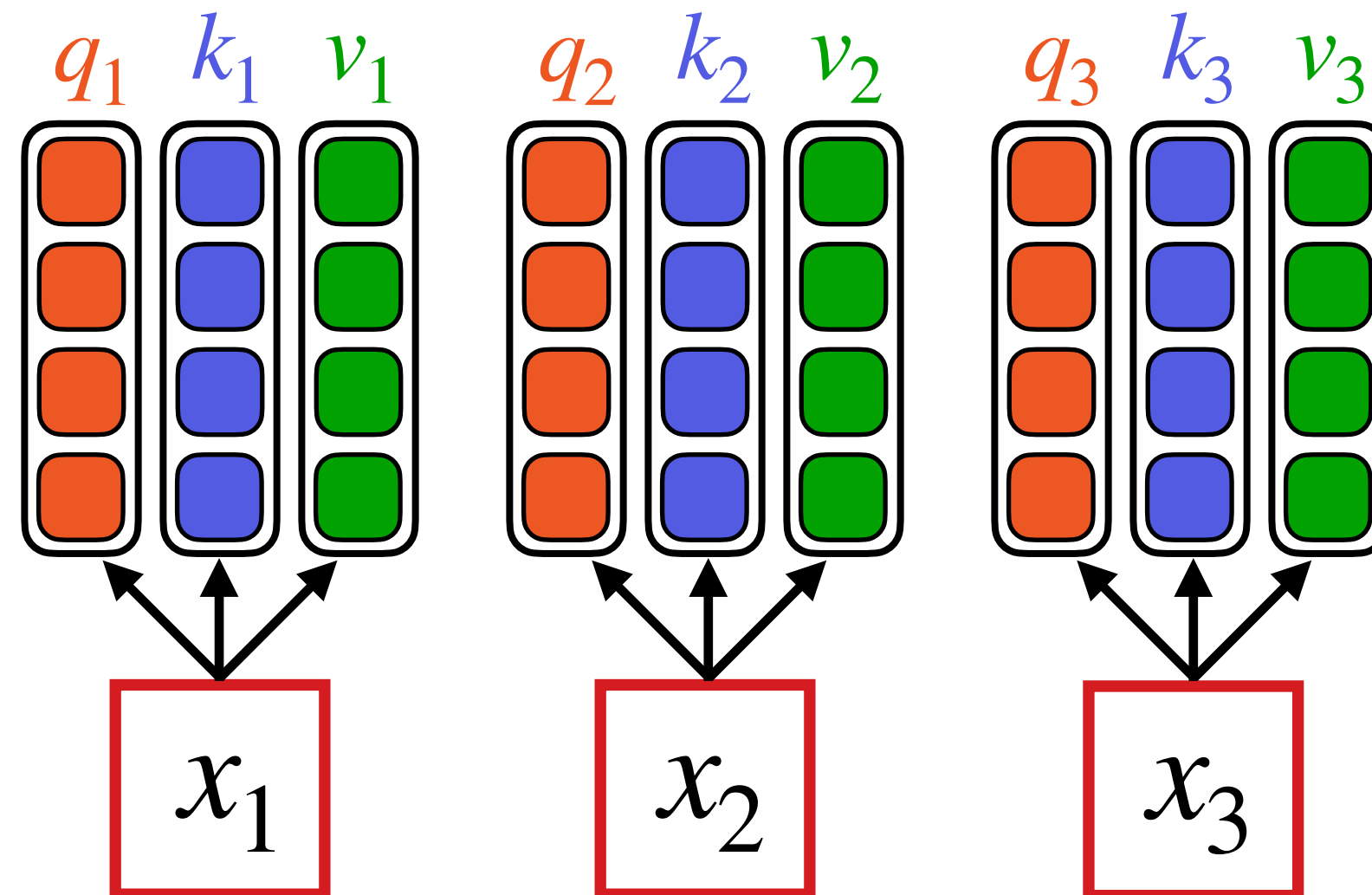
Self-attention

Позволяет каждому токенту посмотреть на все остальные.

q – запрос (query)

k – ключ (key)

v – значение (value)



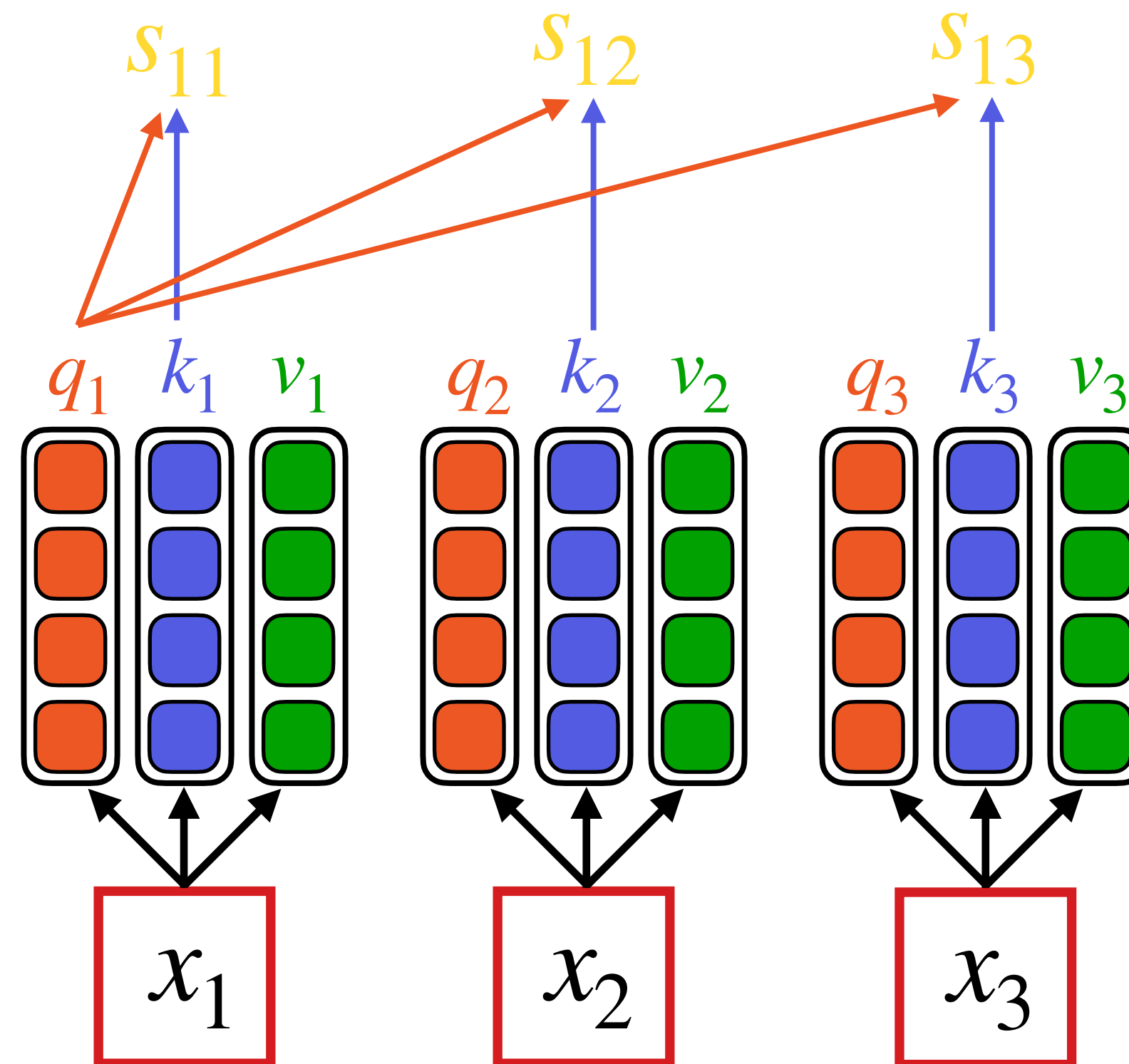
Self-attention

Позволяет каждому токенту посмотреть на все остальные.

q – запрос (query)

k – ключ (key)

v – значение (value)



