

Основы глубинного обучения

Лекция 11

Рекуррентные модели

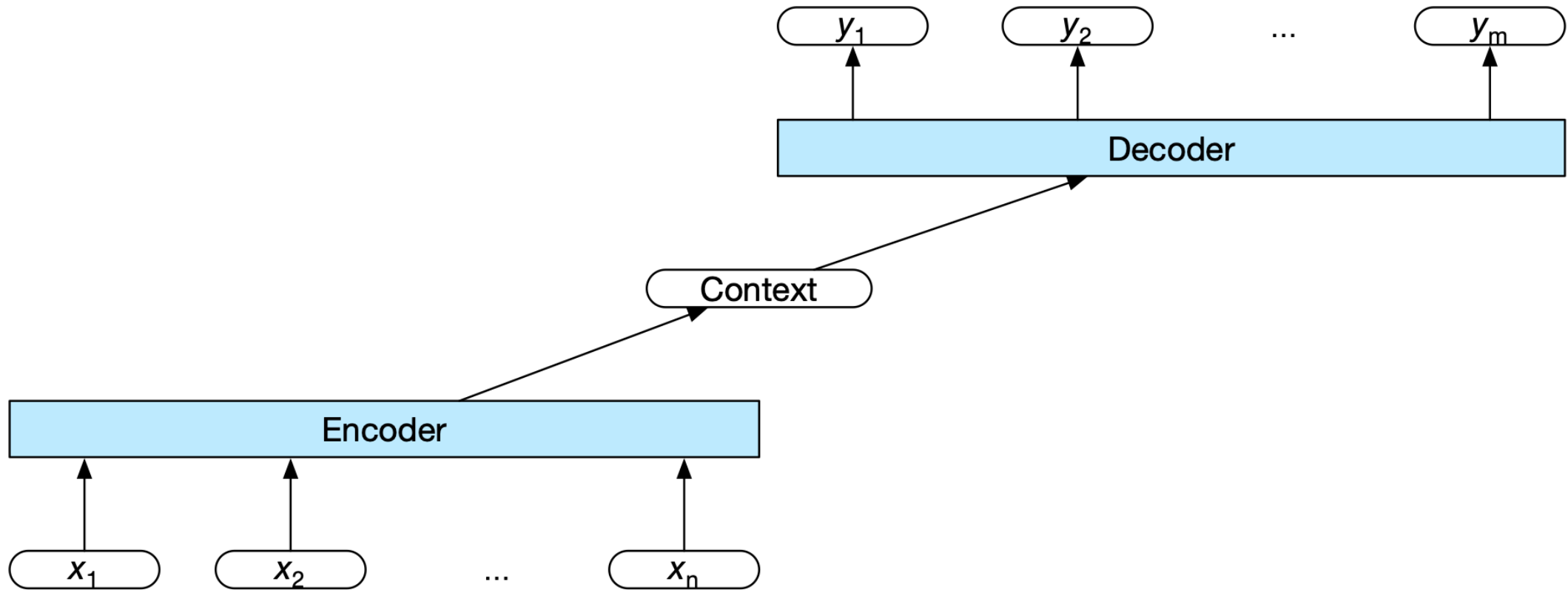
Евгений Соколов

esokolov@hse.ru

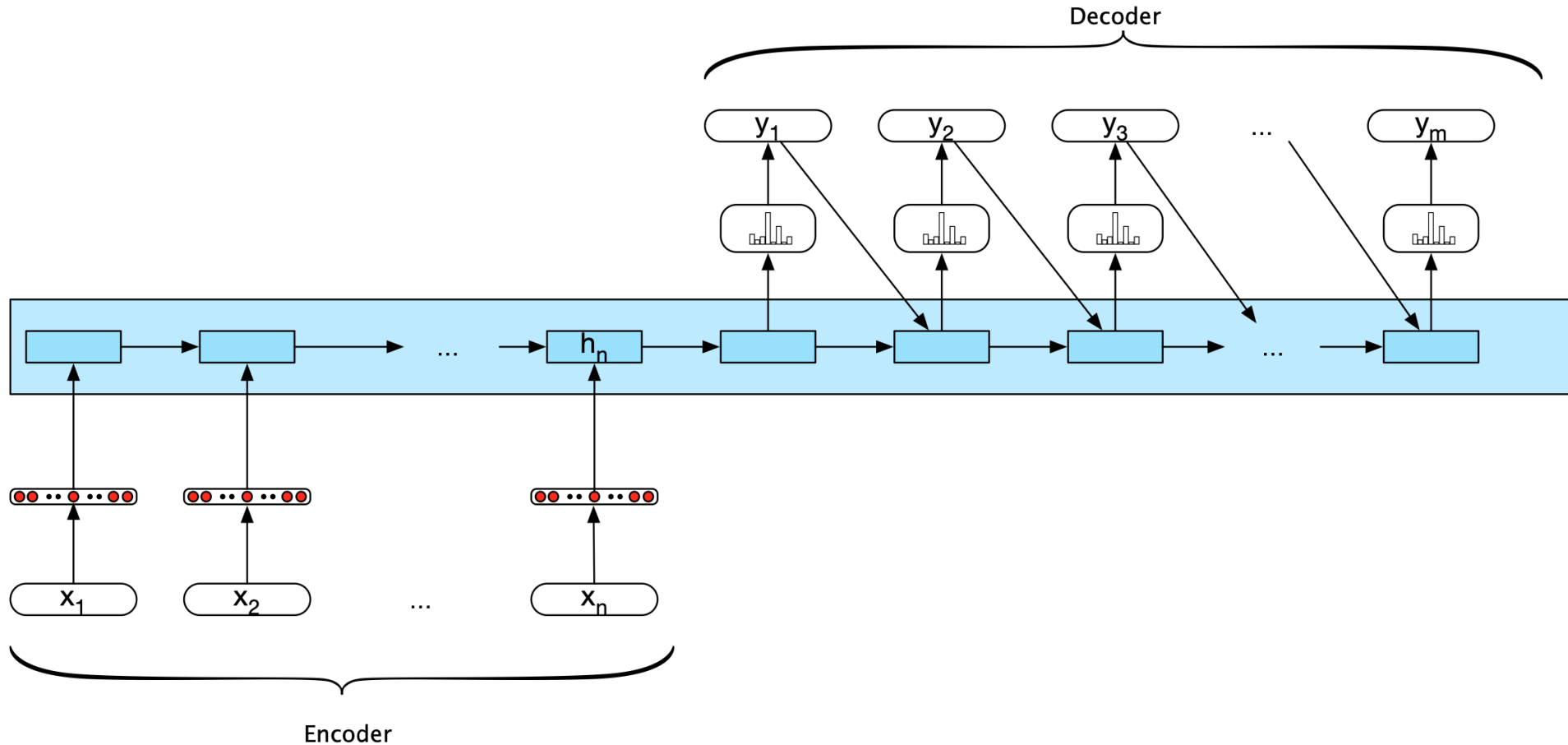
НИУ ВШЭ, 2022

Seq2seq

Seq2seq Machine Translation



Seq2seq Machine Translation



Проблемы seq2seq-архитектуры

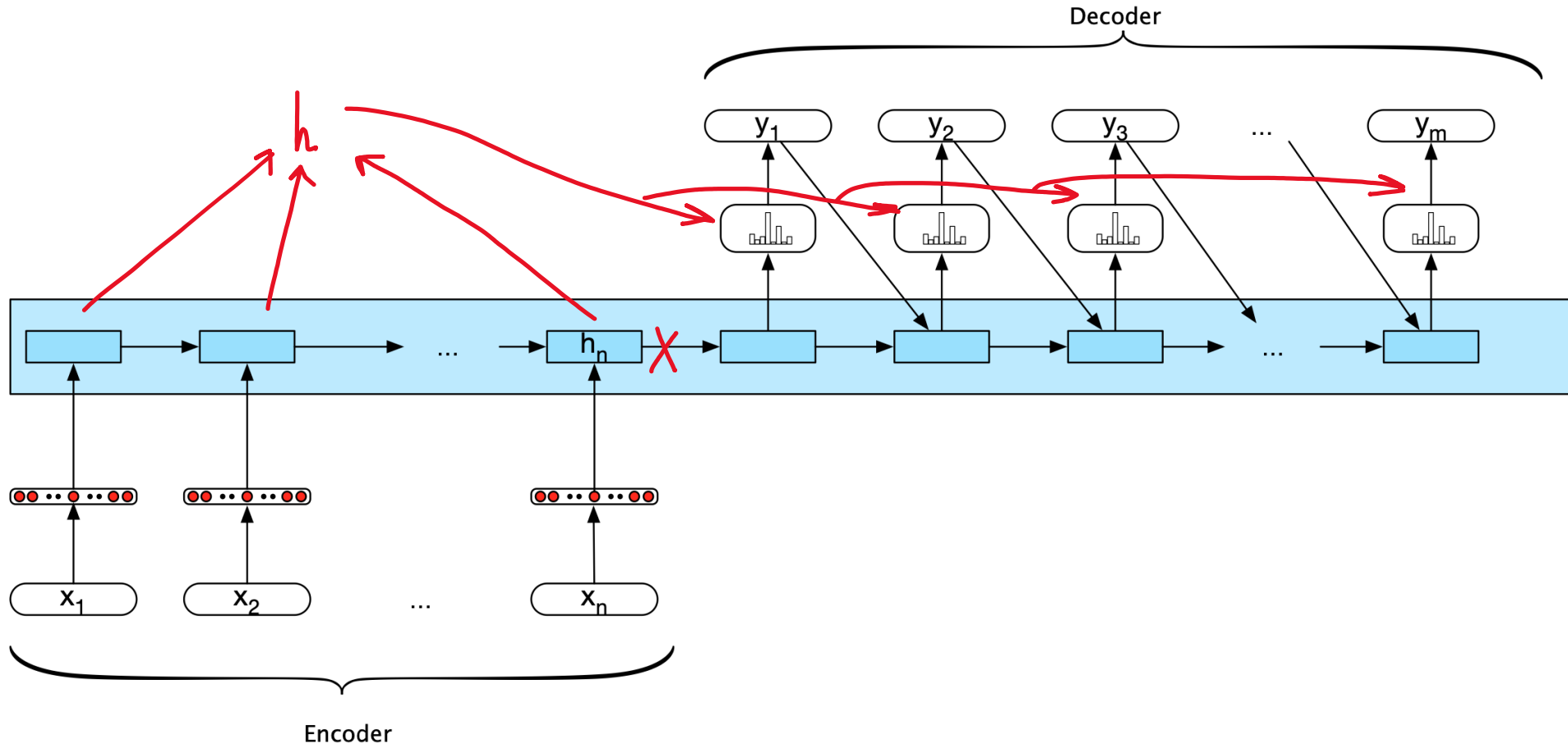
- Нужно сжать весь текст в один вектор
 - Теряется информация о первых словах
 - Декодер тоже может терять информацию по мере генерации последовательности
-
- Можно использовать BiLSTM, но тогда будет теряться информация о словах в середине
