



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

Компьютерная лингвистика и информационные технологии

Архитектура трансформер
(на основе материалов Jay Alammar и Е. Артемовой)

BERT

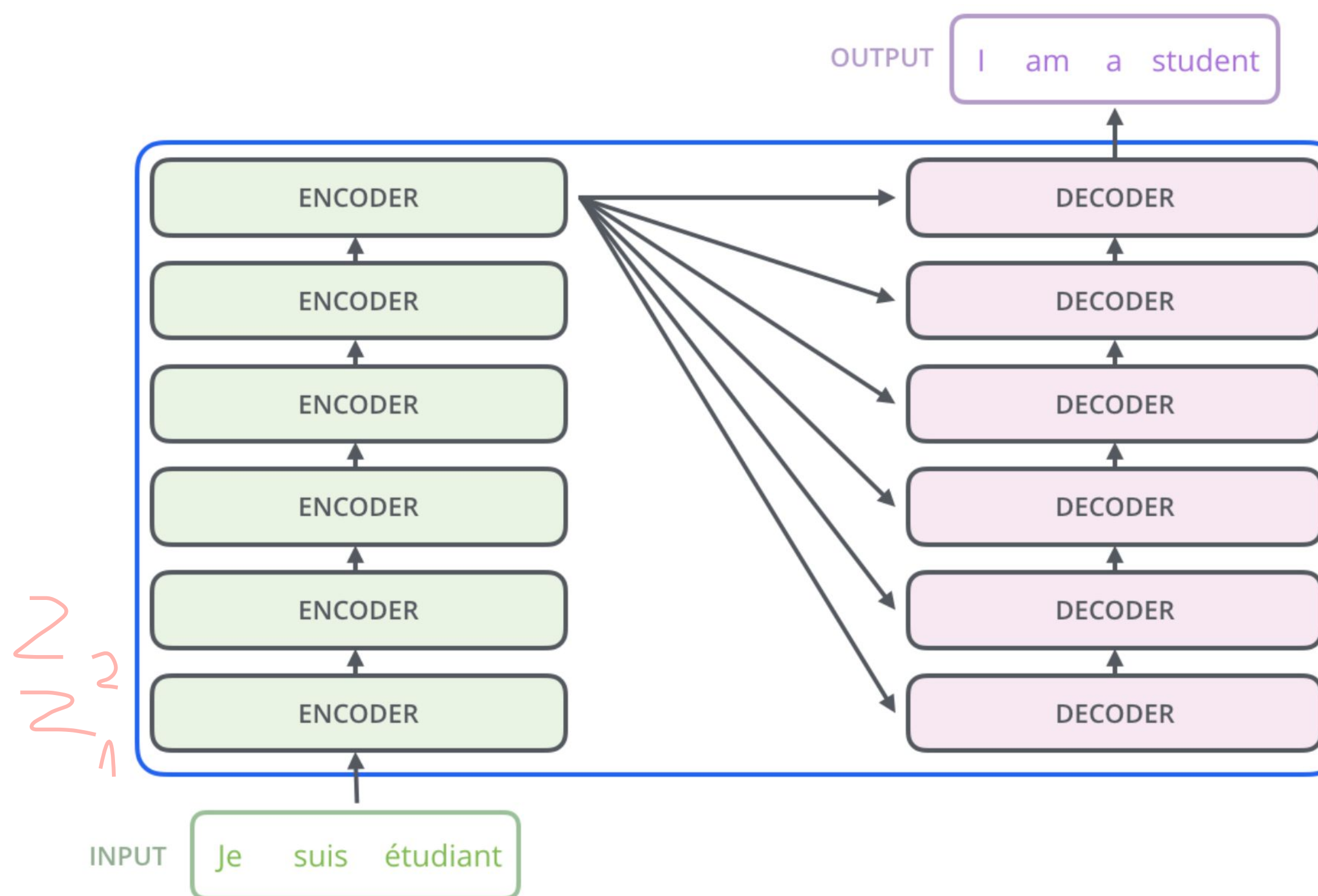
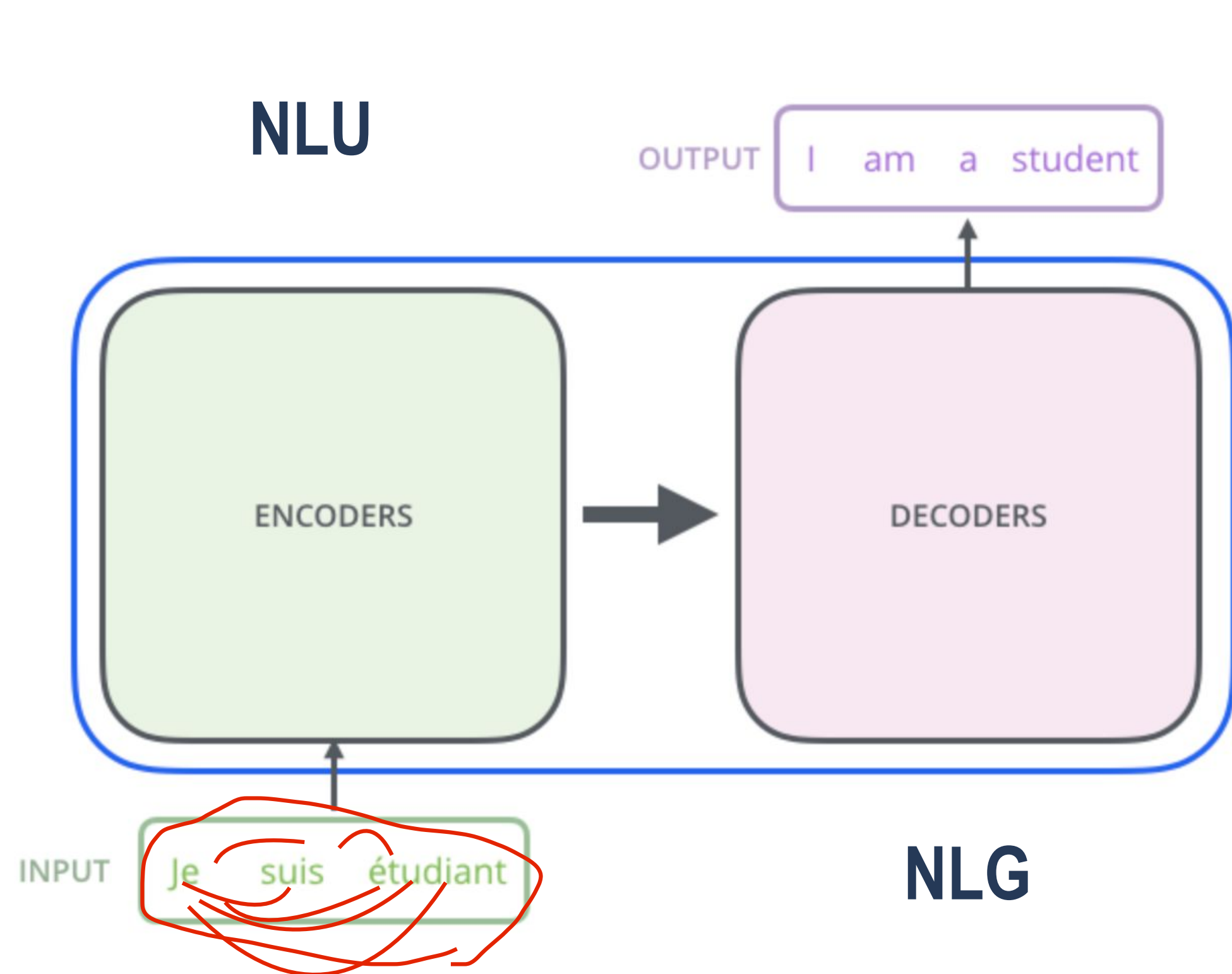
- BERT (Devlin et al., 2019);
- На основе архитектуры трансформер (Vaswani et al., 2017) для NMT;
- Новая парадигма: **обучение** большой языковой модели с использованием **training objectives** -> **дообучение** на целевой задаче;
- Прорыв в области АОЕЯ
- <http://jalamar.github.io/illustrated-transformer/>