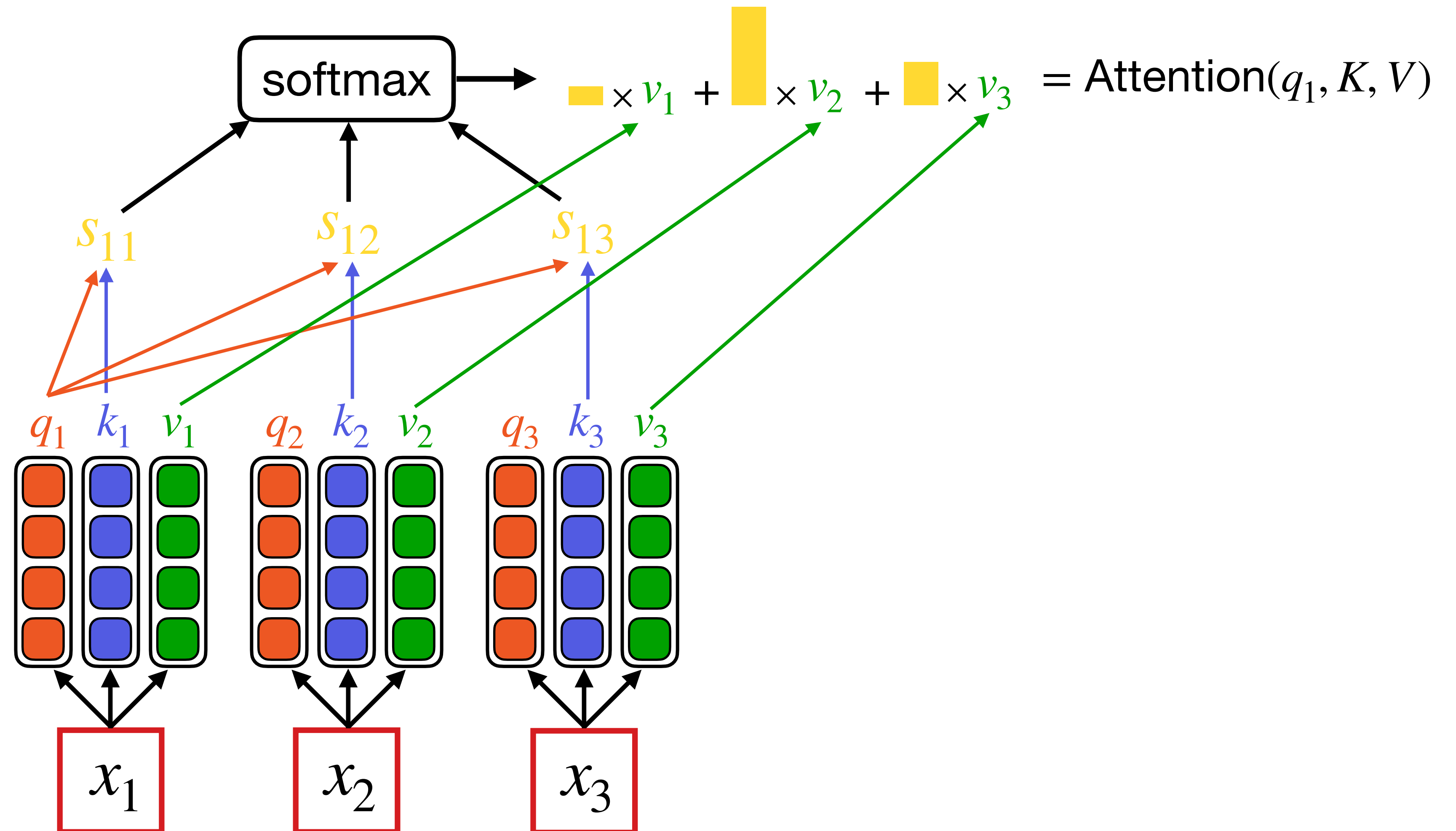
Self-attention

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: $Q = W_q x + b_q$

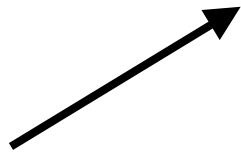
Key: $K = W_k x + b_k$

Value: $V = W_v x + b_v$

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

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

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

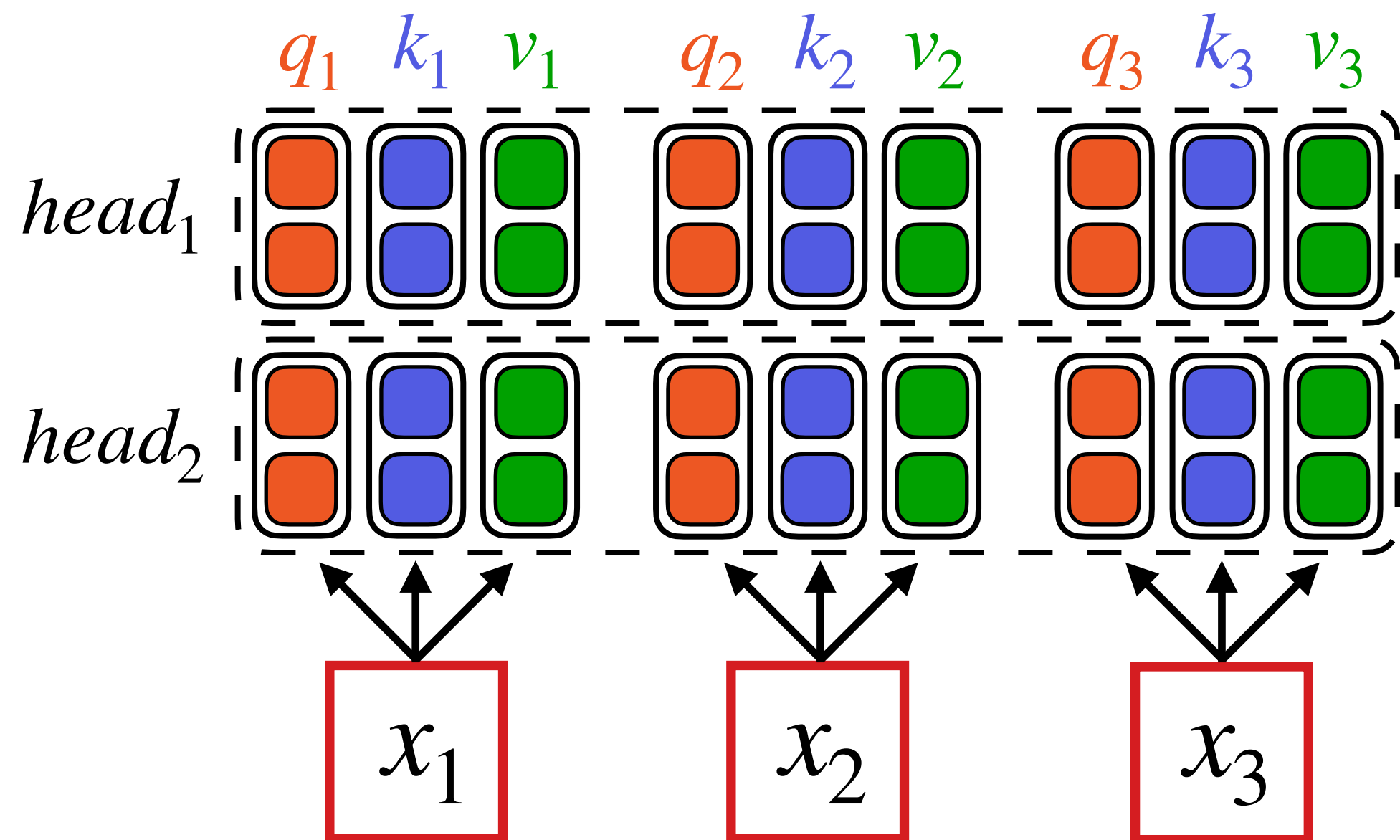


как? что?

Разделим внимание на несколько голов

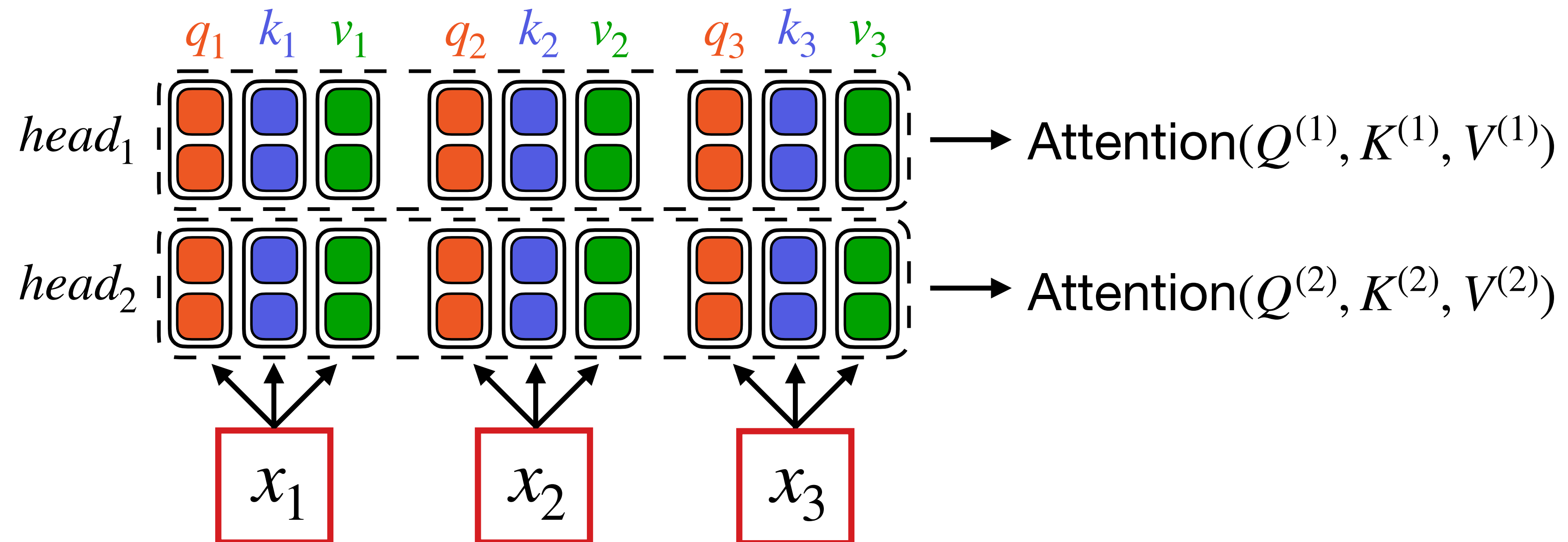
Multi-Head Self-attention

- Делим каждый вектор q , k , v на равные части



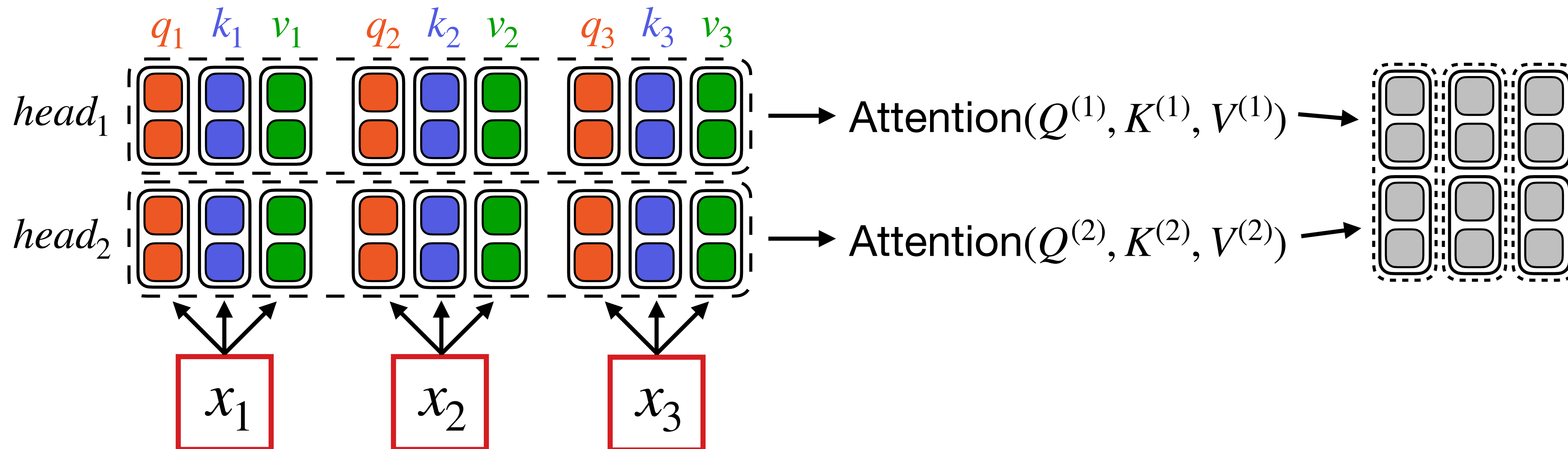
Multi-Head Self-attention

- Делим каждый вектор q , k , v на равные части
- Считаем self-attention для каждой части отдельно



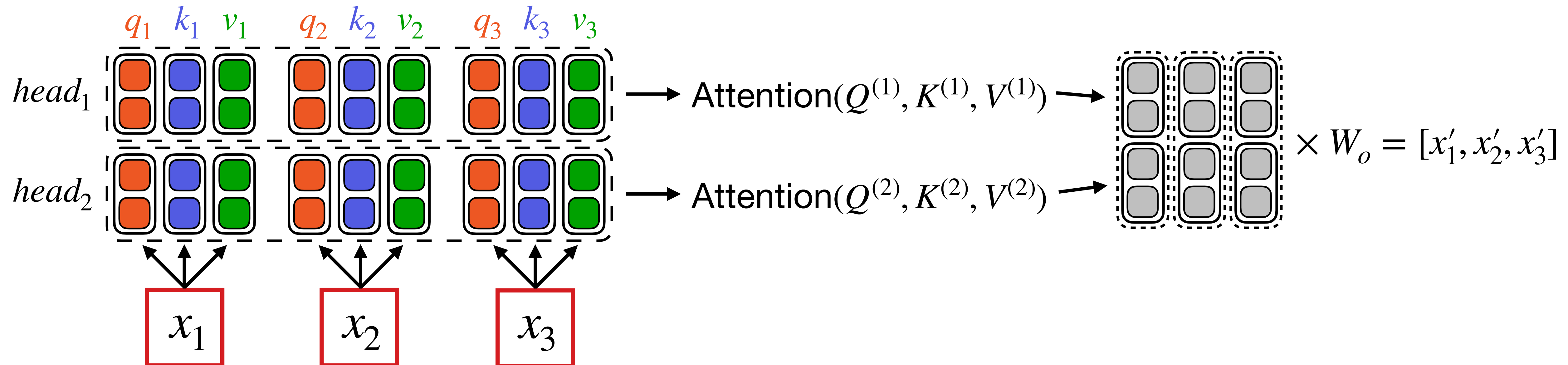
Multi-Head Self-attention

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



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

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

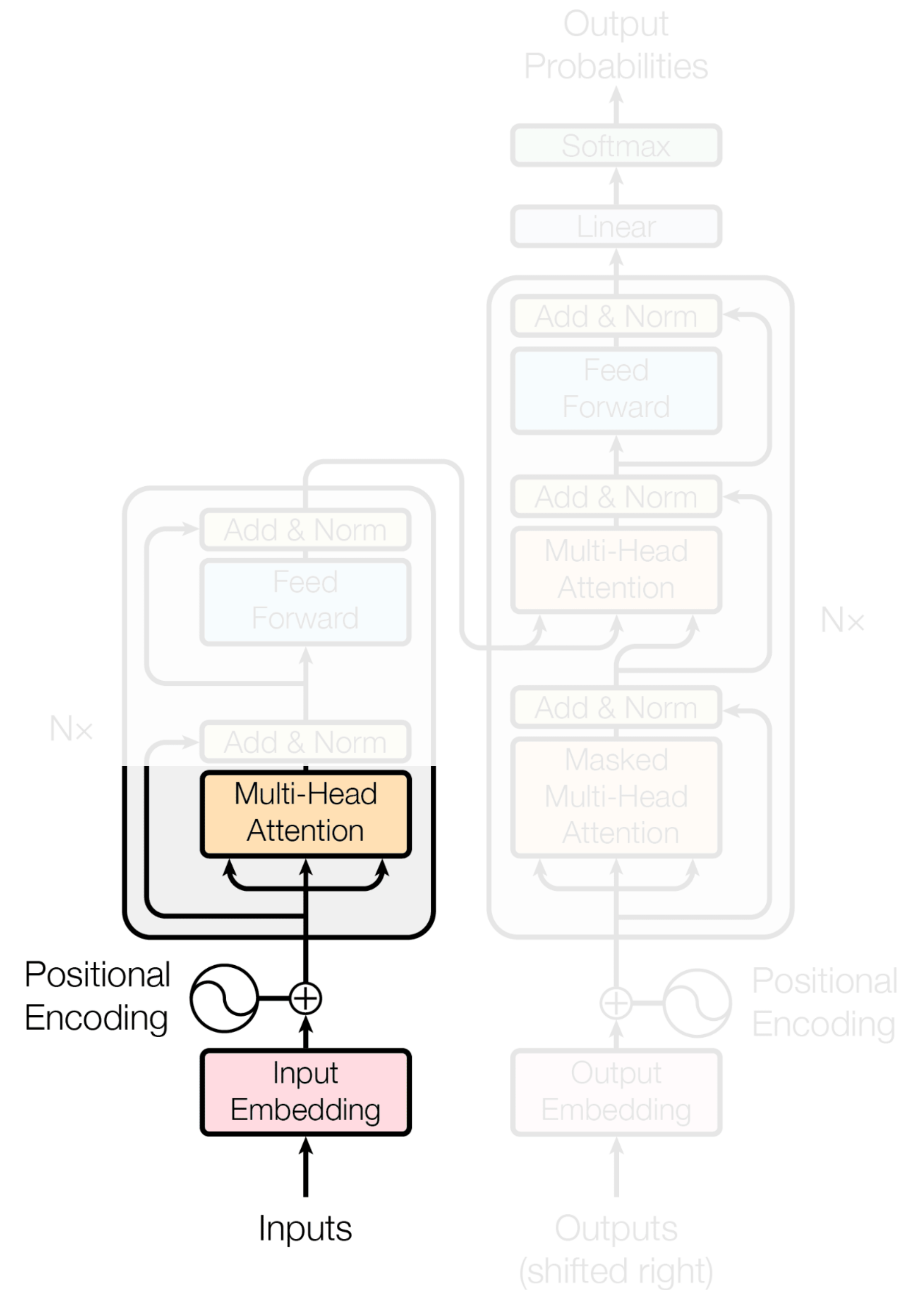
$$\text{MultiHead}(Q, K, V) = W_o \times [\text{head}^{(1)}, \dots, \text{head}^{(h)}]$$

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

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

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!