 - И непонятно, как им декодировать

Механизм внимания

Механизм внимания

- Во время генерации каждого слова будем смотреть на **всю** входную последовательность
- Усредним все скрытые состояния?

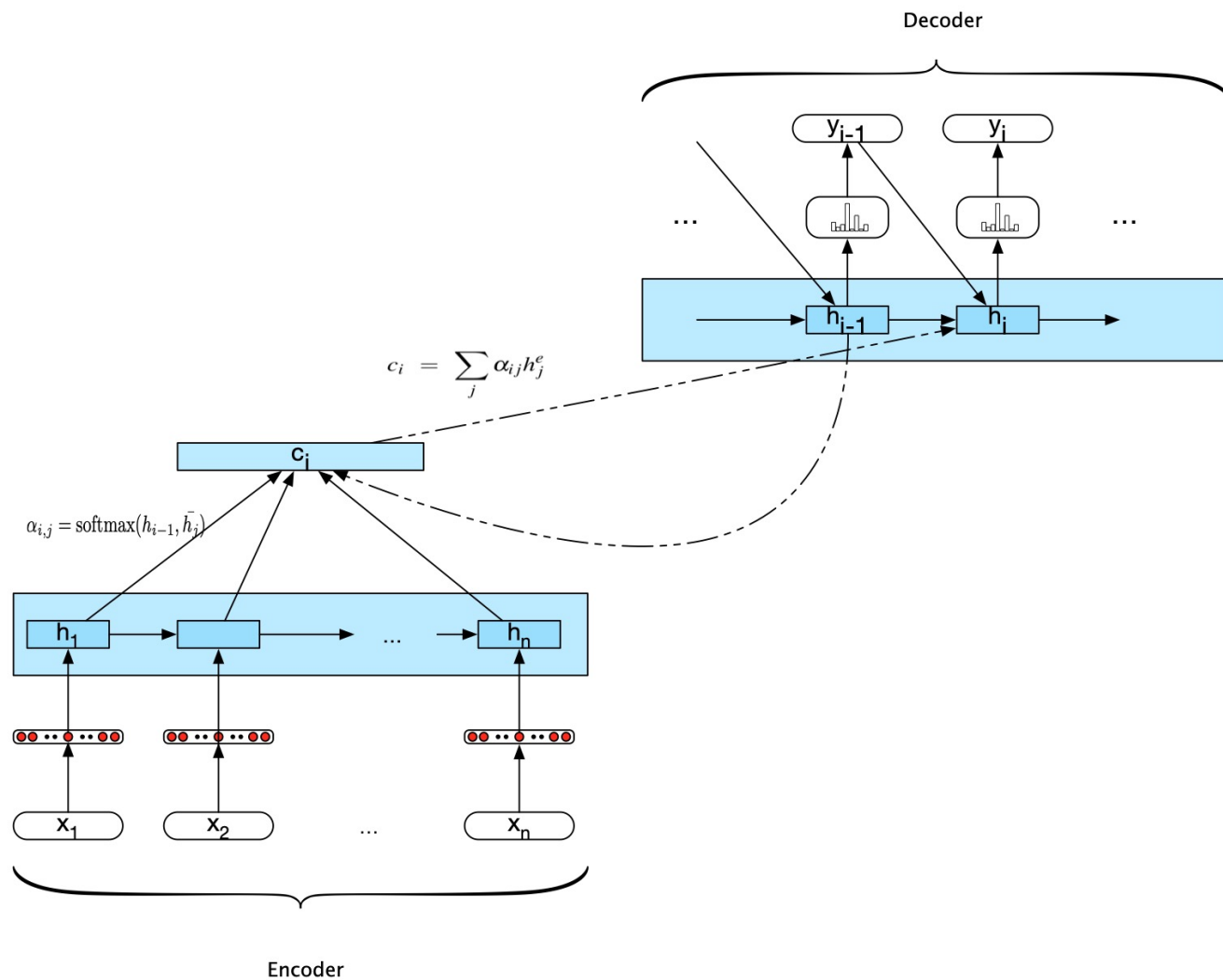
Seq2seq Machine Translation



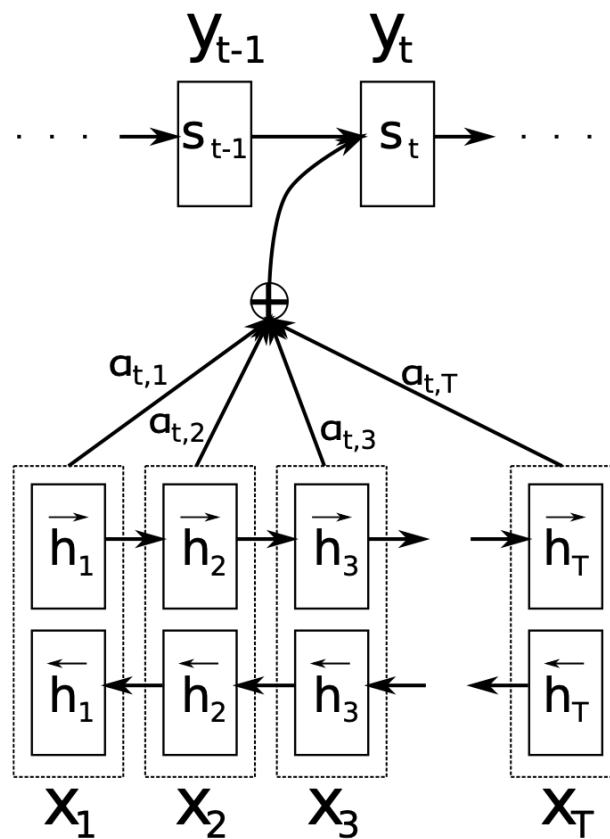
Механизм внимания

- Во время генерации каждого слова будем смотреть на **ВСЮ** входную последовательность
- Усредним все скрытые состояния?
- Плохо, не сможем сделать акцент на одном из входных слов