Трансформер



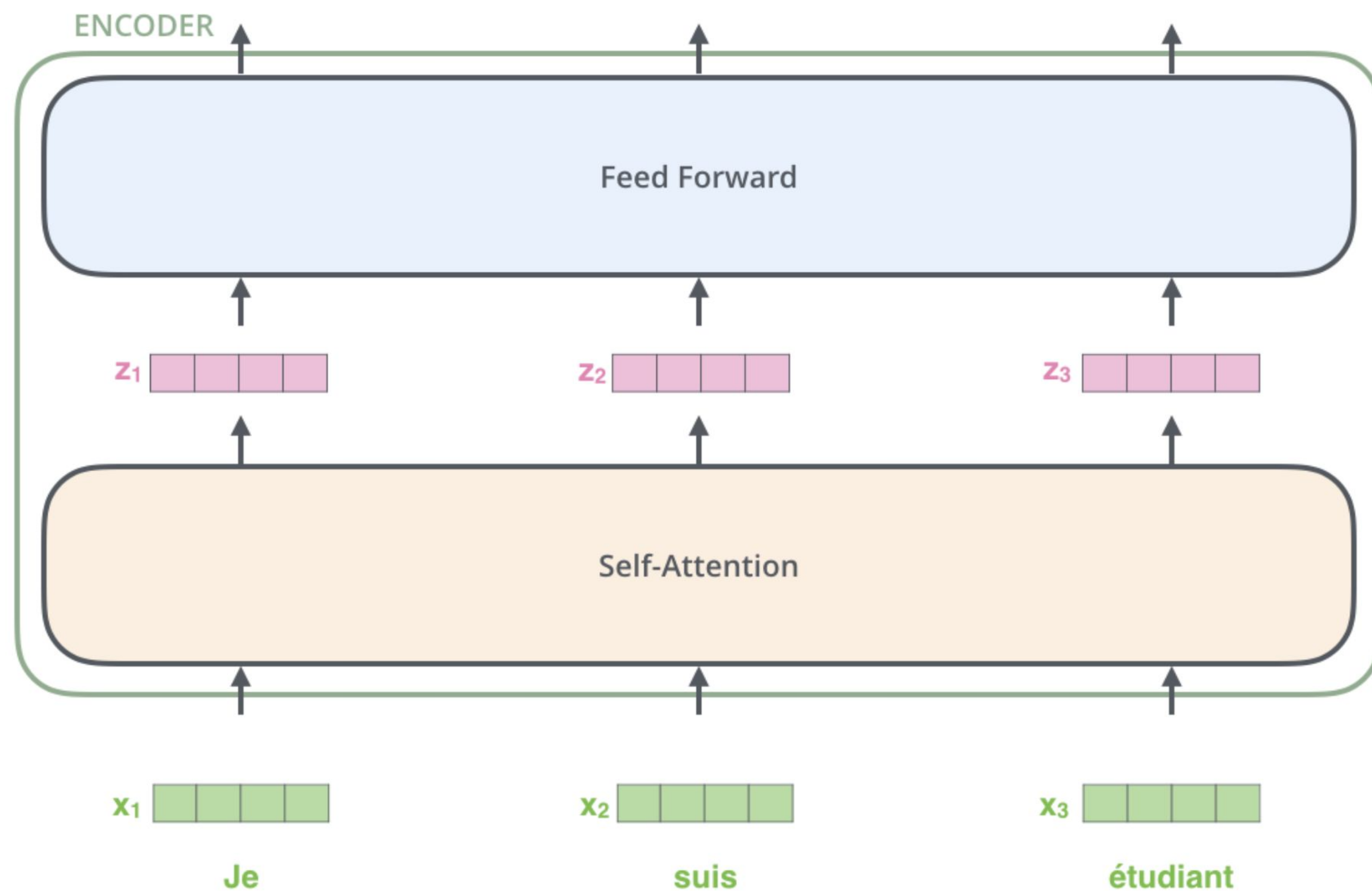
Энкодер-декодер на основе трансформера



The encoders are all identical in structure (yet they do not share weights). Each one is broken down into two sub-layers:

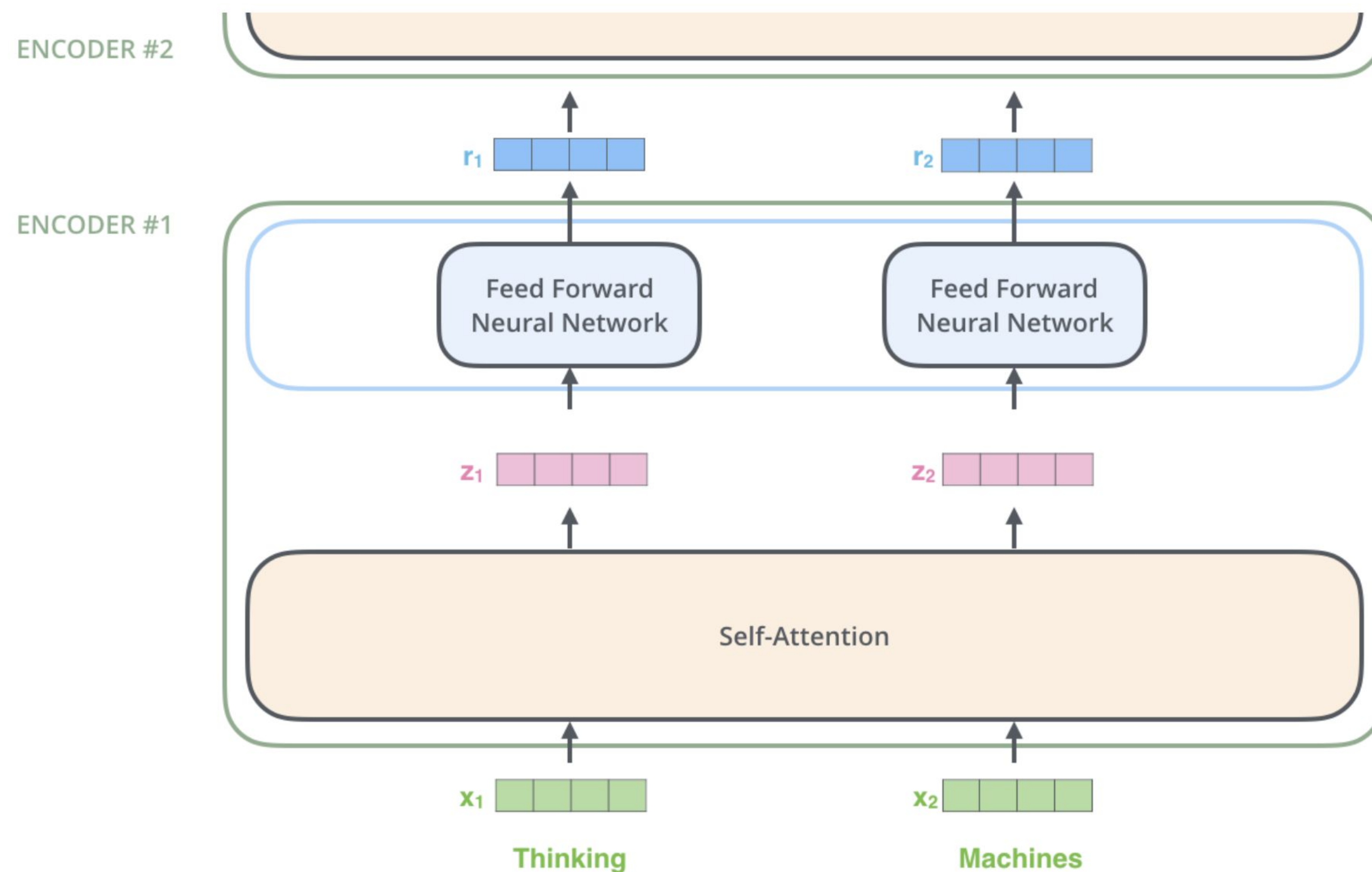
Блок энкодера

- Для простоты – на вход получаем предобученные пословные эмбединги
- Хотим, чтобы эмбединги z_i хорошо кодировали контекст