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

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

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

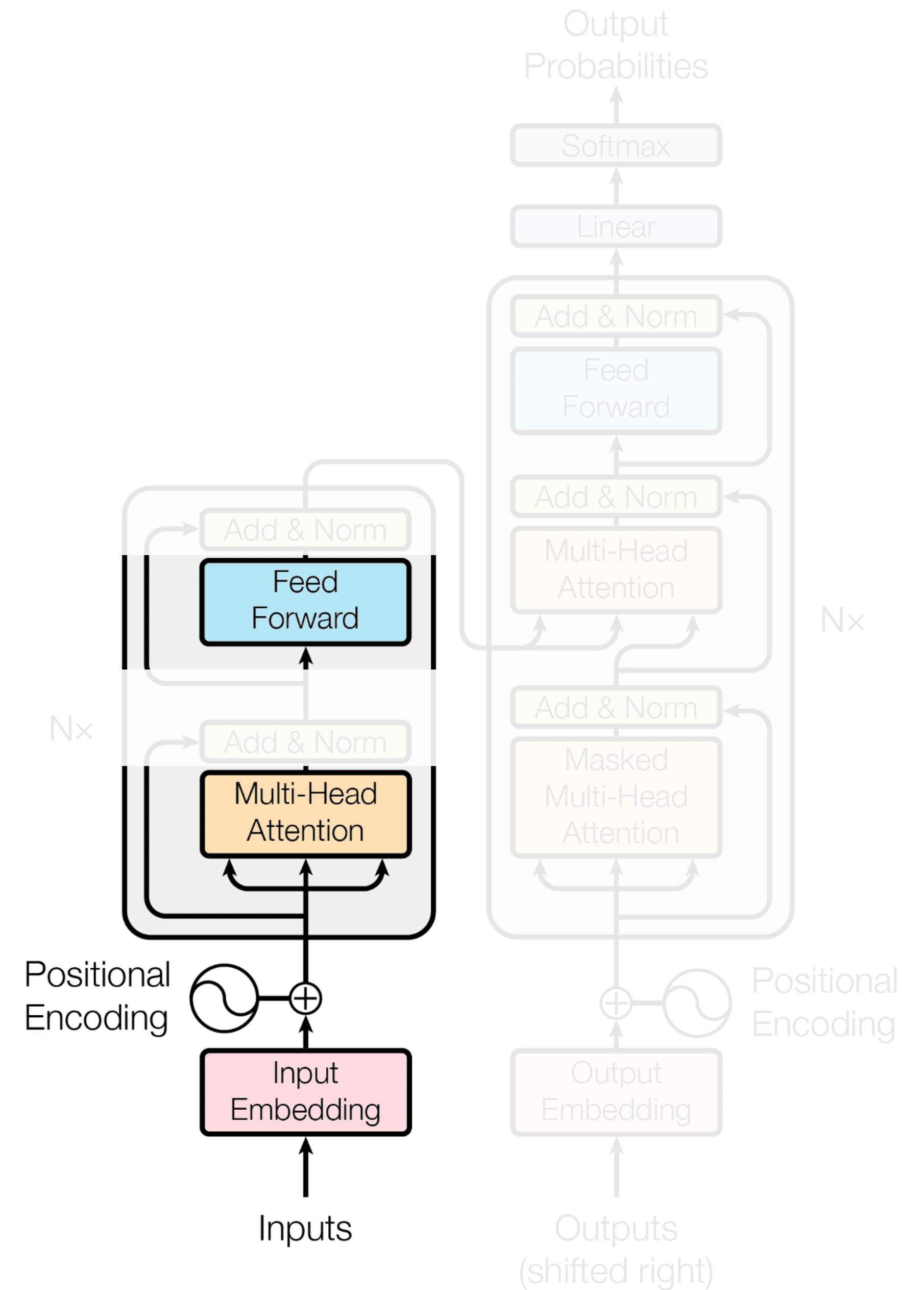
Необходим для учета позиций токенов в тексте.

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

Эмбеддинги позиций учатся вместе с остальными параметрами модели.

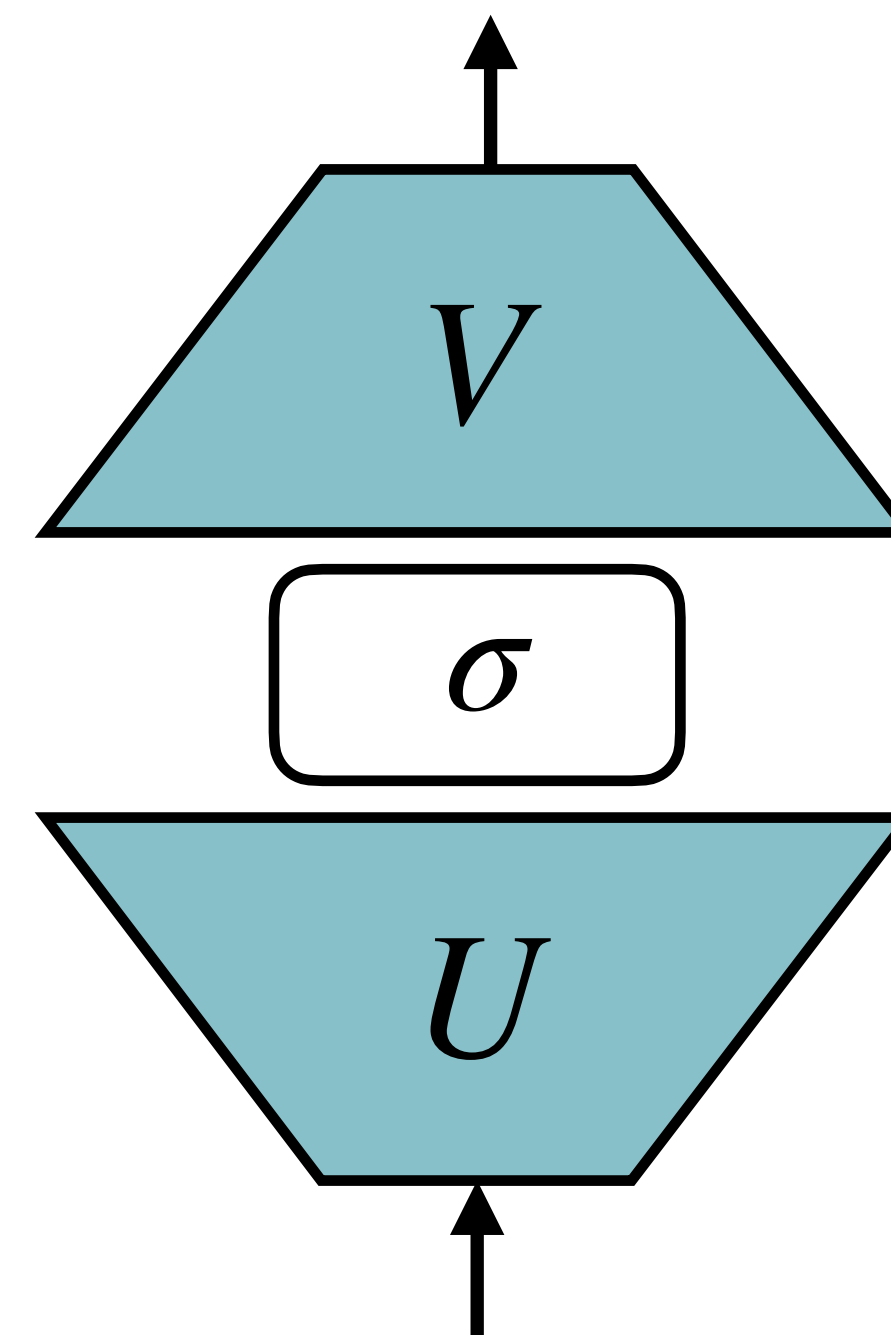
Encoder

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



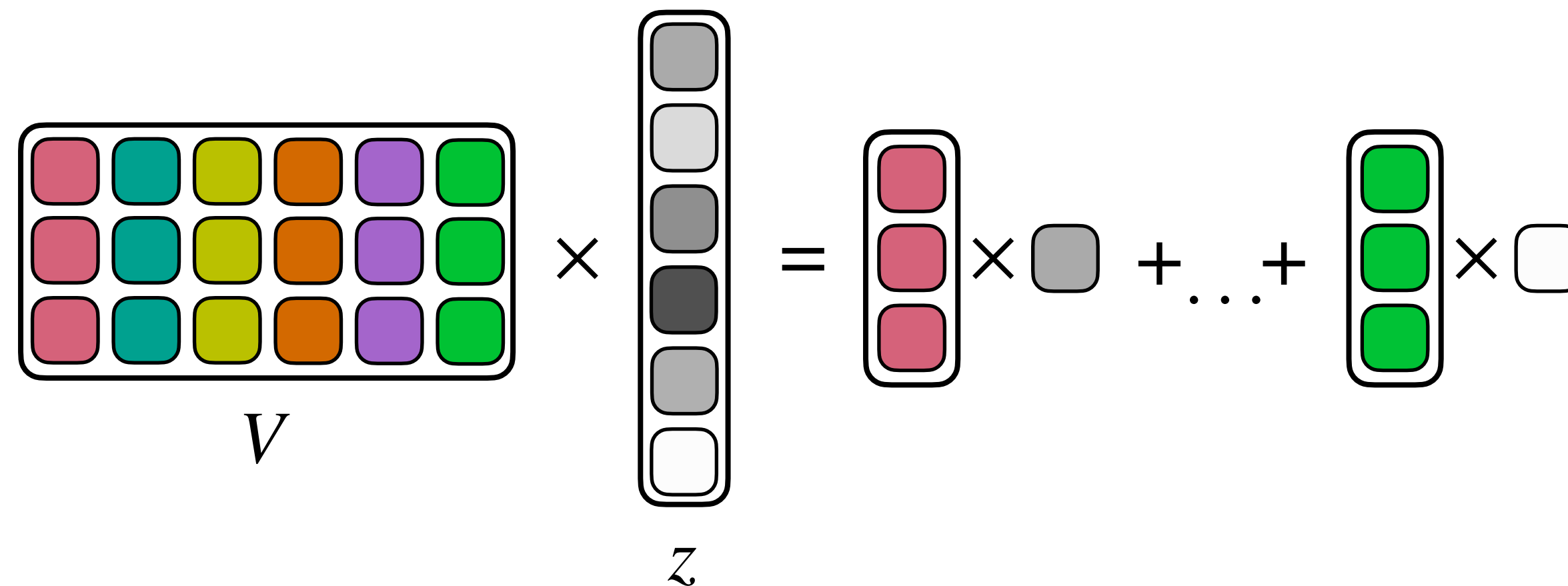
$$U \in \mathbb{R}^{2d \times d}$$

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

FFN: Как ещё можно думать?