Механизм внимания



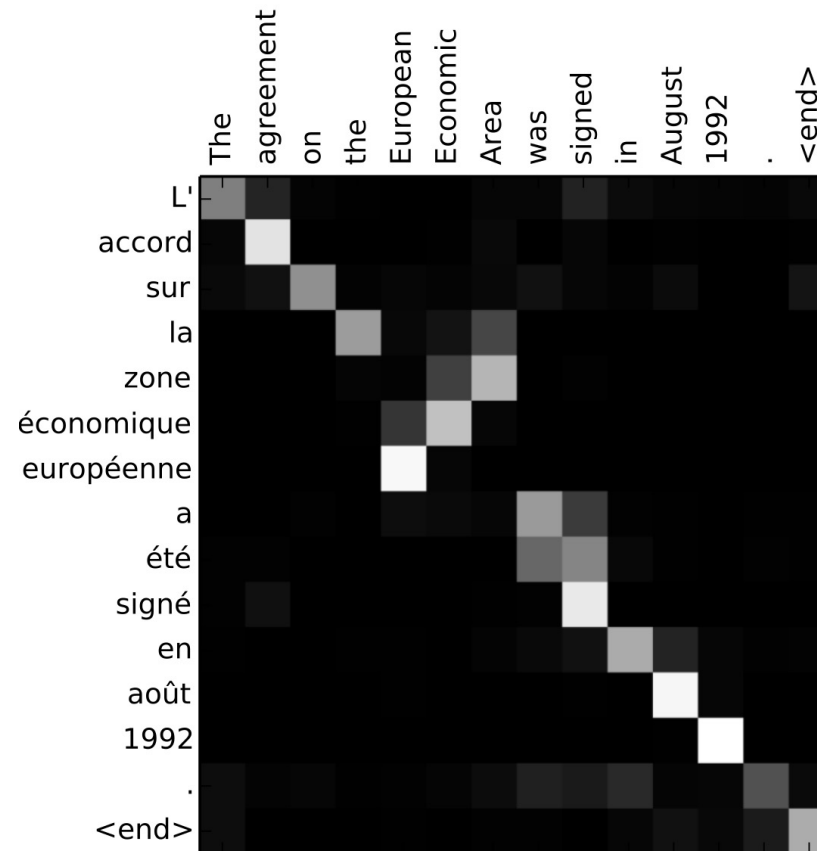
Механизм внимания



Механизм внимания

- Скрытое состояние декодировщика: $h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c_t)$
- Релевантность j -го входного слова t -му выходному слову: $s(h_j^e, h_{t-1}^d)$; обычно это полносвязная нейросеть
- Распределение на входных словах: $\alpha_{jt} = \text{softmax}(s(h_j^e, h_{t-1}^d))$
(берётся по всем j)
- $c_t = \sum_j \alpha_{jt} h_j^e$

Механизм внимания



Трансформеры

Что почитать

- <https://jalammar.github.io/illustrated-transformer/>
- <https://arxiv.org/abs/1706.03762>

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

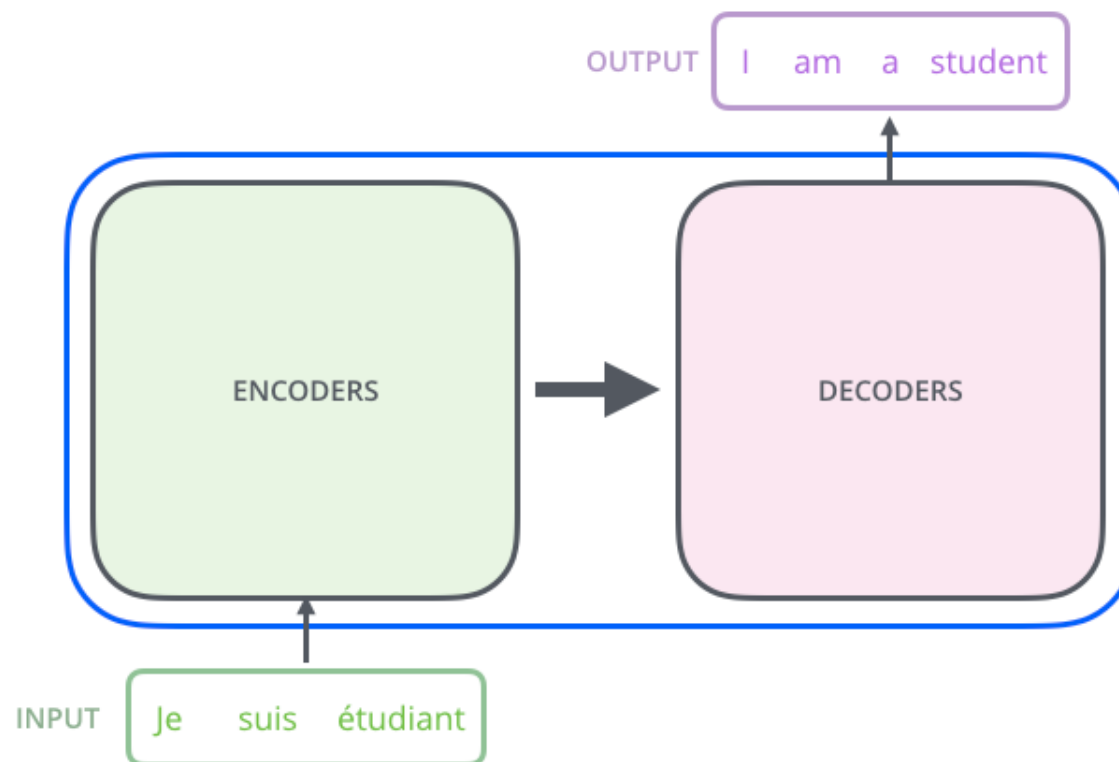
Llion Jones*
Google Research
llion@google.com