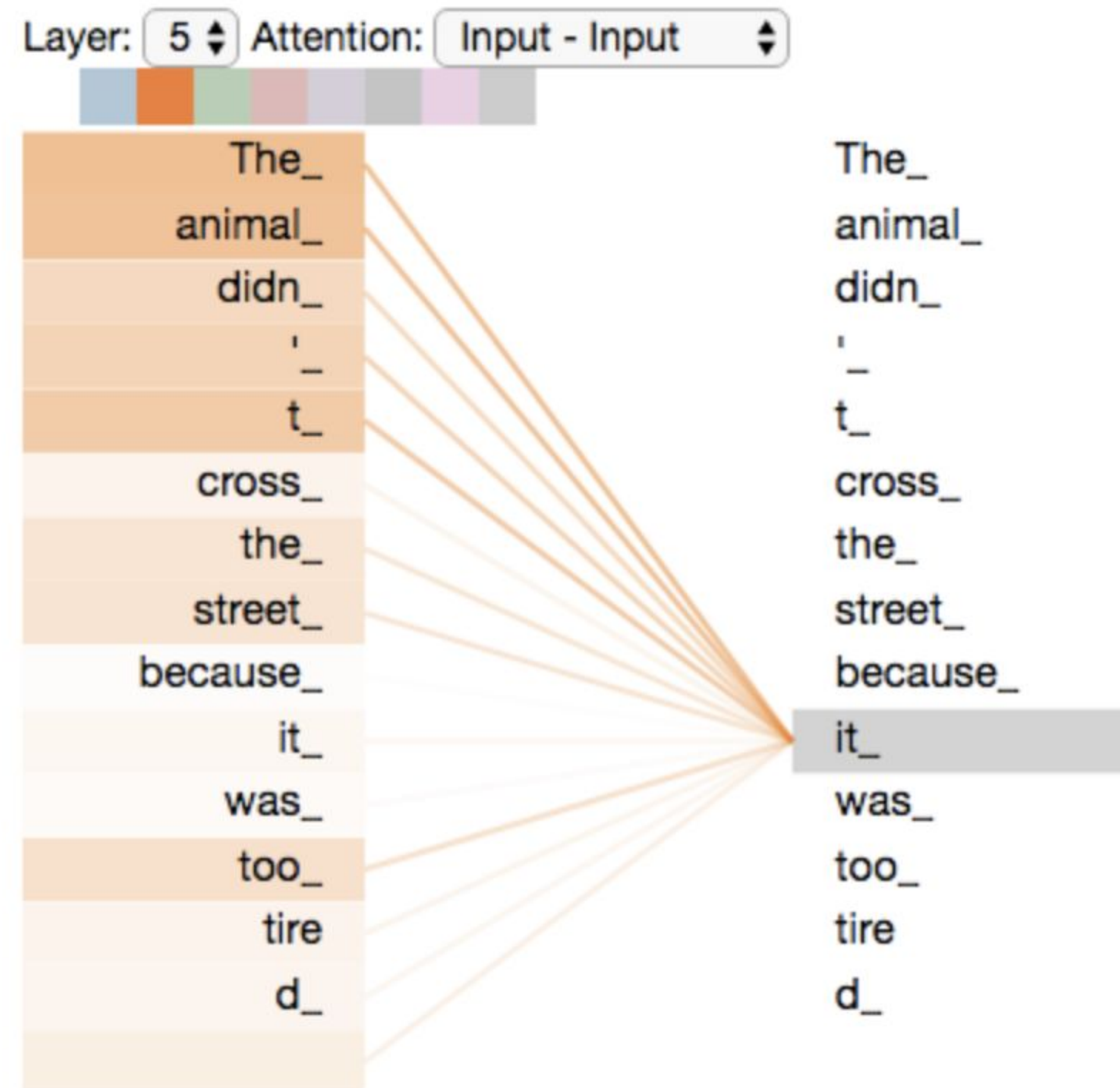
Блок энкодера

- К каждому входному элементу после self-attention независимо применяется FFN
- Каждый блок применяется последовательно