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

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

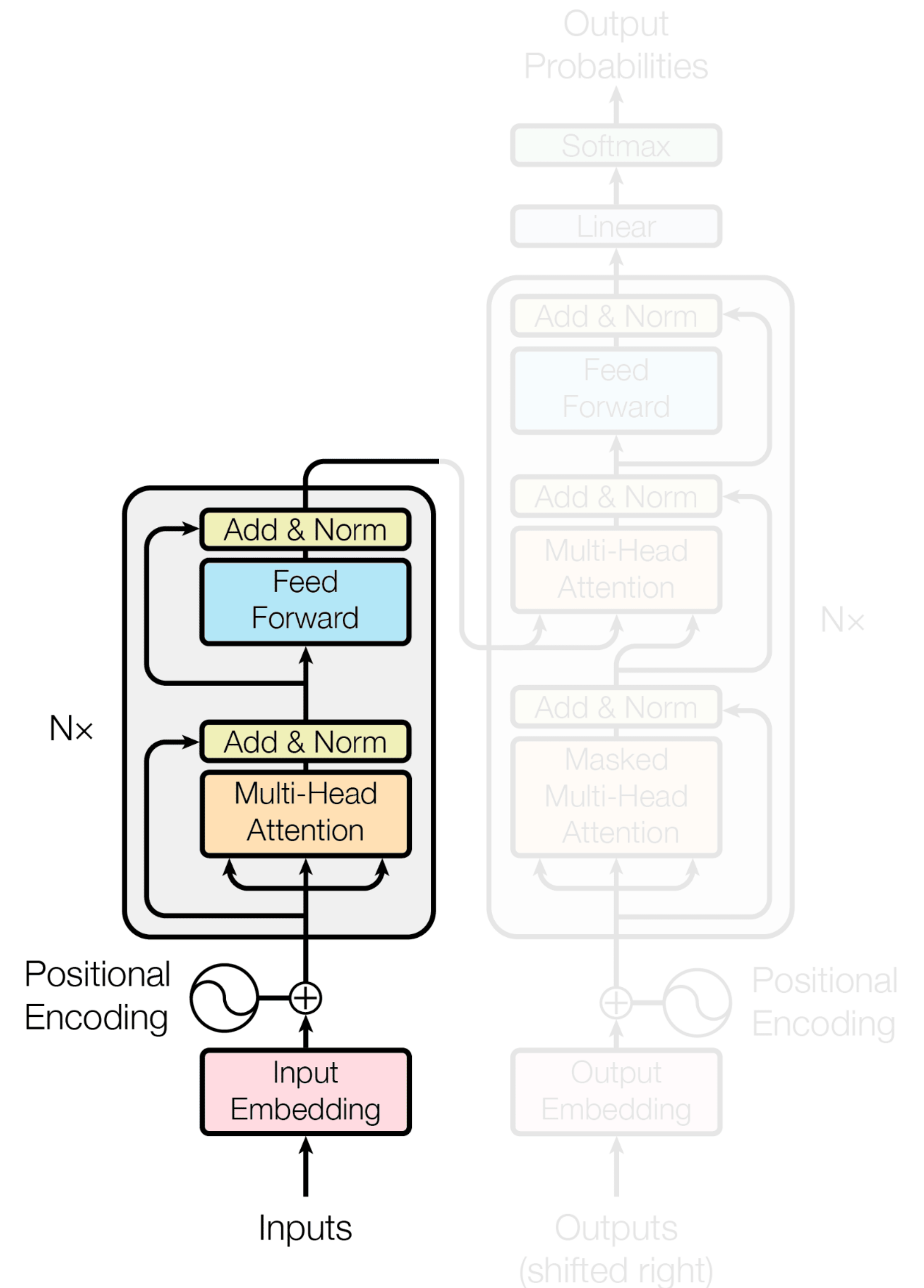


- Столбцы V – определенные факты
- Мы взвешиваем эти факты согласно ключу z

Encoder

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

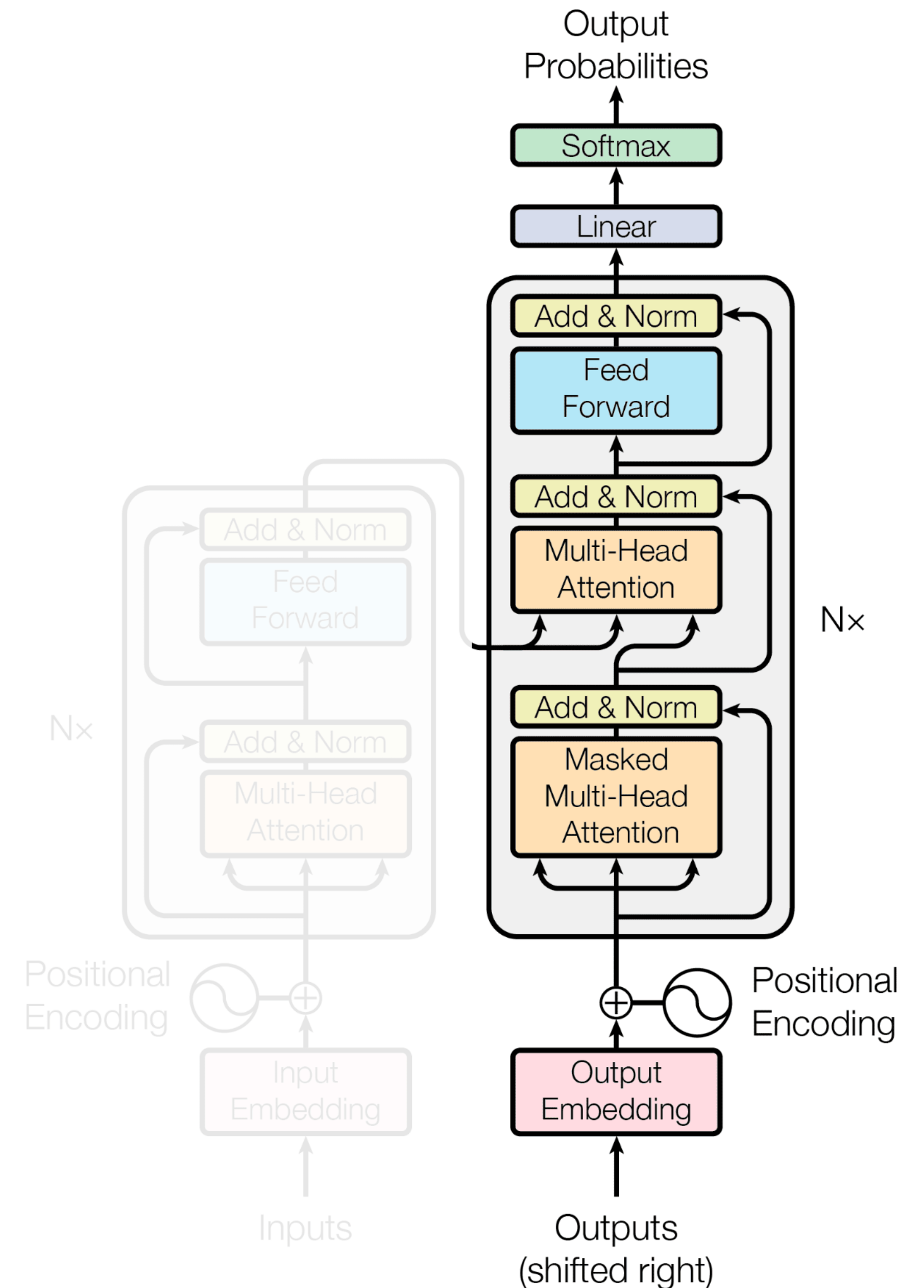
- Добавляем **skip-connection**, чтобы градиенты не затухали
- **Нормализация** для стабилизации обучения



Decoder

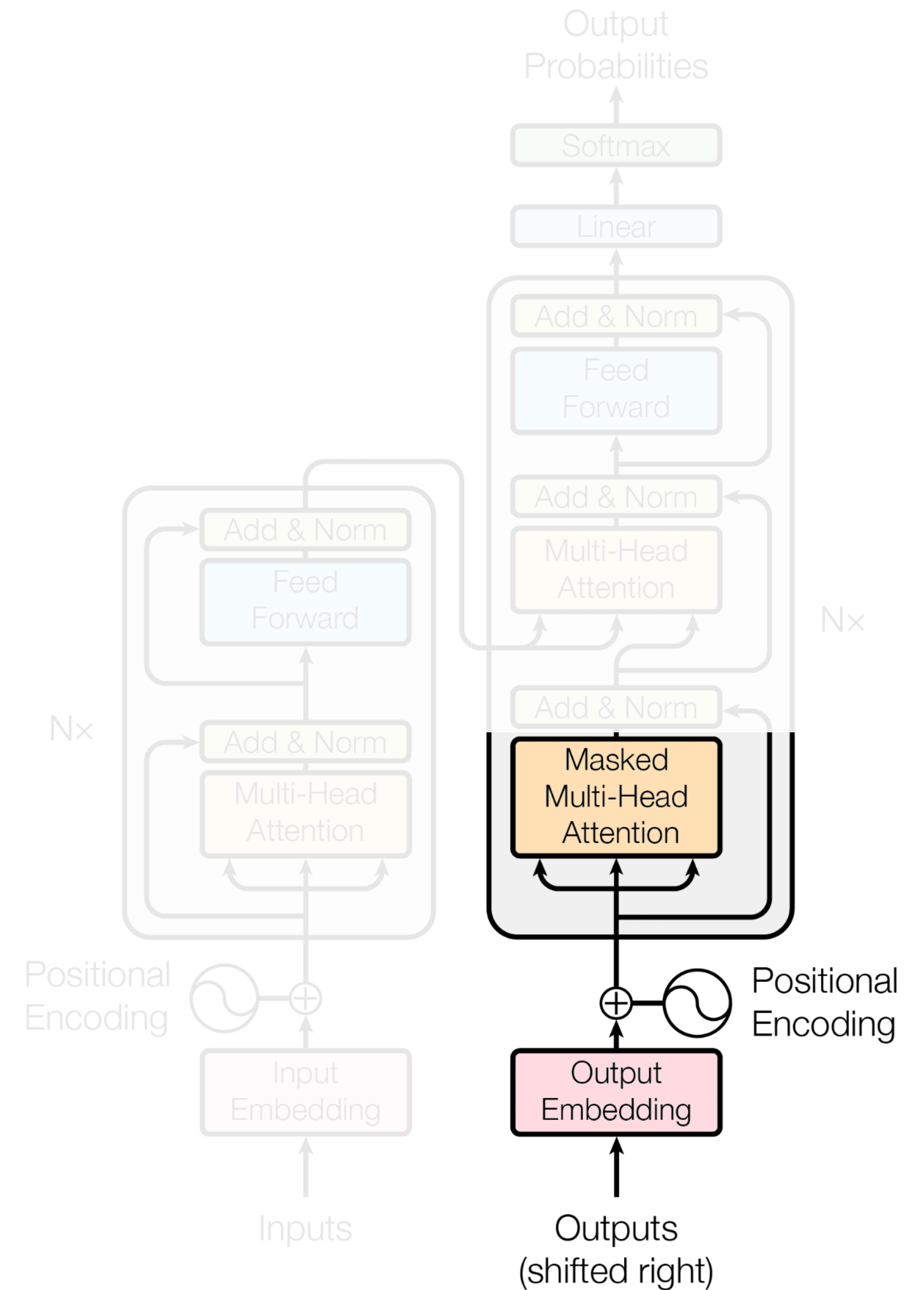
Decoder нужен для генерации
ВЫХОДНОГО ТЕКСТА