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

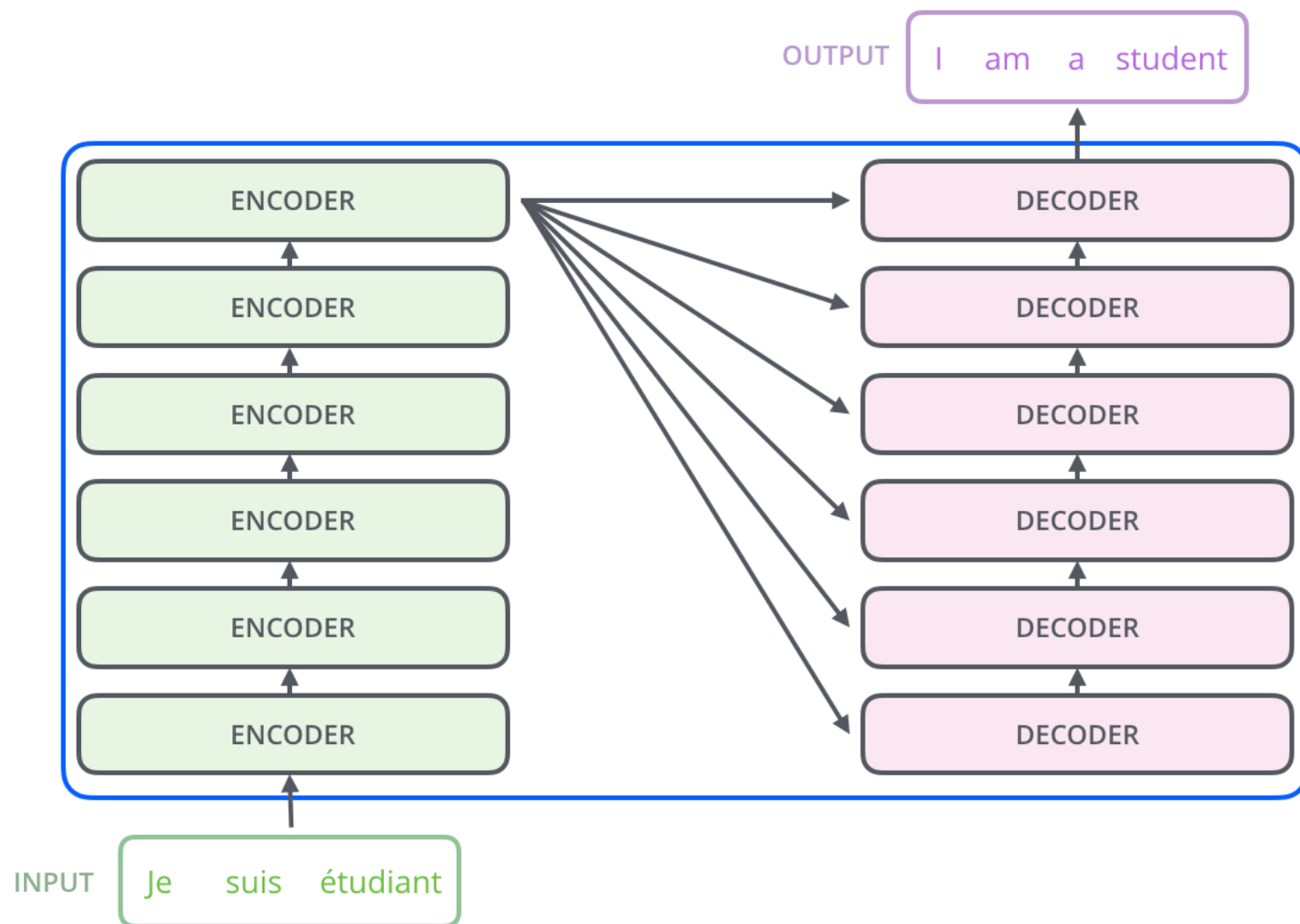
Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

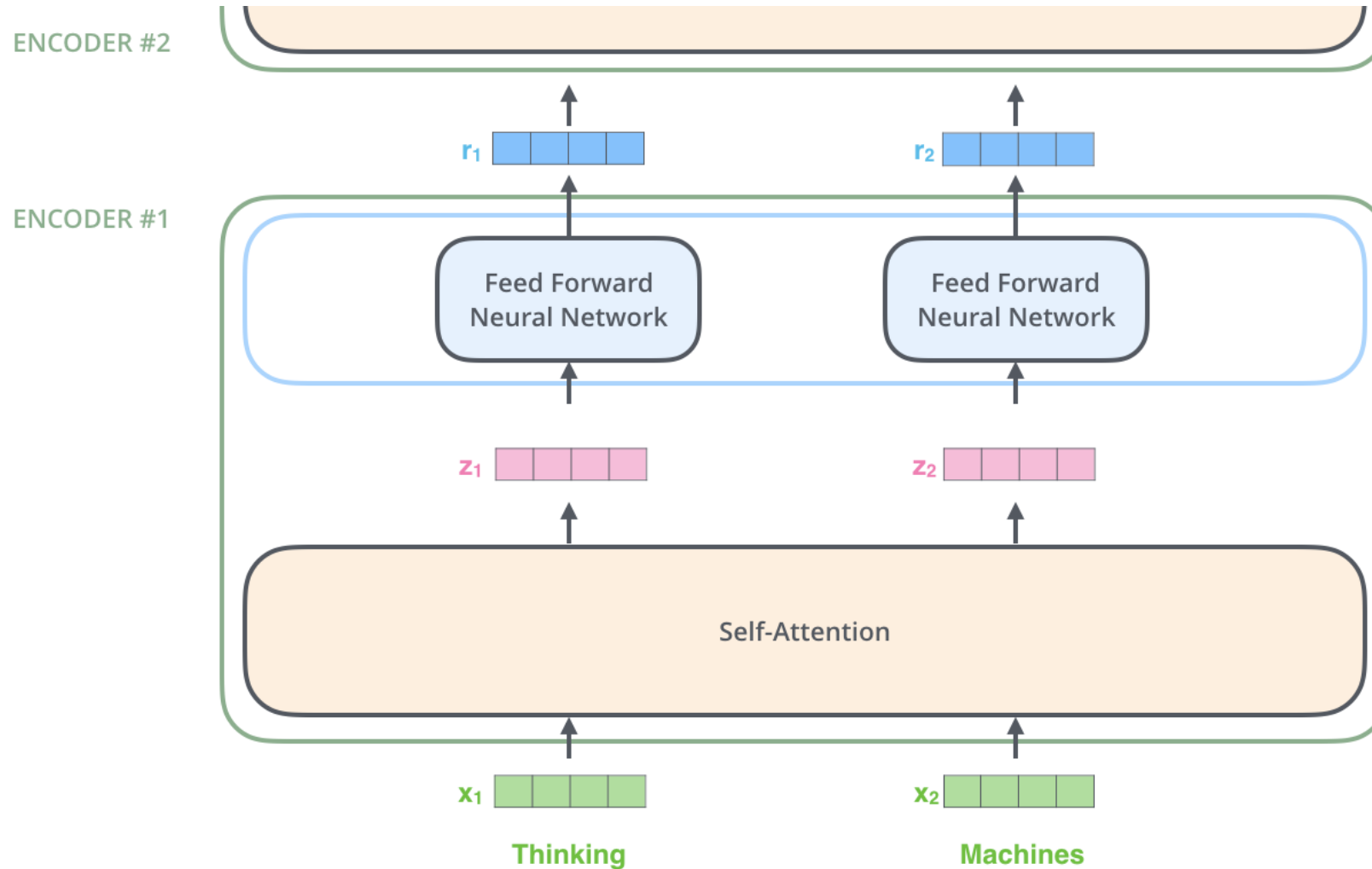
Трансформер



Трансформер



Кодировщик в трансформере



Self-attention

- Будем для каждого слова x_j обучать три вектора:
 - Запрос (query) $q_j = W_Q x_j$
 - Ключ (key) $k_j = W_K x_j$
 - Значение (value) $v_j = W_V x_j$
- «Важность» слова x_i для слова x_j : $\langle q_j, k_i \rangle$

Self-attention

- Вклад слова x_i в новое представление слова x_j :

$$w_{ij} = \frac{\exp\left(\frac{\langle q_j, k_i \rangle}{\sqrt{d}}\right)}{\sum_{p=1}^n \exp\left(\frac{\langle q_j, k_p \rangle}{\sqrt{d}}\right)}$$

- d — размерность векторов q_j и k_i
- n — число слов во входной последовательности

Self-attention

Новое представление слова x_j :

$$z_j = \sum_{p=1}^n w_{pj} v_p$$

Self-attention

То же самое, но в матричном виде:

$$Z = \text{softmax}\left(\frac{1}{\sqrt{d}} Q K^T\right) V$$

The diagram illustrates the self-attention mechanism using matrix dimensions. It shows the following components:

- Q** (Query matrix): A 2x3 purple grid.
- K^T** (Key matrix, transposed): A 3x2 orange grid.
- V** (Value matrix): A 2x3 blue grid.
- Z** (Result matrix): A 2x3 pink grid.

The operation is represented as:

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

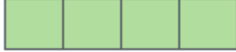
Where d_k is the dimension of the key matrix (2).