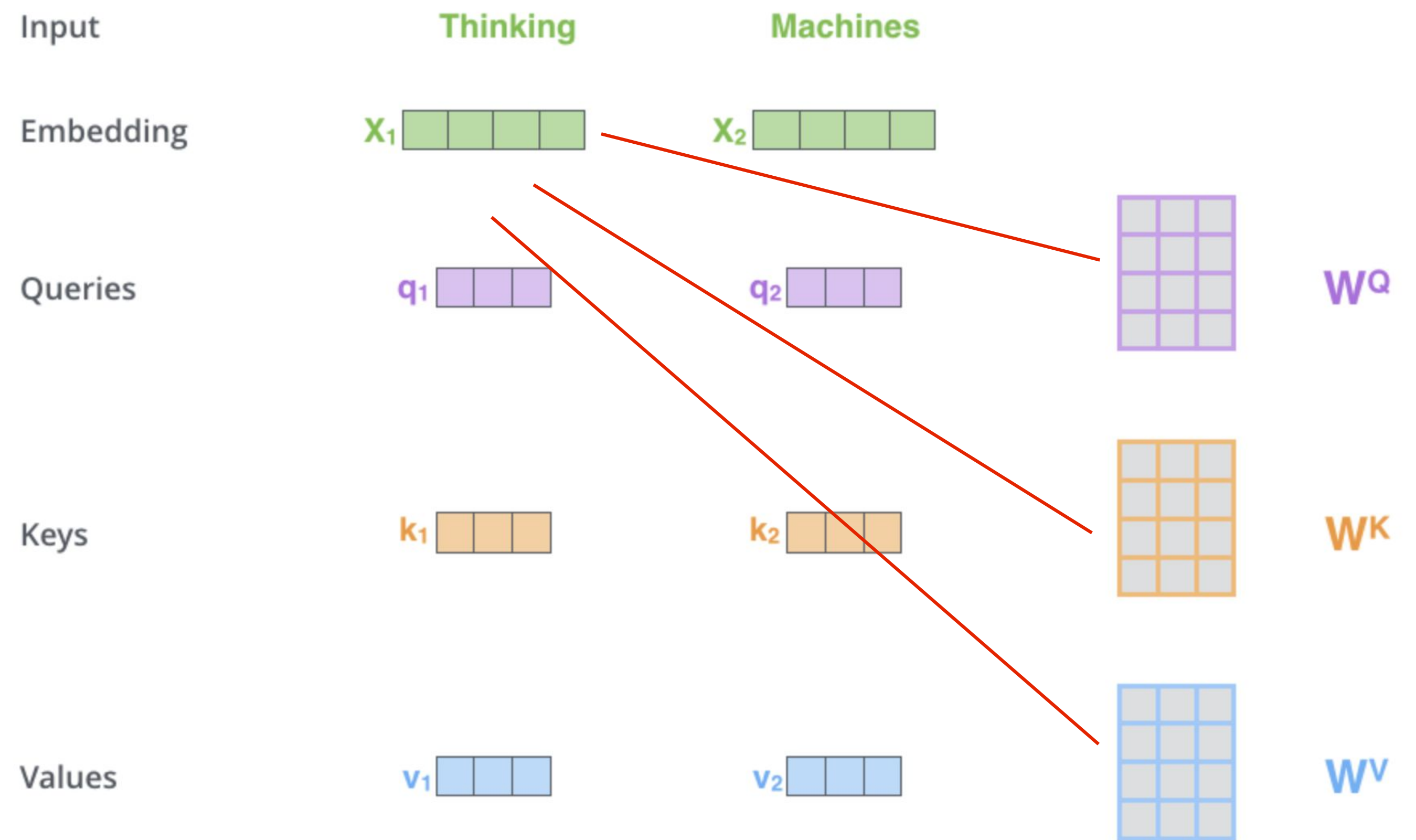
(Self-)Attention

- Для каждого элемента последовательности можем определить “значимость” других элементов в контексте
- Attention как метод интерпретации
- Хотим распространить “значимость” элементов последовательности вверх по сети



Self-Attention

- Для каждого входного вектора хотим получить ключ, запрос и значение (key, value and query)
- Имеем три обучаемые матрицы: W^Q , W^K , W^V
- Умножаем входные векторы на эти матрицы, получаем векторы q , k и v (линейное преобразование)



Self-Attention

- Воспринимаем элемент последовательности как запрос (q)
- Перебираем все комбинации $q_i * k_j$ и считаем промежуточные скоры, включая сам элемент