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



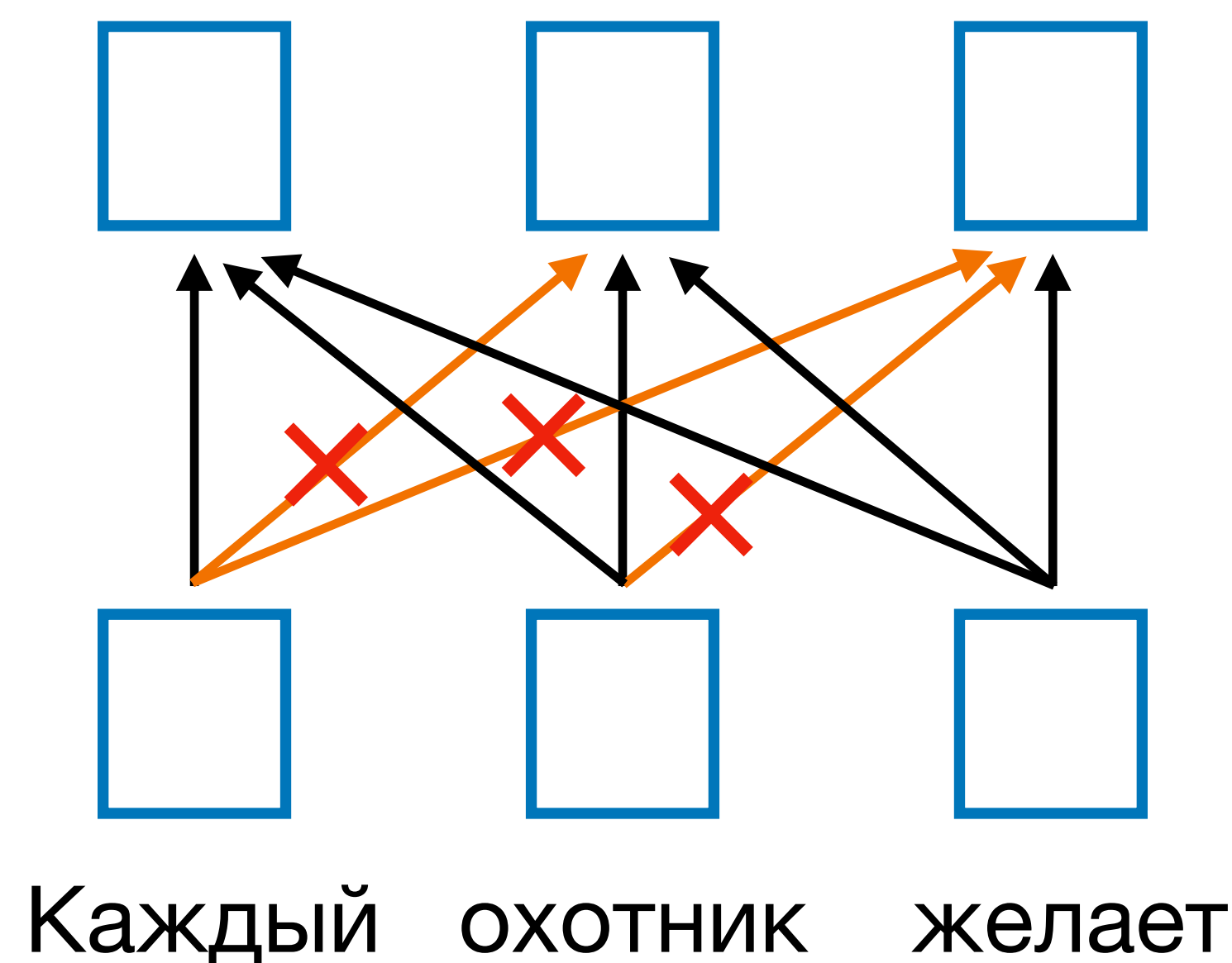
Decoder

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



Masked Self-attention

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



Masked Self-attention

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

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

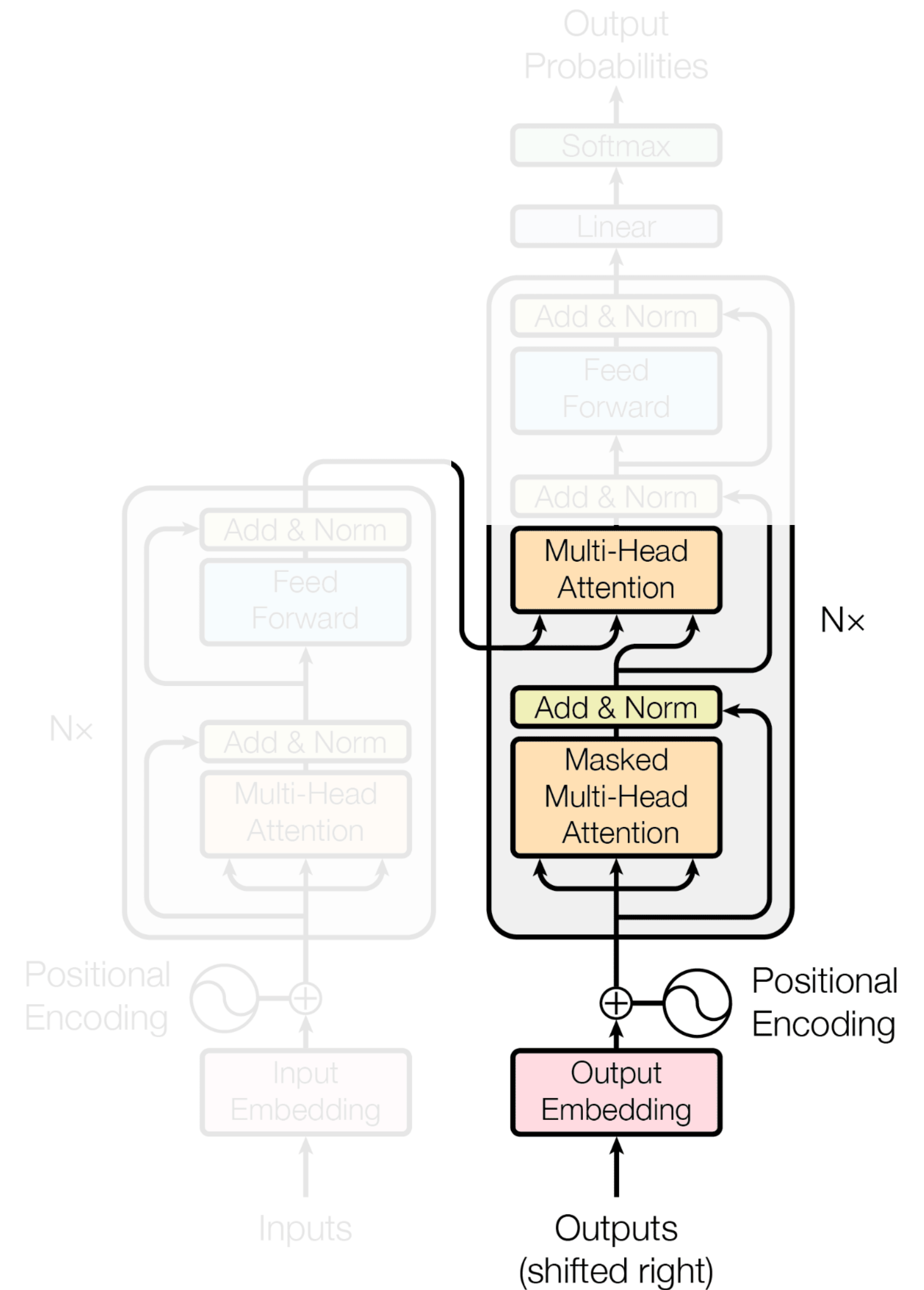
$$M_{ij} = \begin{cases} 0, & i \leq j \\ -\infty, & i > j \end{cases}$$

$M =$

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

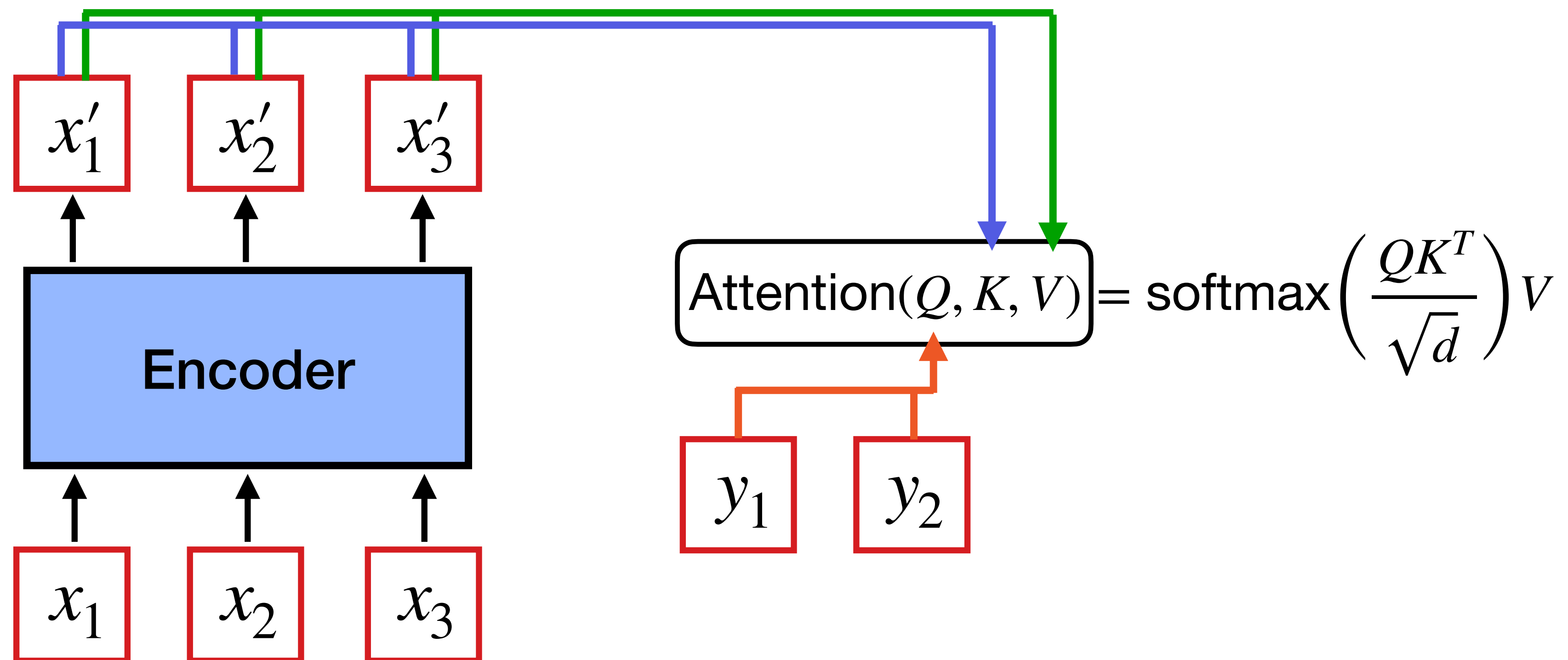
Decoder

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



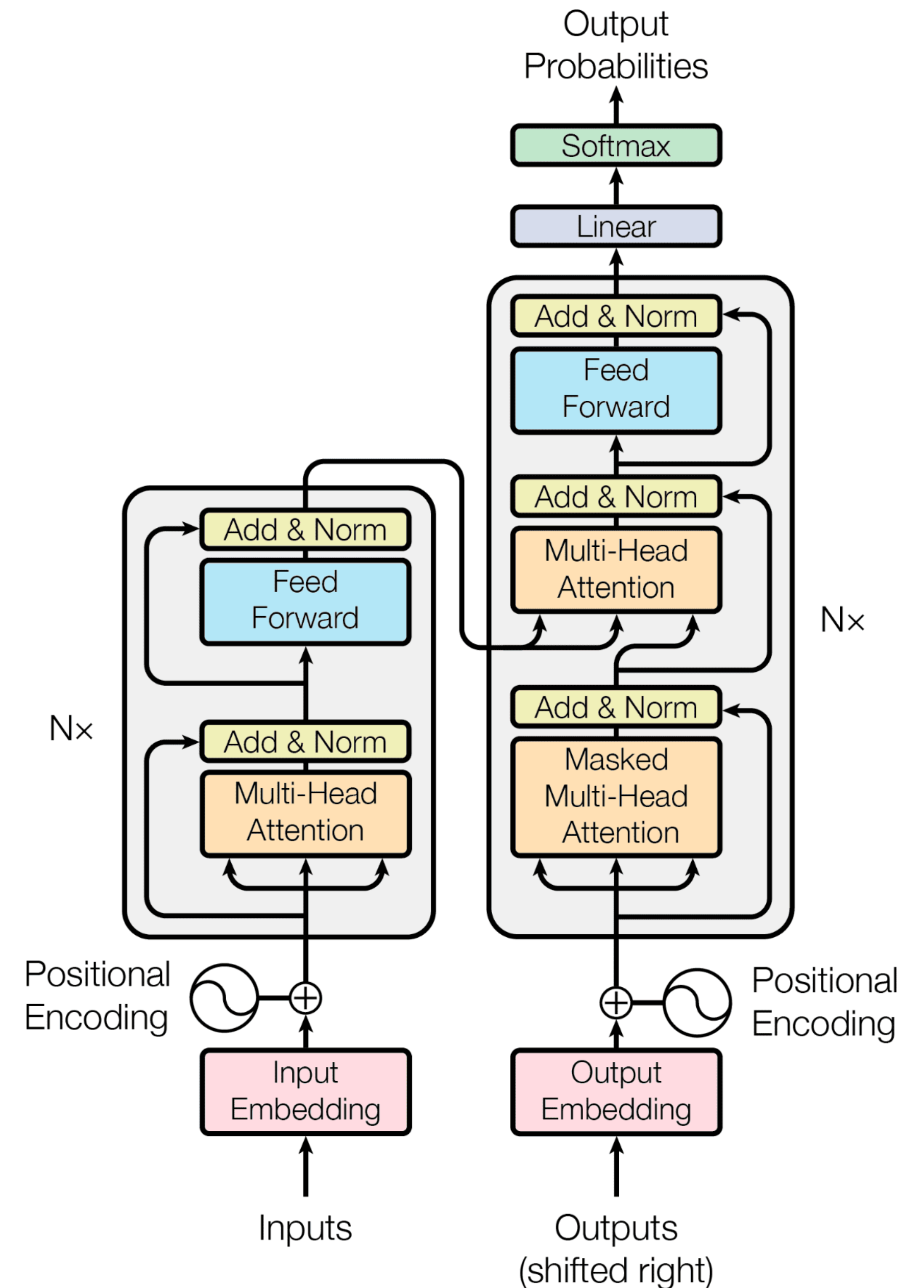
Cross Attention

- 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



Метрики качества

Seq2seq

Метрики качества Seq2seq

- Для оценки качества Seq2seq модели логично сравнить сгенерированный текст с правильным
- Обычно есть несколько подходящих текстов
- Поэтому требовать точное совпадение глупо
- Чаще всего считают совпадение **n-грамм** слов
- Самая известная метрика – **BLEU**

BLEU: BiLingual Evaluation Understudy

Была придумана в **2001** и до сих пор остается самой популярной метрикой

Применяется для

- Машинного перевода
- Диалоговых систем
- Суммаризации текстов

BLEU: Как считать?

BLEU основана на **precision**

\hat{y} – сгенерированный текст

y – правильный текст

Precision

$p_n(\hat{y}, y)$ = доля n-грамм слов в \hat{y} , которые присутствуют в y .

Если правильных текстов несколько, то проверяется присутствие в каждом тексте.

BLEU: Как считать?

BLEU основана на **precision**

\hat{y} – сгенерированный текст

y – правильный текст

Precision

$p_n(\hat{y}, y)$ = доля n-грамм слов в \hat{y} , которые присутствуют в y .

Если правильных текстов несколько, то проверяется присутствие в каждом тексте.

Пример 1:

\hat{y} : рыжий кот прилег на лавку

$y^{(1)}$: кот лежит на лавке

$y^{(2)}$: на лавку легла кошка

$$p_1(\hat{y}, y) = \frac{3}{5}$$

BLEU: Как считать?

BLEU основана на **precision**

\hat{y} – сгенерированный текст

y – правильный текст

Precision

$p_n(\hat{y}, y)$ = доля n-грамм слов в \hat{y} , которые присутствуют в y .

Если правильных текстов несколько, то проверяется присутствие в каждом тексте.