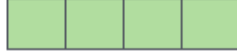
Input

Thinking

Machines

Embedding

x_1 

x_2 

Queries

q_1 

q_2 

Keys

k_1 

k_2 

Values

v_1 

v_2 

Score

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

Divide by 8 ($\sqrt{d_k}$)

14

12

Softmax

0.88

0.12

Softmax

X
Value

v_1 

v_2 

Sum

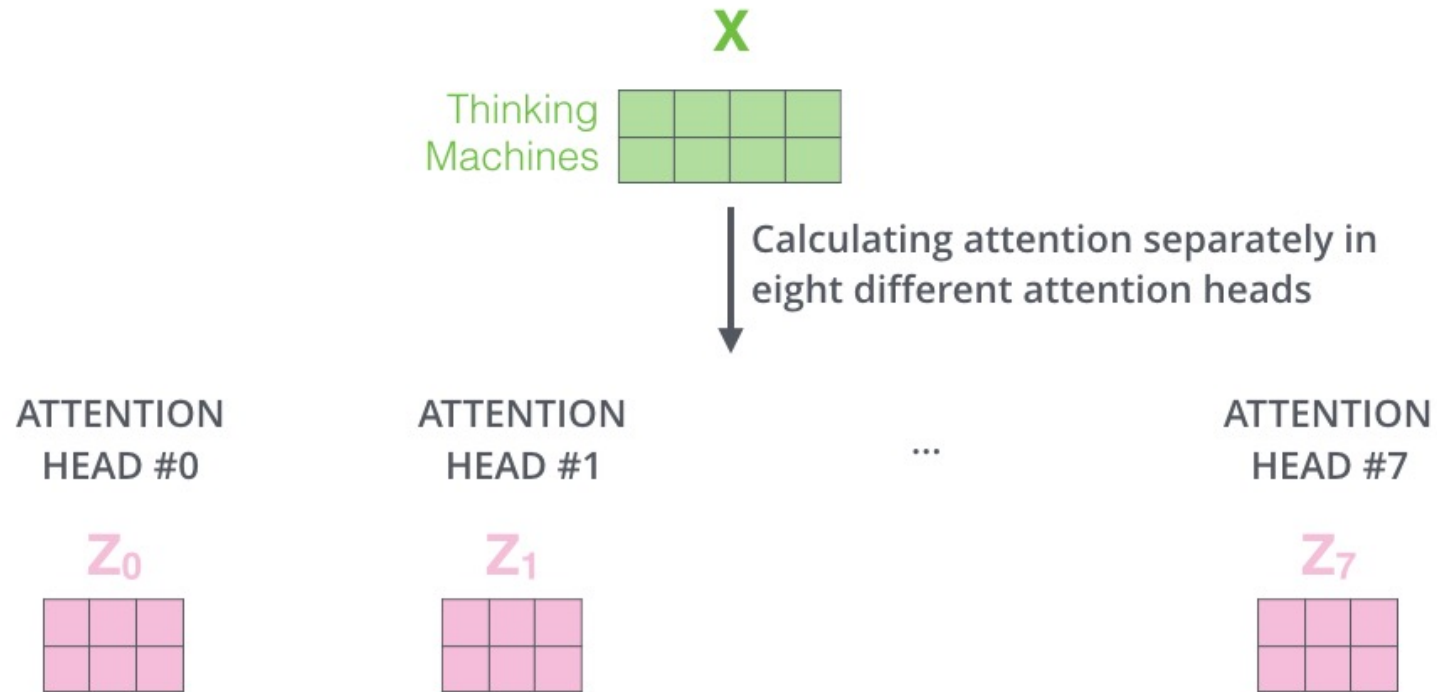
z_1 

z_2 

Multi-headed Self-attention

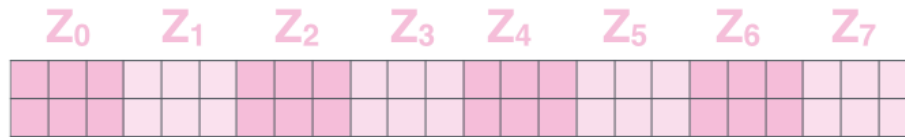
- Механизм работает хорошо
- Если напастать его, станет ещё лучше, наверное

Multi-headed Self-attention



Multi-headed Self-attention

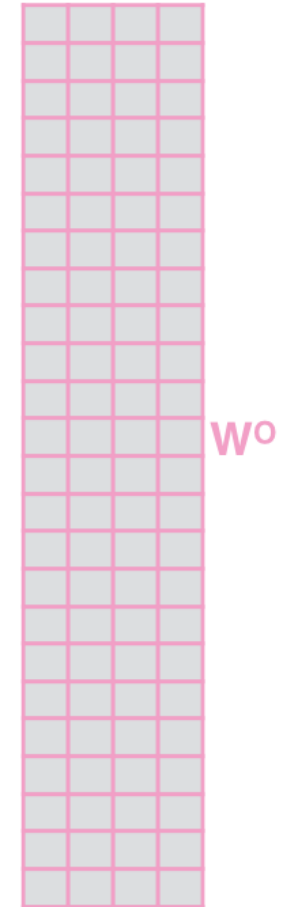
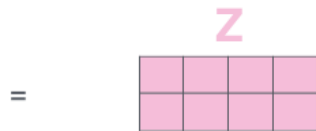
1) Concatenate all the attention heads



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

\times

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

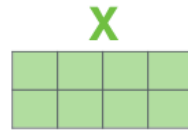


Multi-headed Self-attention

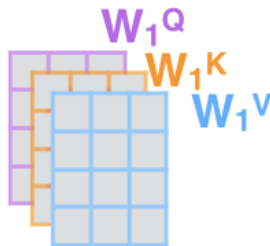
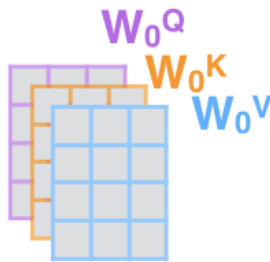
1) This is our
input sentence*

Thinking
Machines

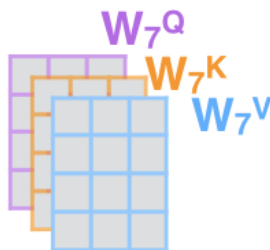
2) We embed
each word*



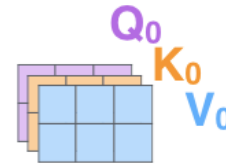
3) Split into 8 heads.
We multiply X or
 R with weight matrices



...



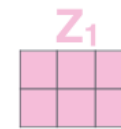
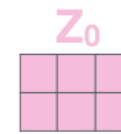
4) Calculate attention
using the resulting
 $Q/K/V$ matrices



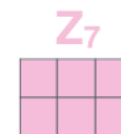
...



5) Concatenate the resulting Z matrices,
then multiply with weight matrix W^O to
produce the output of the layer



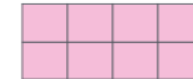
...



W^O

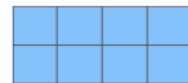


Z



* In all encoders other than #0,
we don't need embedding.
We start directly with the output
of the encoder right below this one