Input

Embedding

Queries

Keys

Values

Score

Thinking

x_1

q_1

k_1

v_1

$$\underline{q_1} \cdot k_1 = 112$$

Machines

x_2

q_2

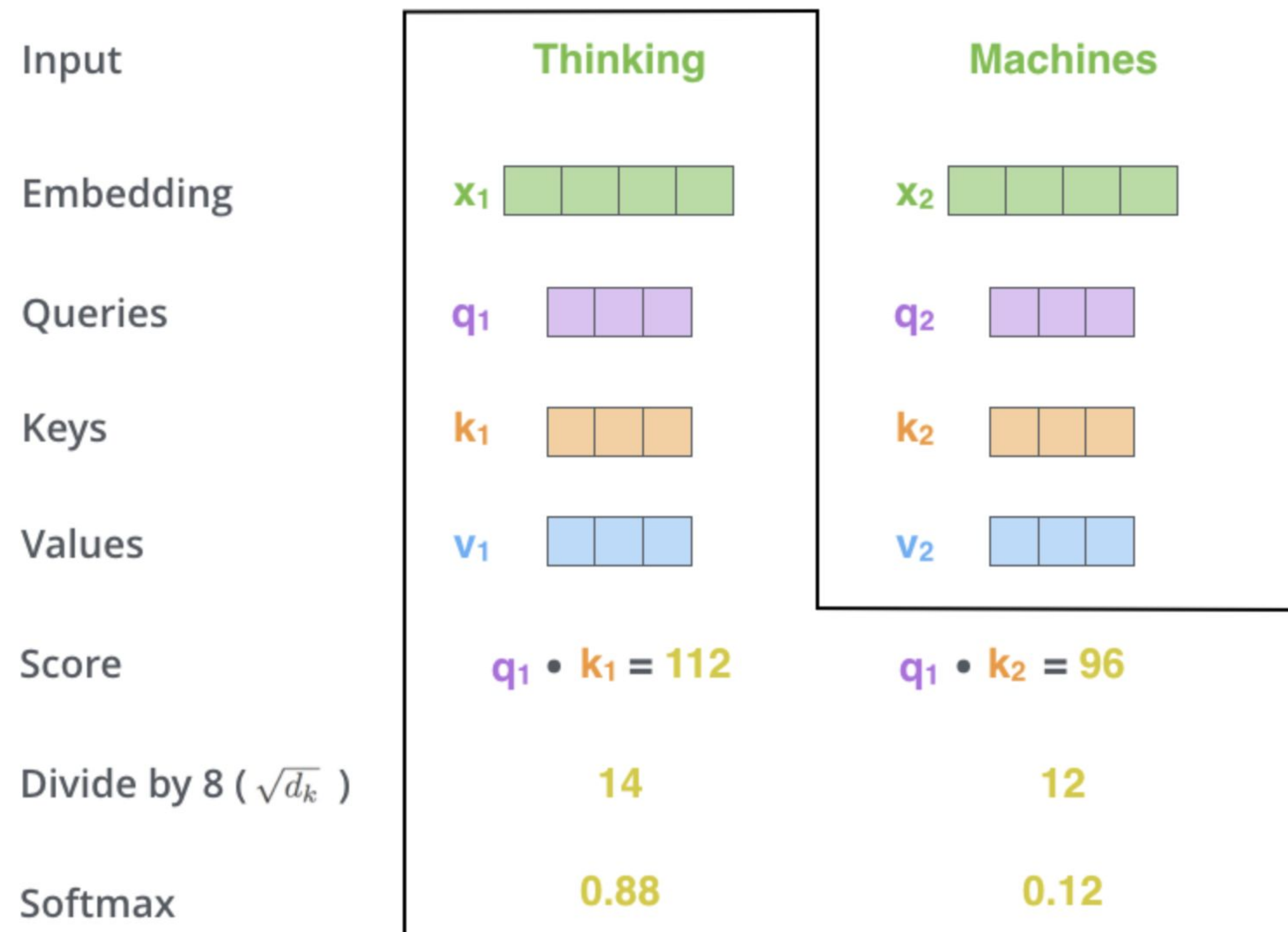