Пример 2:

\hat{y} : рыжий кот прилег на лавку

$y^{(1)}$: кот лежит на лавке

$y^{(2)}$: на лавку легла кошка

$$p_2(\hat{y}, y) = \frac{1}{4}$$

Precision для корпуса текстов

$\hat{Y} = (\hat{y}_1, \dots, \hat{y}_m)$ – корпус сгенерированных текстов

$Y = (y_1, \dots, y_m)$ – корпус правильных текстов

У каждого правильного текста может быть несколько вариантов.

$$y_i = (y_i^{(1)}, \dots, y_i^{(k)})$$

$$p_n(\hat{Y}, Y) = \frac{\sum_{i=1}^m |\text{пересечение } n\text{-грамм слов из } \hat{y}_i \text{ и } y_i|}{\sum_{i=1}^m |\text{n-граммы слов в } \hat{y}_i|}$$

Для максимизации **precision** разумно генерировать очень короткие тексты!

Штраф за краткость (Brevity penalty)

$$BP(\hat{Y}, Y) = \exp\left(-\left[\frac{\sum_{i=1}^m |y_i|}{\sum_{i=1}^m |\hat{y}_i|} - 1\right]_+\right)$$

длины правильных
текстов

длины сгенерированных
текстов

Если сгенерированные тексты суммарно

✓ Длиннее $\rightarrow BP(\hat{Y}, Y) = 1$

✗ Короче $\rightarrow BP(\hat{Y}, Y) = \exp\left(1 - \frac{\sum_{i=1}^m |y_i|}{\sum_{i=1}^m |\hat{y}_i|}\right) < 1$

BLEU: Итоговая метрика

$$\begin{aligned} BLEU(\hat{Y}, Y) &= BP(\hat{Y}, Y) \cdot \exp \left(\sum_{n=1}^N w_n \ln p_n(\hat{Y}, Y) \right) \\ &= BP(\hat{Y}, Y) \cdot \prod_{n=1}^N p_n(\hat{Y}, Y)^{w_n} \end{aligned}$$

- w_n – веса для разных n-грамм. Подбираются вручную.
- Обычно считают **1,2,3,4-граммы**.

$$BLEU(\hat{Y}, Y) \in [0,1]$$

Чем **больше**, тем **лучше**.

BLEU: Недостатки

- Не важен порядок слов и их смысл
- Плохо работает для морфологически богатых языков
- Требуется много эталонных текстов
- Много различных имплементаций. Отличия в
 - Токенизации
 - Методах сглаживания

Улучшения BLEU

ROUGE – F1-мера вместо precision

Использование:

- Суммаризация (почти все датасеты)
- Перефразирование

METEOR – F1-мера, учитывается порядок слов, их формы и синонимы

Использование:

- Машинный перевод

Другие метрики

- Есть масса других метрик, которые исправляют некоторые недостатки предыдущих
- Все они крайне редко используются

Метрики:

- eBLEU
- NIST
- HPPR
- LEPOR
- RIBES

Методы семплирования токенов из распределения

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

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

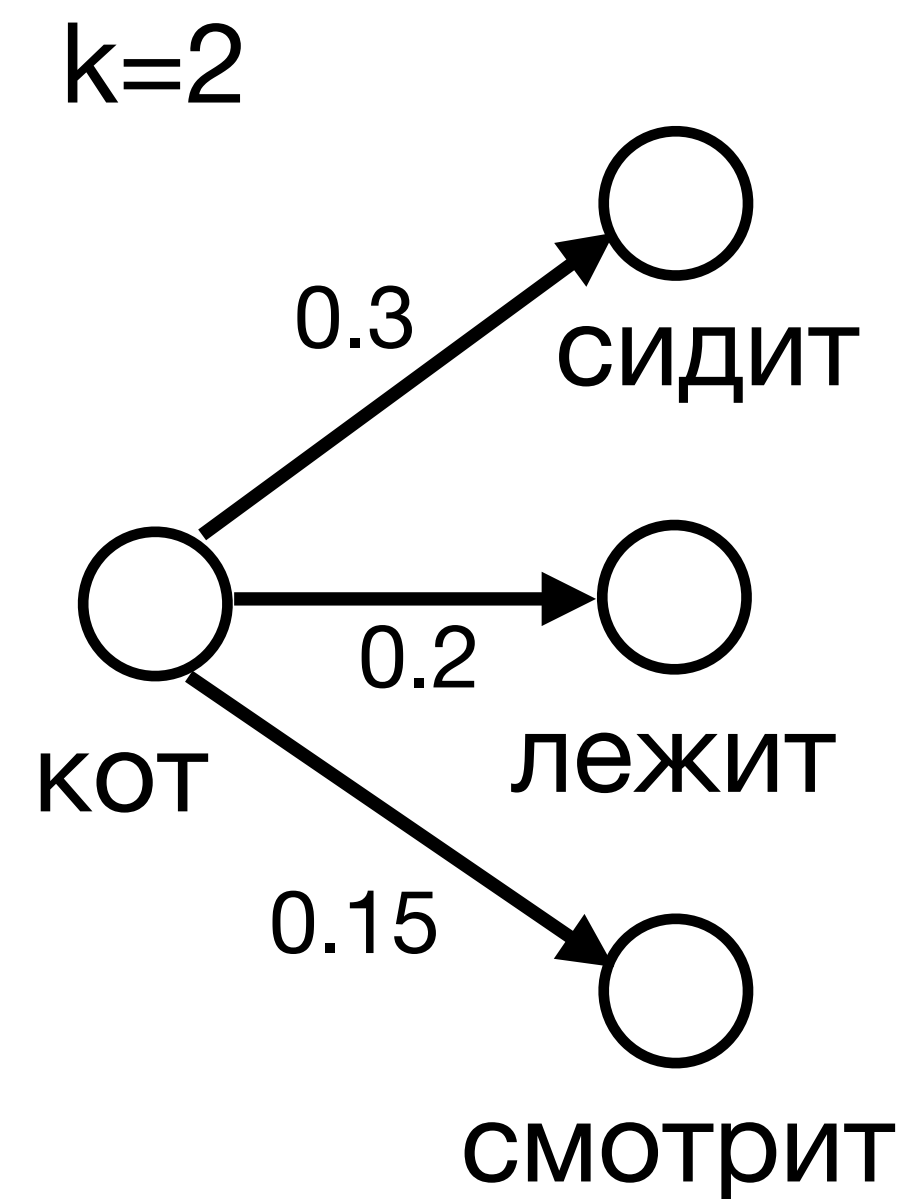
- Попробуем максимизировать вероятность текста еще больше
- При жадном семплировании мы **не учитываем** влияние токена на следующие за ним
- Будем строить предсказания на несколько токенов вперед, а затем выбирать токен с наибольшей совокупной вероятностью

Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

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

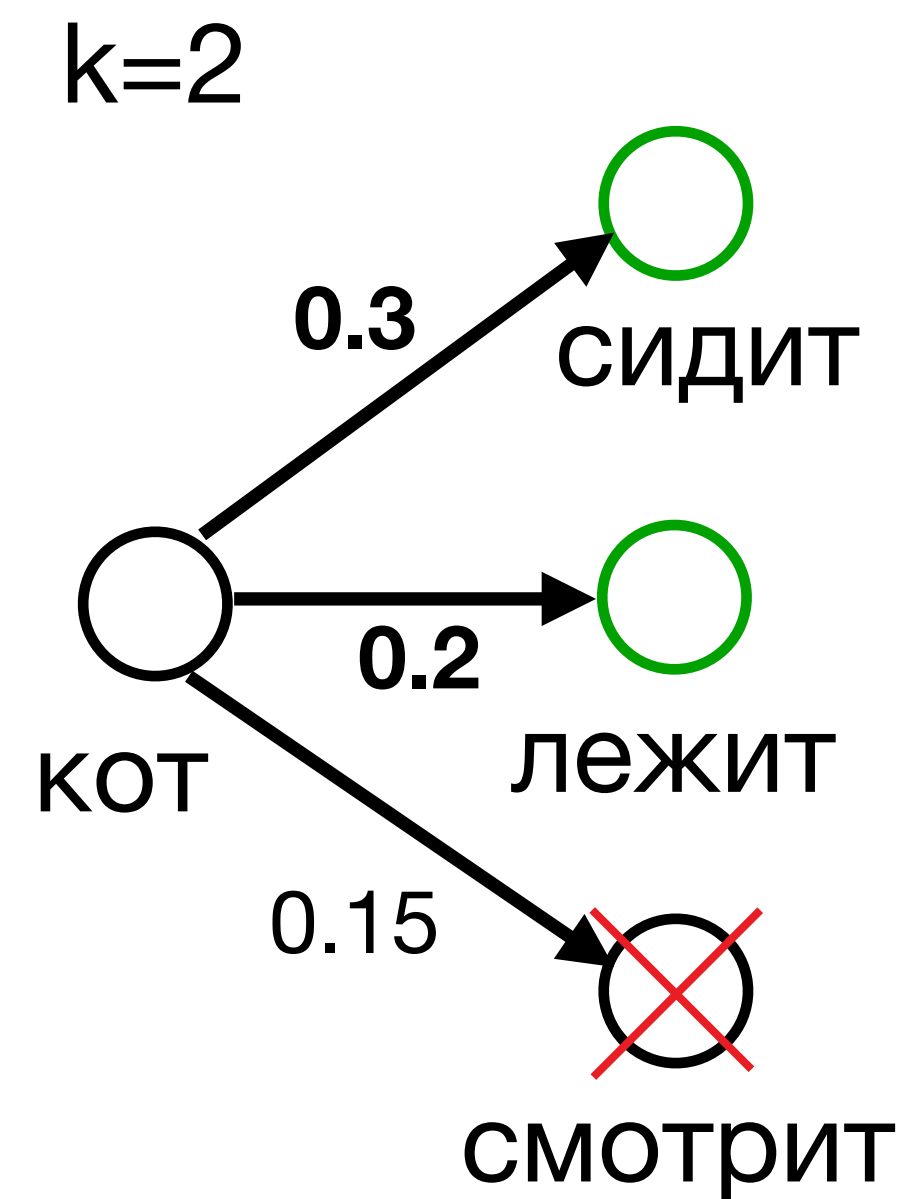


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

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

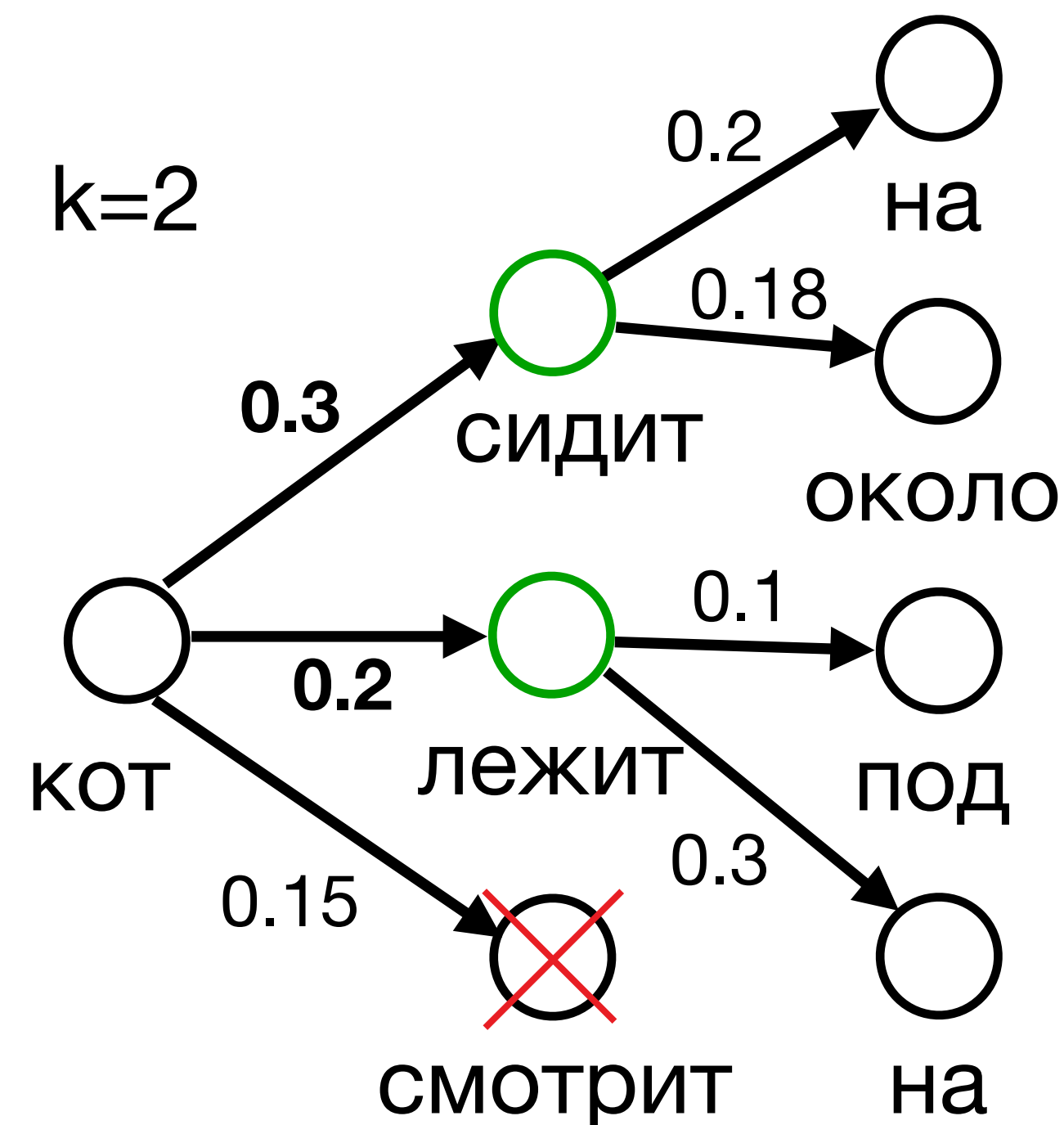


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

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

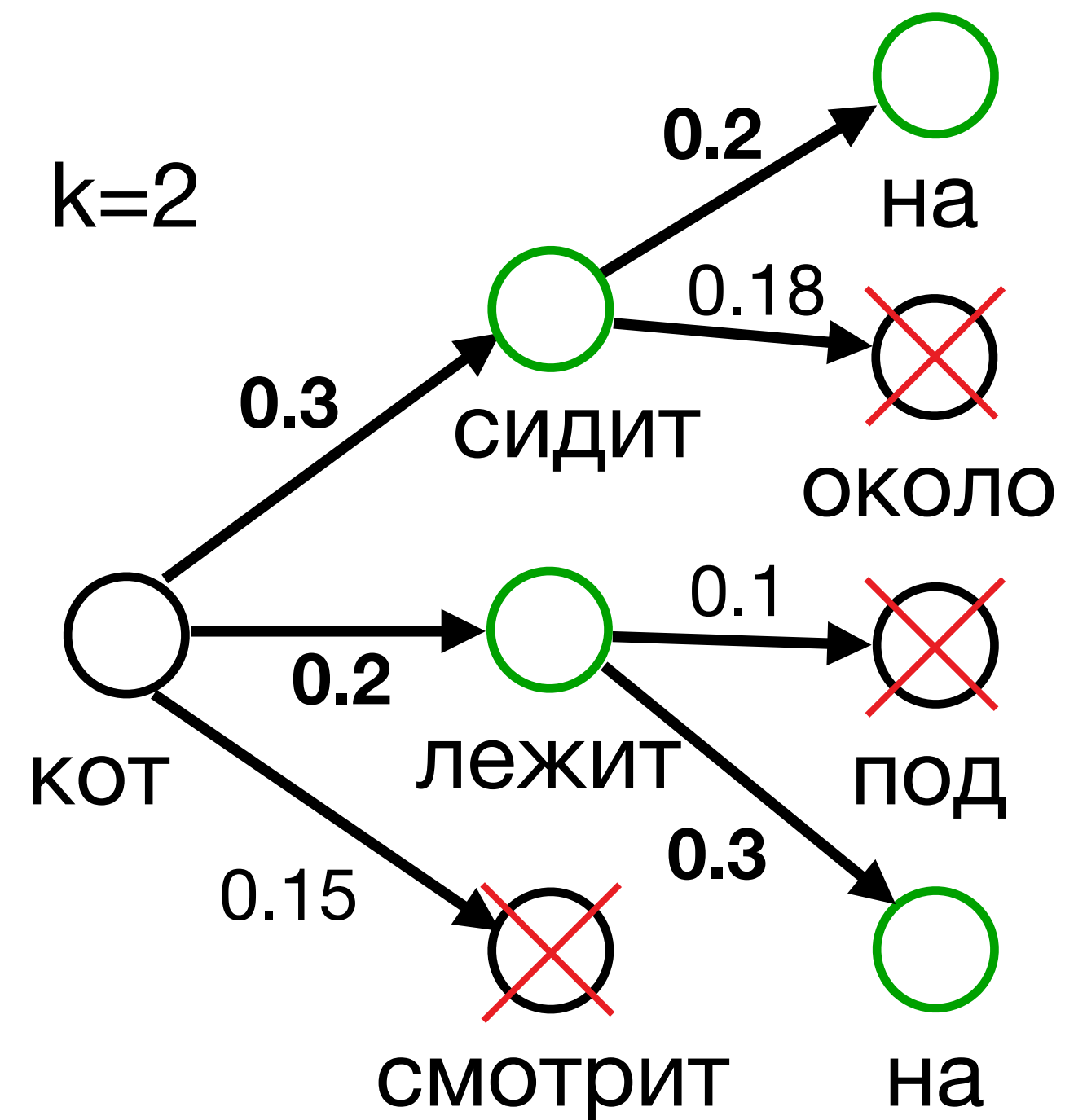


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

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

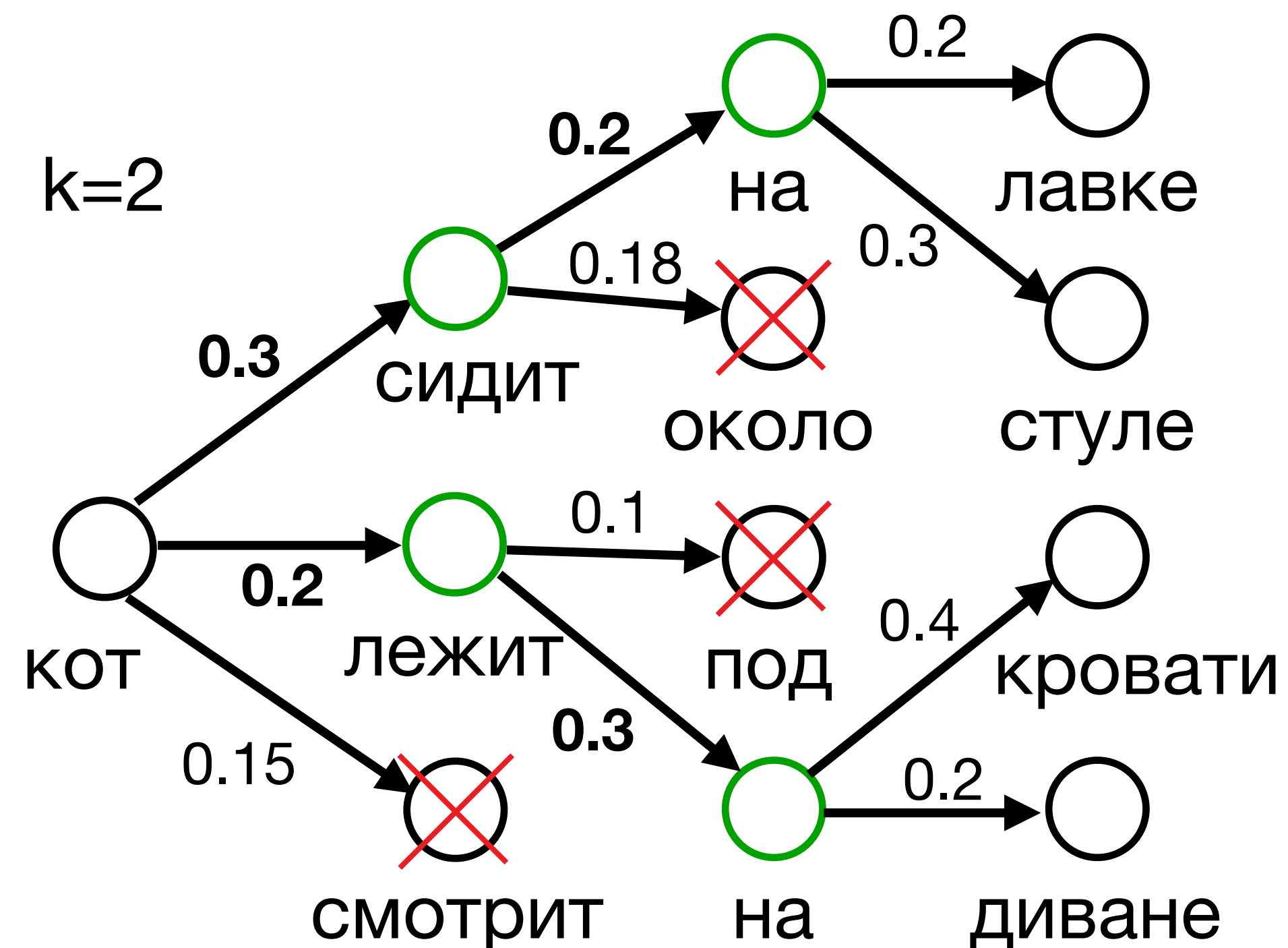


Beam Search

Лучевой поиск

Идея: Поддерживаем k наиболее вероятных траекторий

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

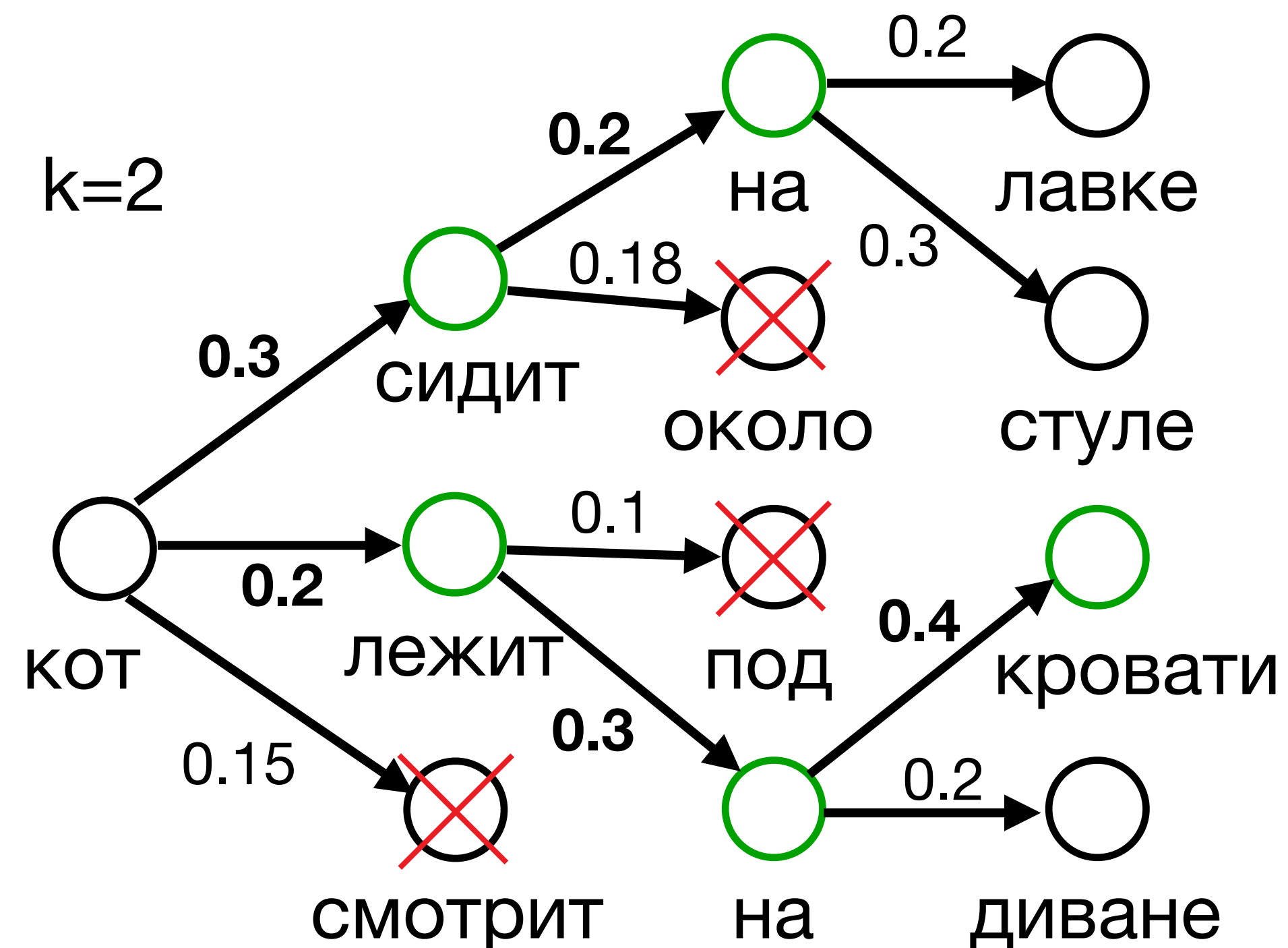


Beam Search

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

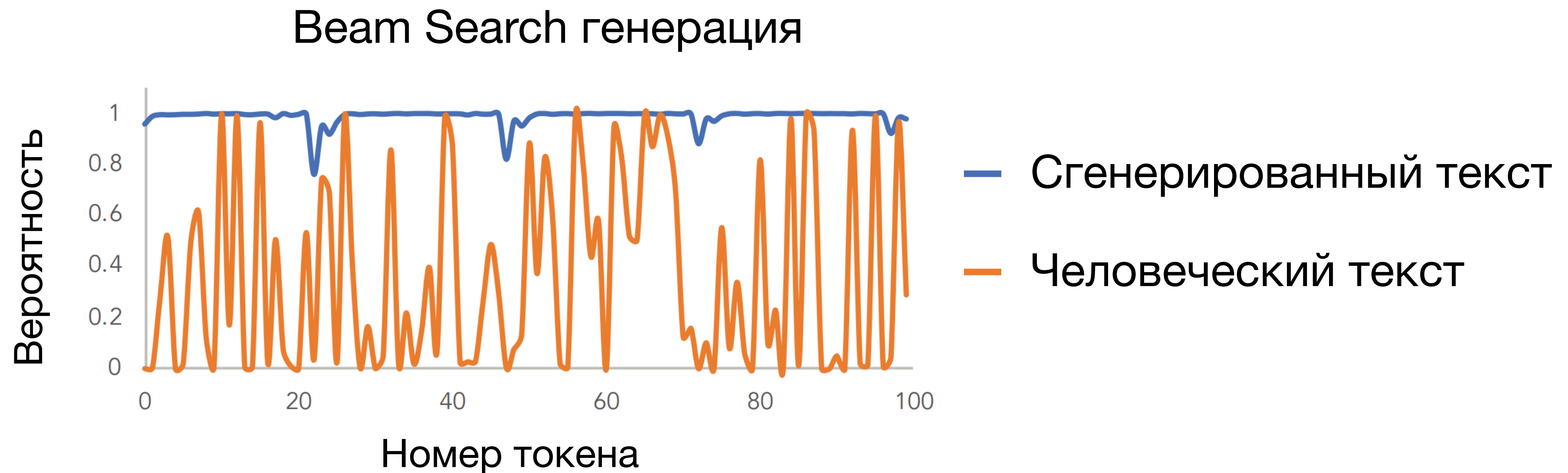
=> выбираем "лежит"