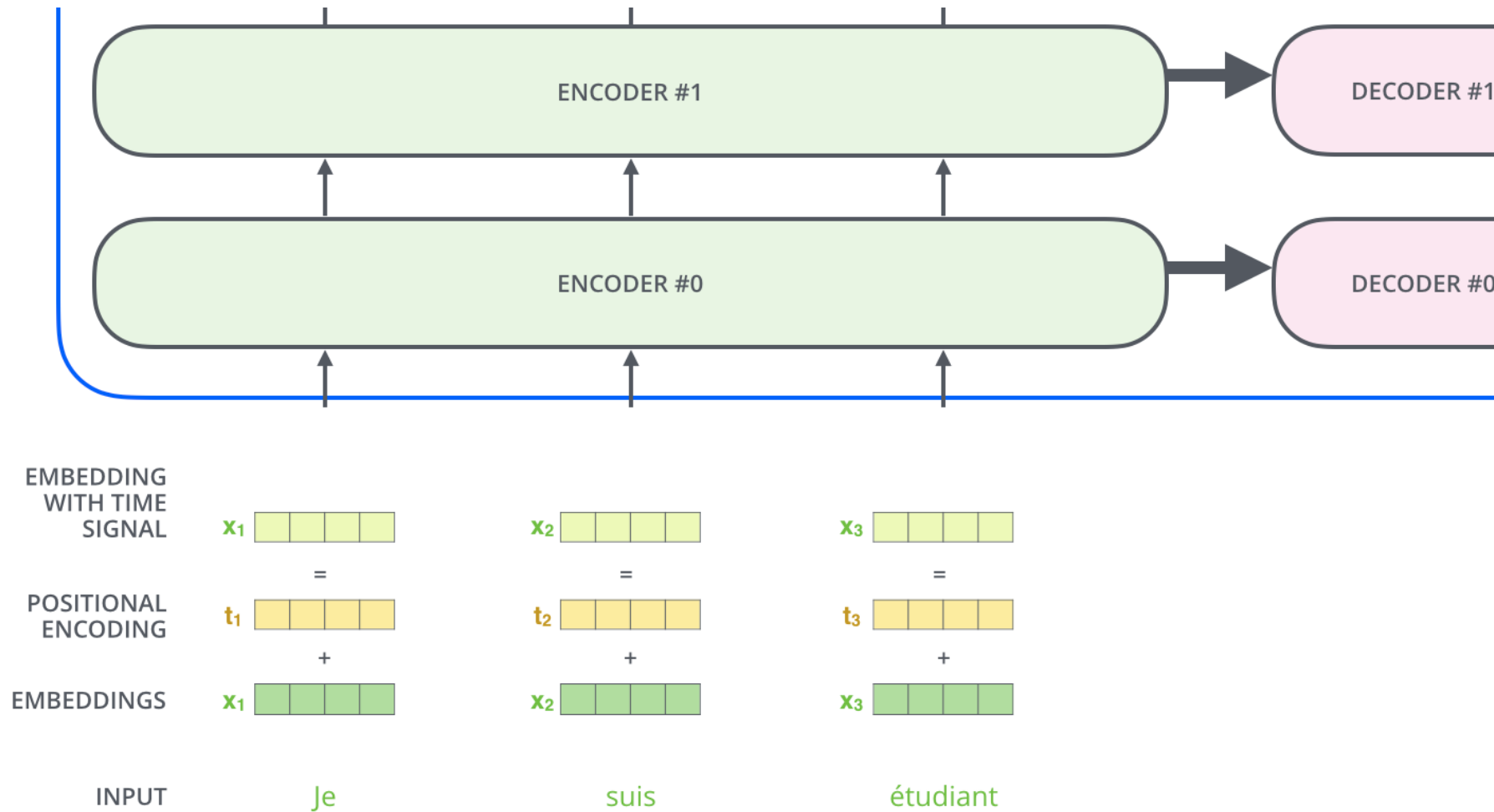
R



Positional encoding

- Сейчас у модели нет информации о порядке слов
- Попробуем её добавить путём прибавки чего-нибудь ко входным векторам