k_2

v_2

$$\underline{q_1} \cdot k_2 = 96$$

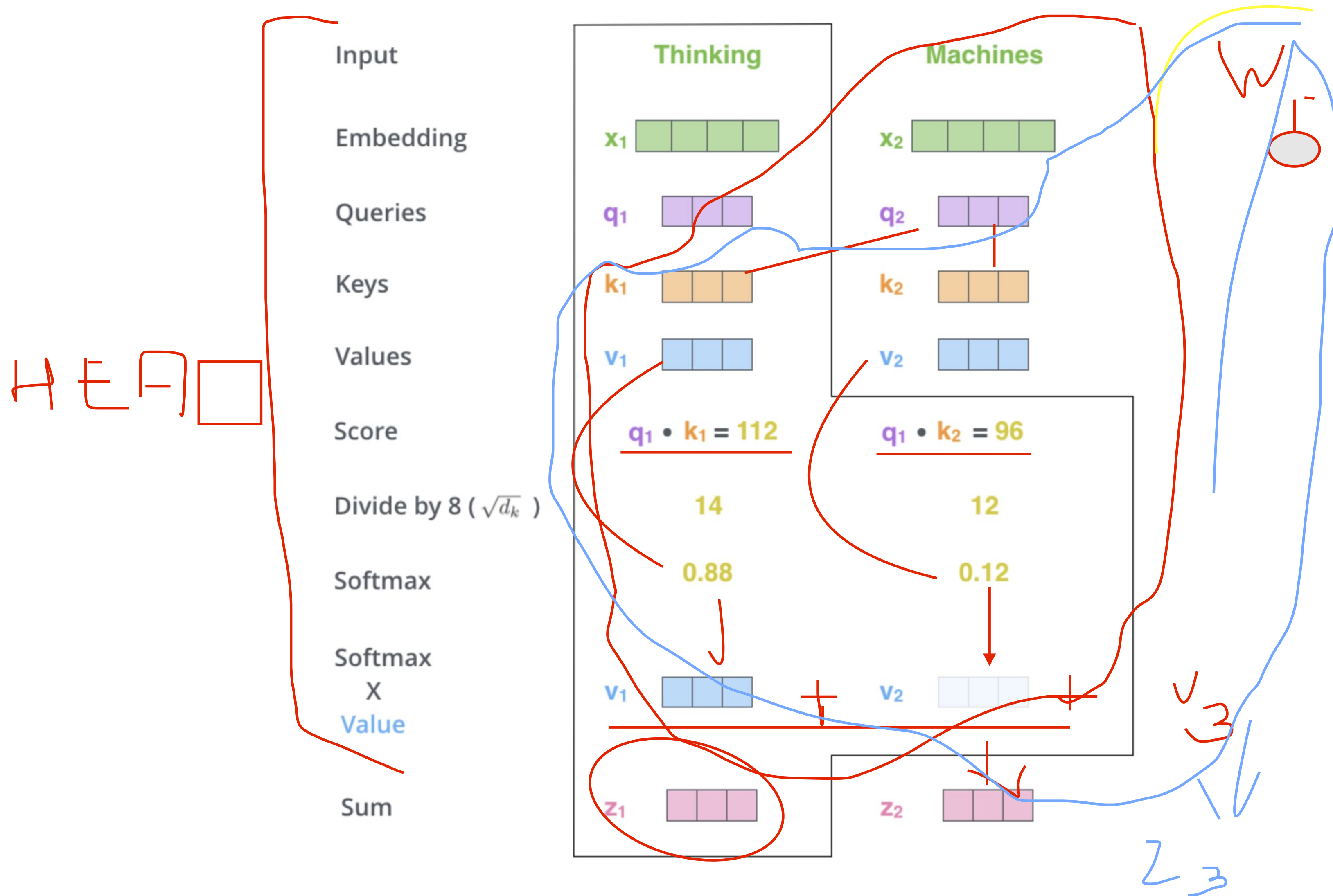
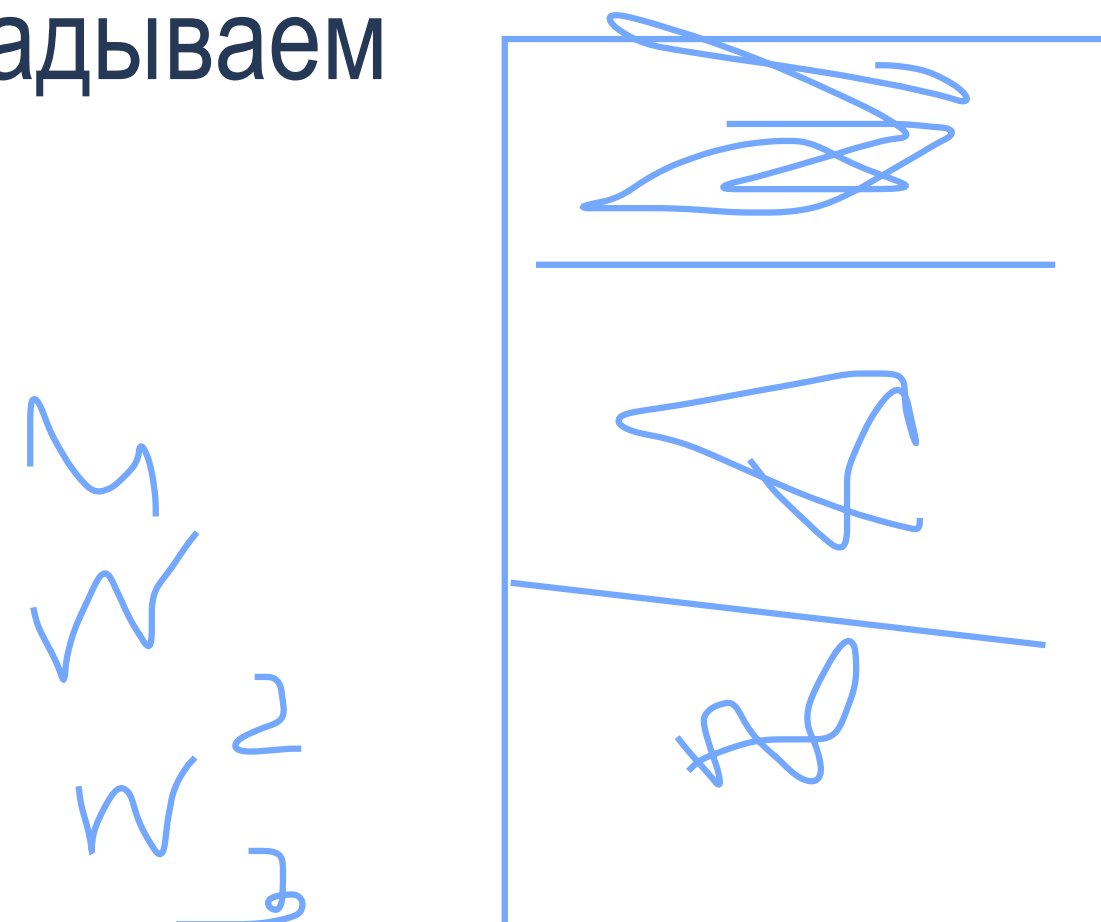
Self-Attention