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

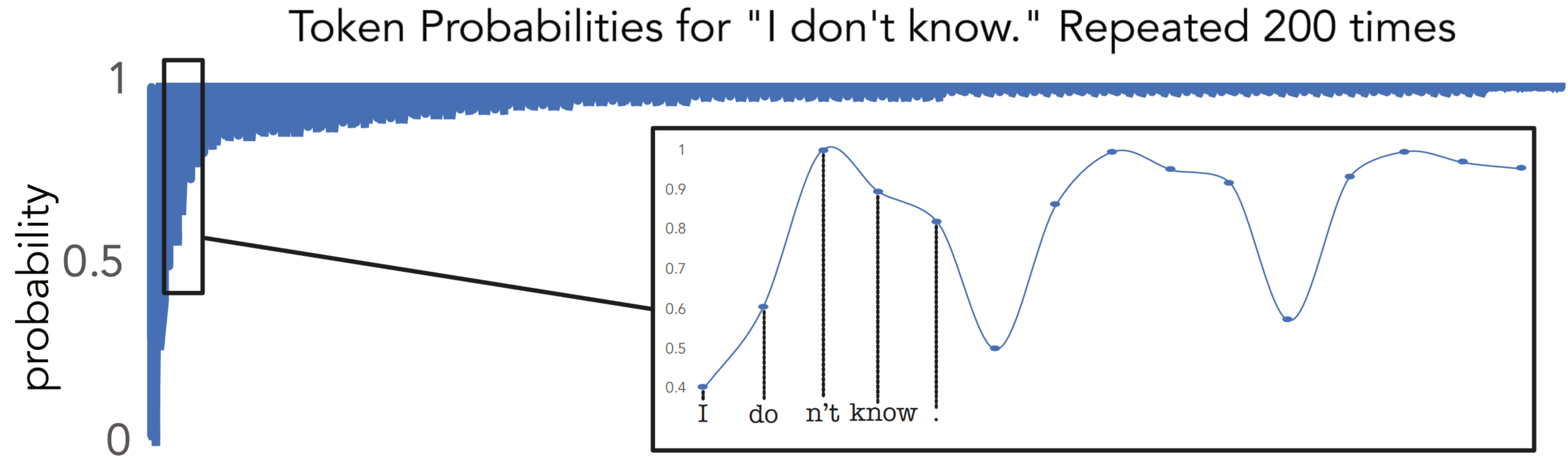
Безусловная генерация

Оба метода уменьшают разнообразие безусловной генерации!



Безусловная генерация

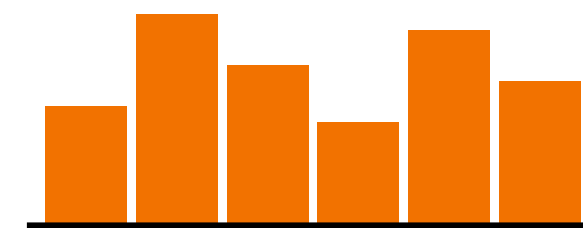
Оба метода поощряют повторения!



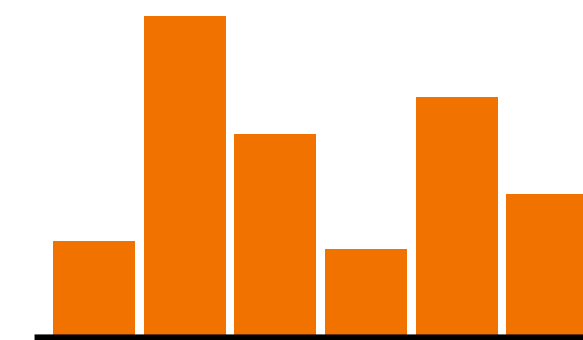
Семплирование с температурой

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

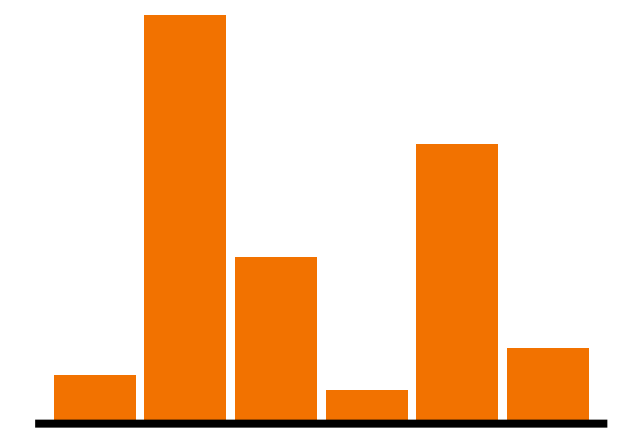
$$p_{\tau}(y | y_{<t}) = \frac{\exp \frac{p(y | y_{<t})}{\tau}}{\sum_{w \in V} \exp \frac{p(w | y_{<t})}{\tau}}$$



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



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

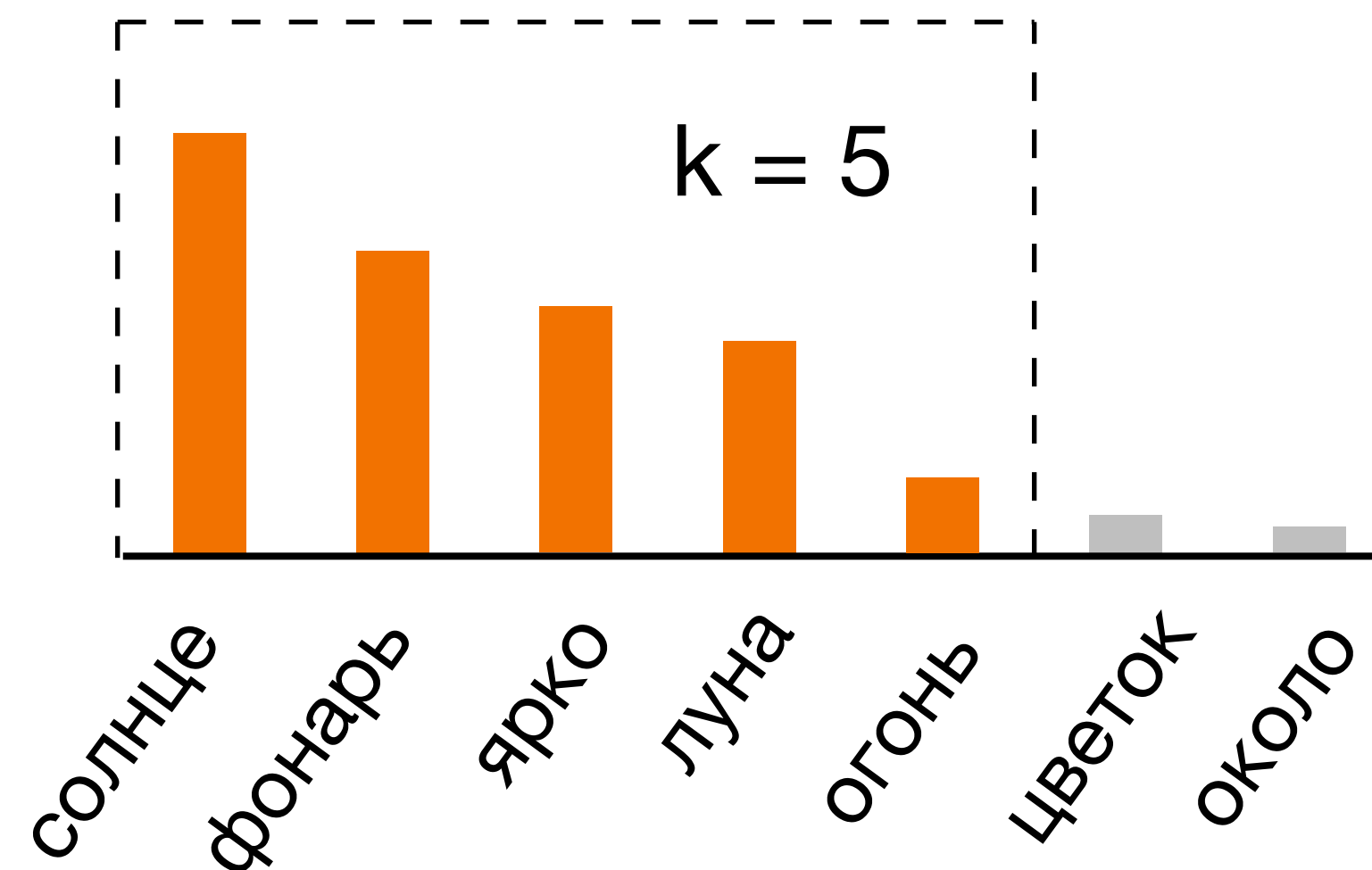


$p_{0.5}(y_t | y_{<t})$

Top-k семплирование

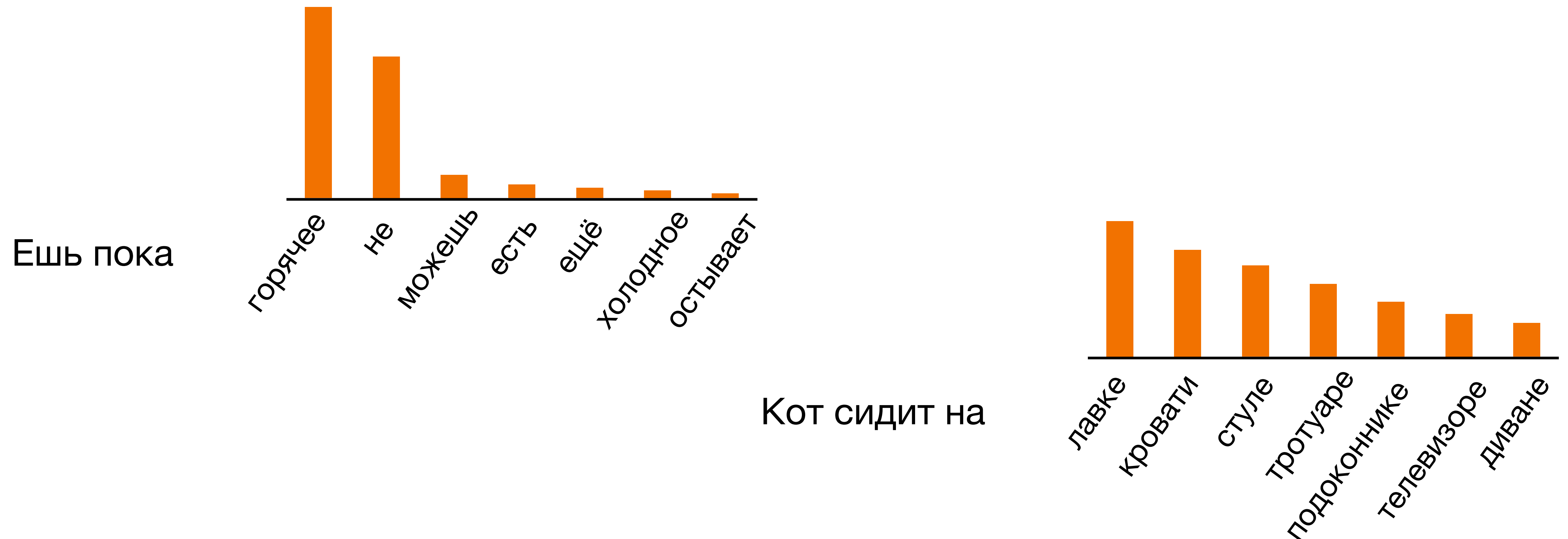
- Даже с температурой остается ненулевая генерация выдать случайный токен
- Оставим только k самых вероятных токенов и будем семплировать из них

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