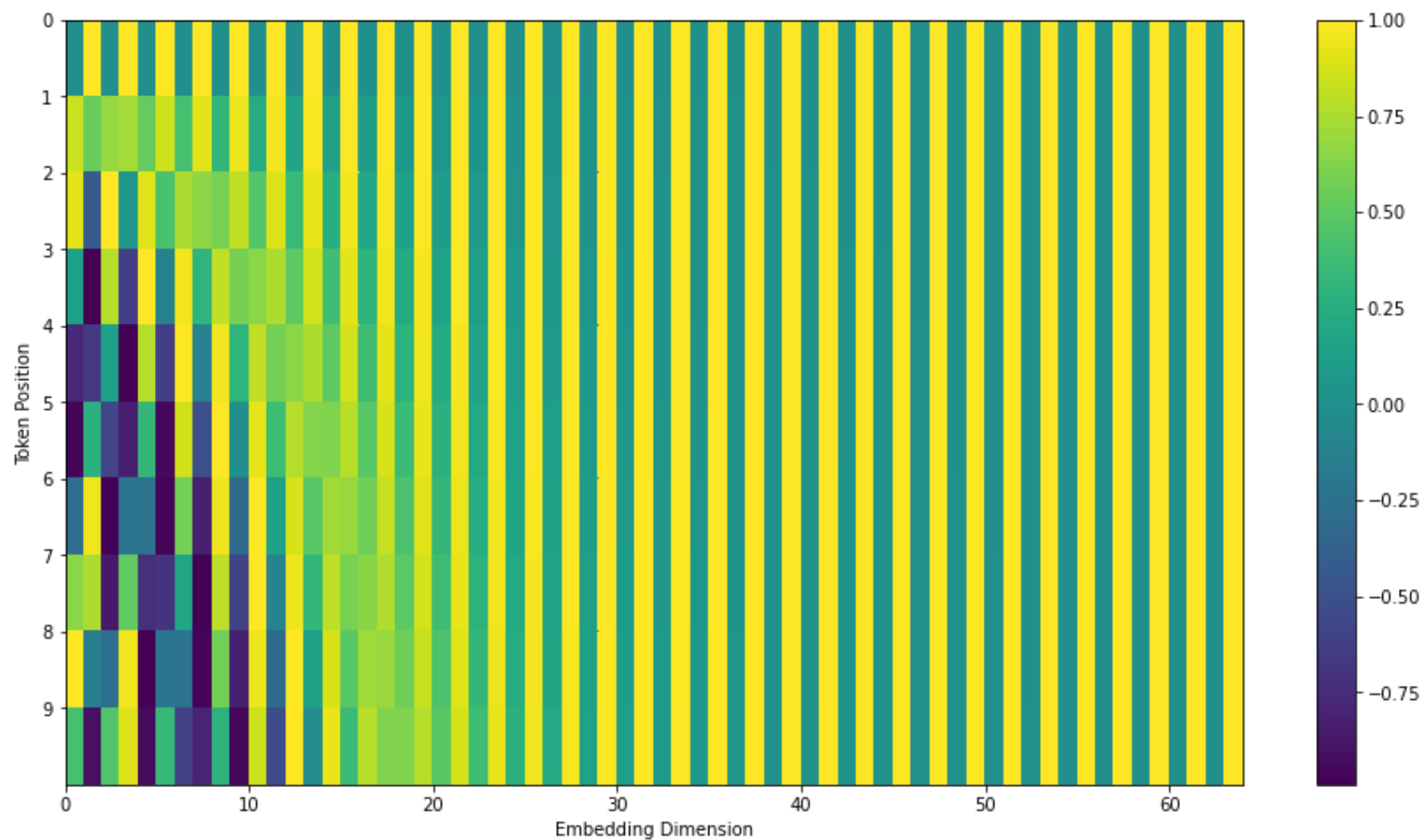
Positional encoding



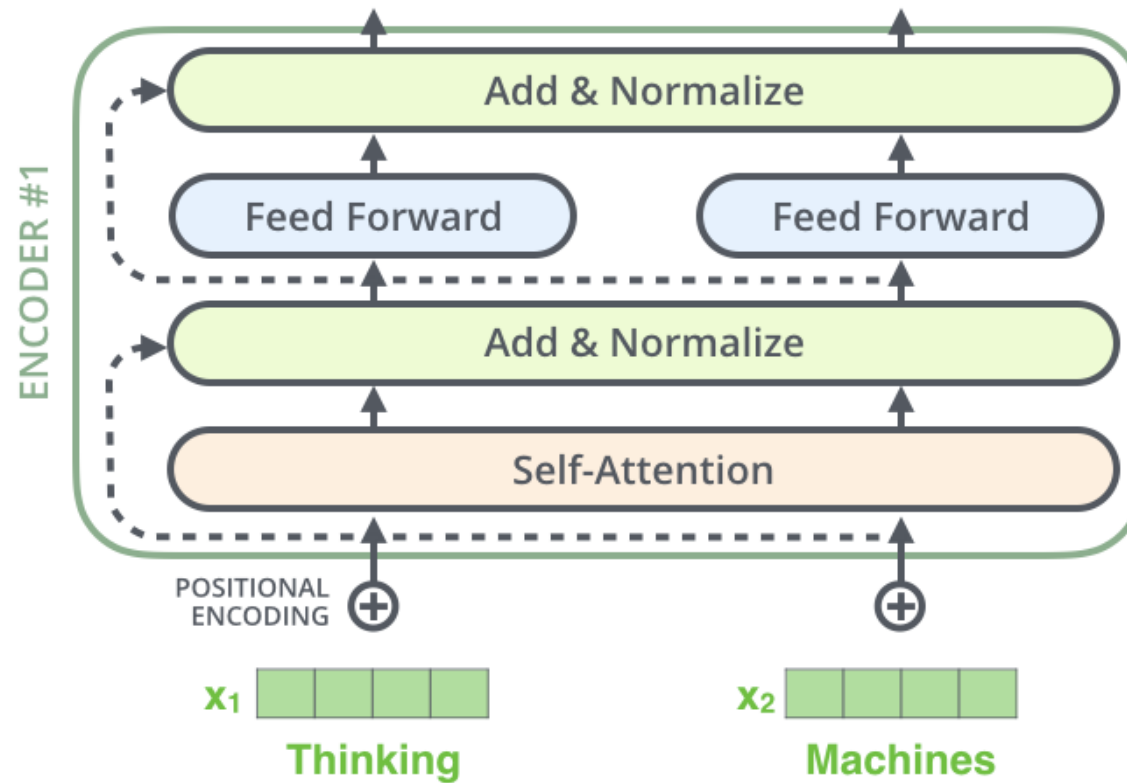
Positional encoding

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

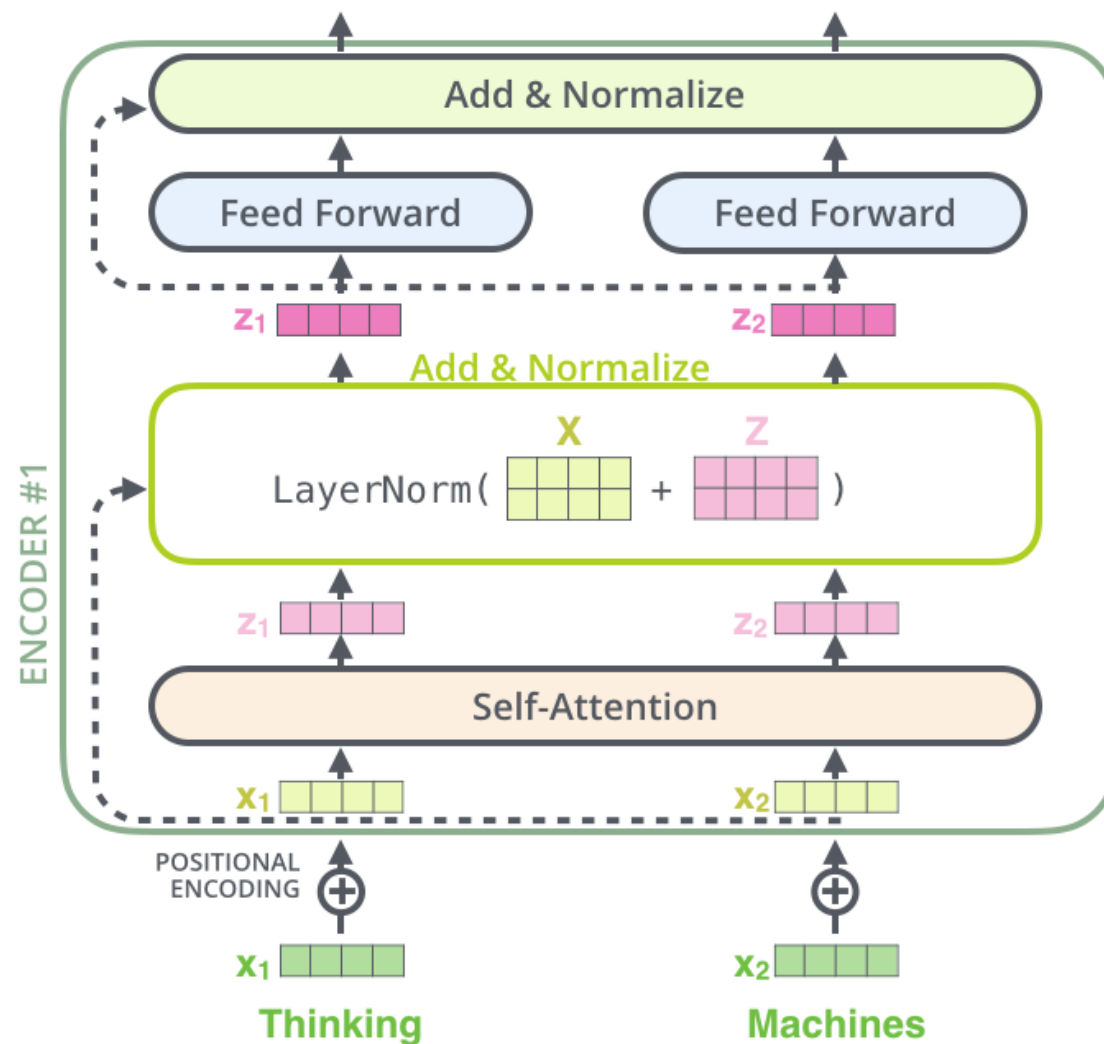
Positional encoding



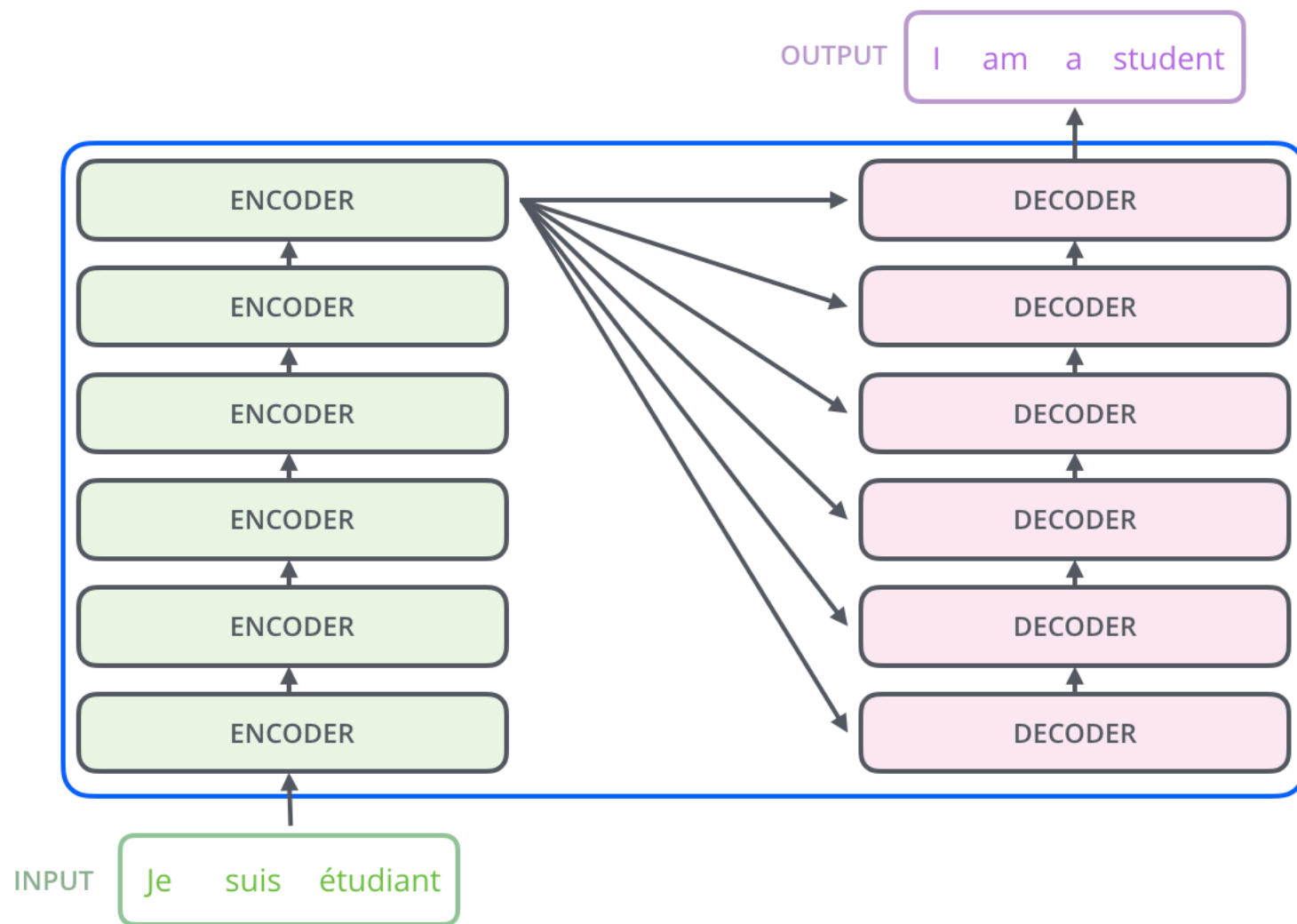
Кодировщик в трансформере



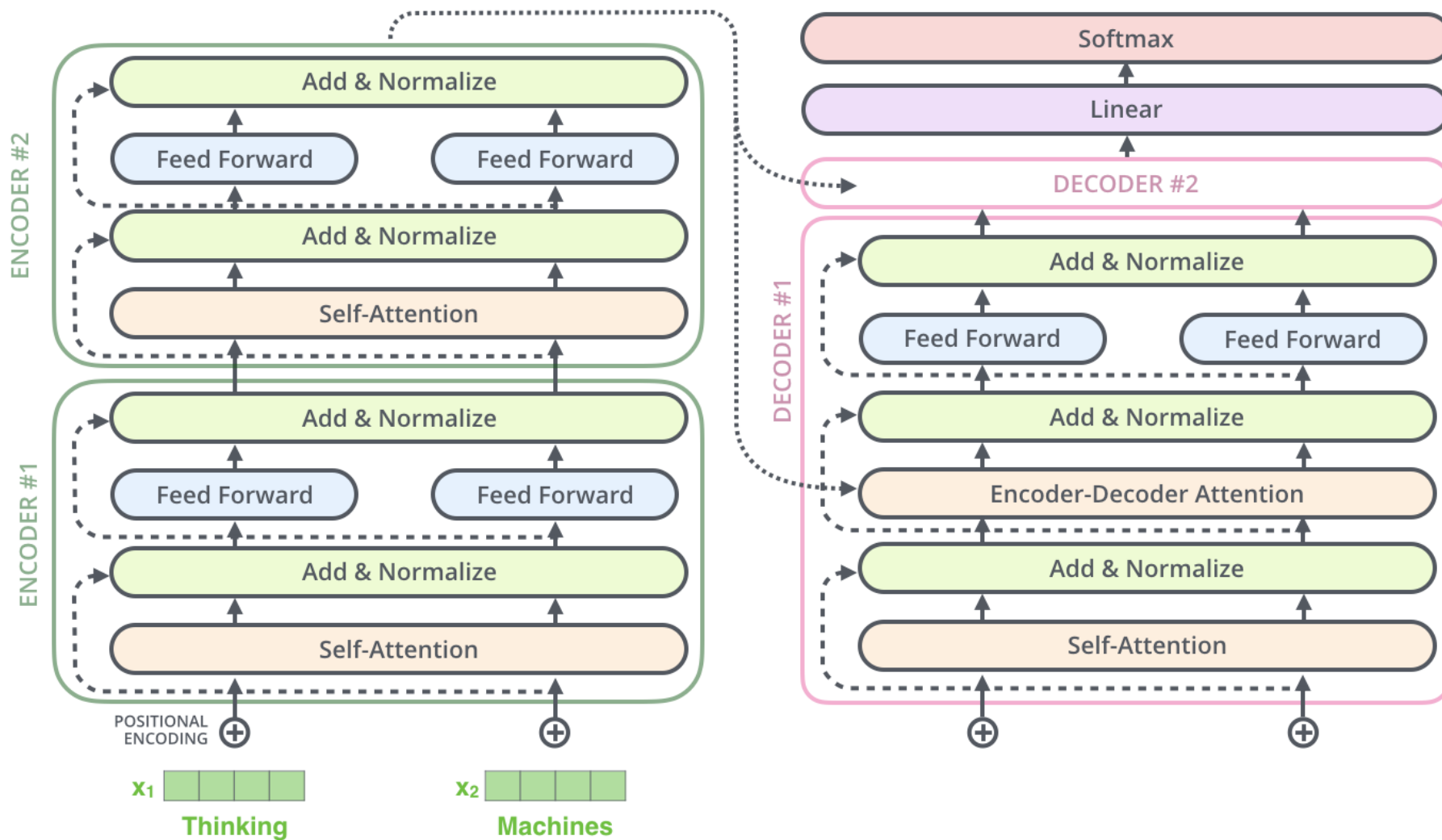
Кодировщик в трансформере



Трансформер



Деодировщик в трансформере



Encoder-Decoder Attention

- Векторы k_j и v_j получаются домножением матриц K_{encdec} и V_{encdec} соответственно на выходы последнего кодировщика
- Векторы q_j получаются стандартным образом из предыдущего слоя декодировщика

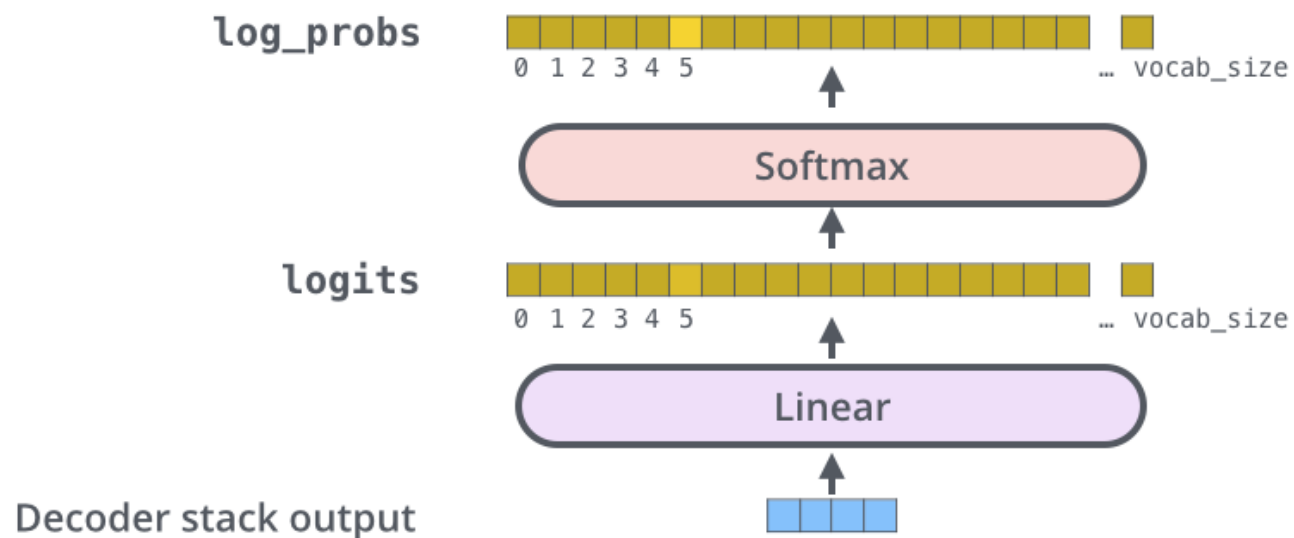
Self-attention в декодировщике

- Разрешается использовать только предыдущие слова

Выходной блок

Which word in our vocabulary
is associated with this index?

Get the index of the cell
with the highest value
(**argmax**)



Выходной блок

В случае с машинным переводом — «авторегрессионное» применение:

- Сначала декодировщик выдаёт одно слово
- Затем два (первое подаётся как вход)
- Затем три (первые два подаются ему как вход)
- И т.д.

Число параметров

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$	
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65	
(A)					1	512	512				5.29	24.9	
					4	128	128				5.00	25.5	
					16	32	32				4.91	25.8	
					32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58	
					32					5.01	25.4	60	
(C)	2									6.11	23.7	36	
	4									5.19	25.3	50	
	8									4.88	25.5	80	
		256			32	32				5.75	24.5	28	
		1024			128	128				4.66	26.0	168	
			1024								5.12	25.4	53
			4096								4.75	26.2	90
(D)							0.0				5.77	24.6	
							0.2				4.95	25.5	
									0.0	4.67	25.3		
									0.2	5.47	25.7		
(E)	positional embedding instead of sinusoids									4.92	25.7		
big	6	1024	4096	16					0.3	300K	4.33	26.4	213