- Скоры (результаты скалярных произведений на шаге 2) делим на корень из d_k
- d_k – гиперпараметр (возьмем за данность)
- Берем софтмакс по вектору скоров, чтобы получить веса внимания



Self-Attention


- Умножаем все векторы v_i на полученные веса внимания
- Все полученные значения складываем



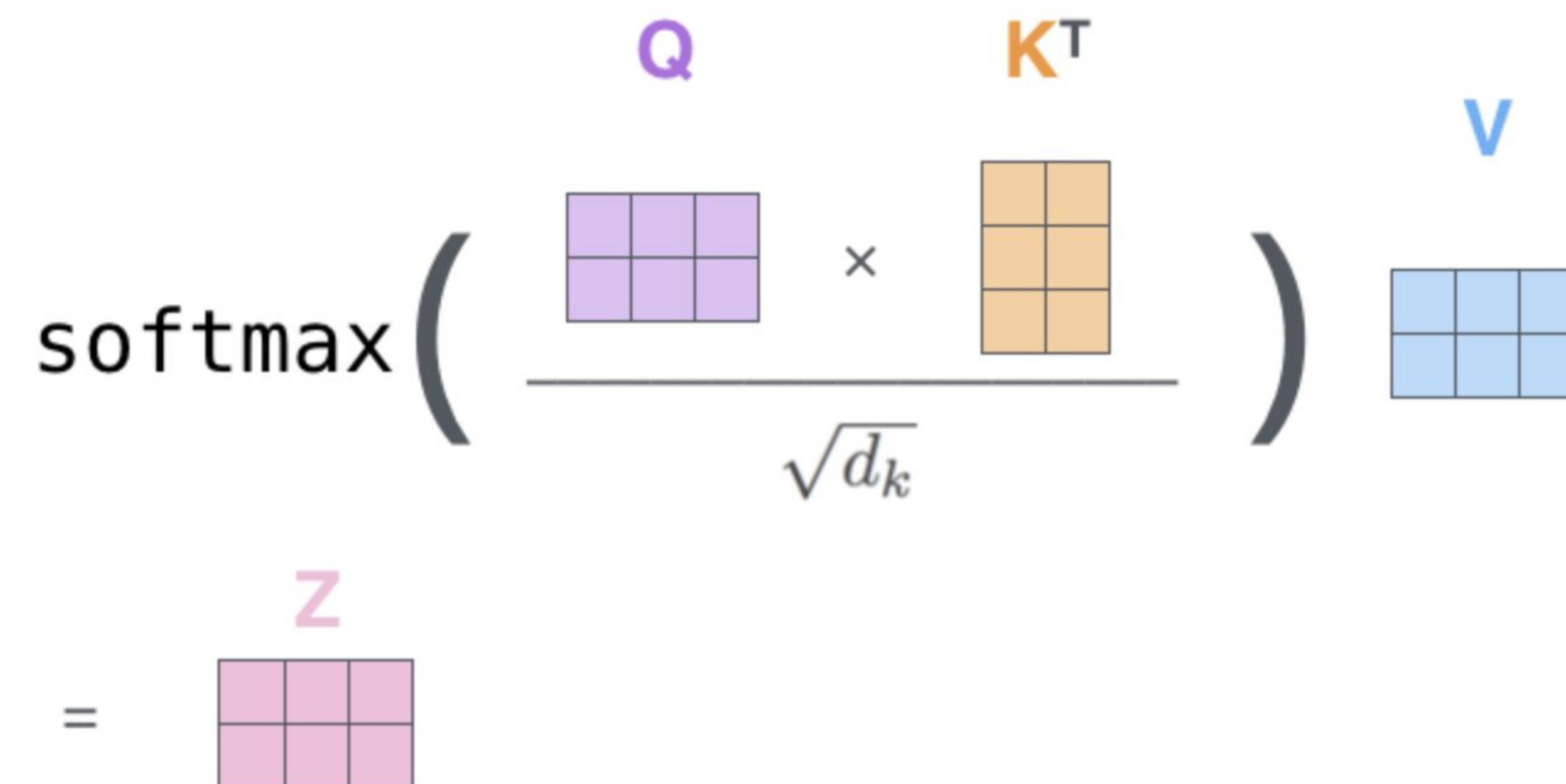
Self-Attention



$$X \times W^Q = Q$$


$$X \times W^K = K$$


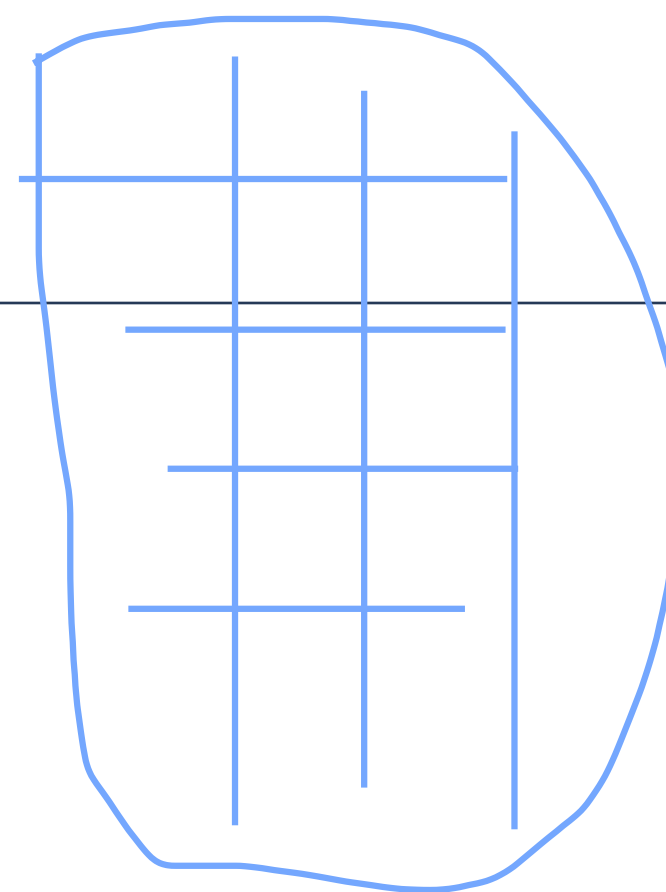
$$X \times W^V = V$$


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


The self-attention calculation in matrix form

Змей Горыныч

- Имеем k голов, для каждой i -ой головы получаем вектор z_i
- Голова – отдельный механизм self-attention (набор из 3-х матриц)
- Считаем self-attention для каждой головы независимо
- Ширина и глубина модели

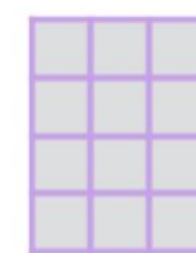
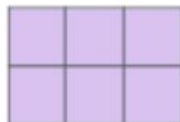


Thinking
Machines X



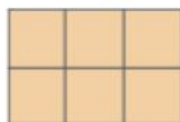
ATTENTION HEAD #0

Q_0



W_0^Q

K_0



W_0^K

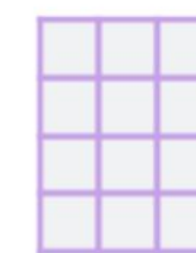
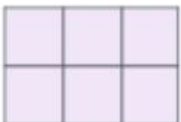
V_0



W_0^V

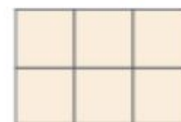
ATTENTION HEAD #1

Q_1



W_1^Q

K_1



W_1^K

V_1



W_1^V

With multi-headed attention, we maintain separate Q/K/V weight matrices for each head resulting in different Q/K/V matrices. As we did before, we multiply X by the $W_Q/W_K/W_V$ matrices to produce Q/K/V matrices.



Змей Горыныч

512

LINEAR(d^a_n, d)

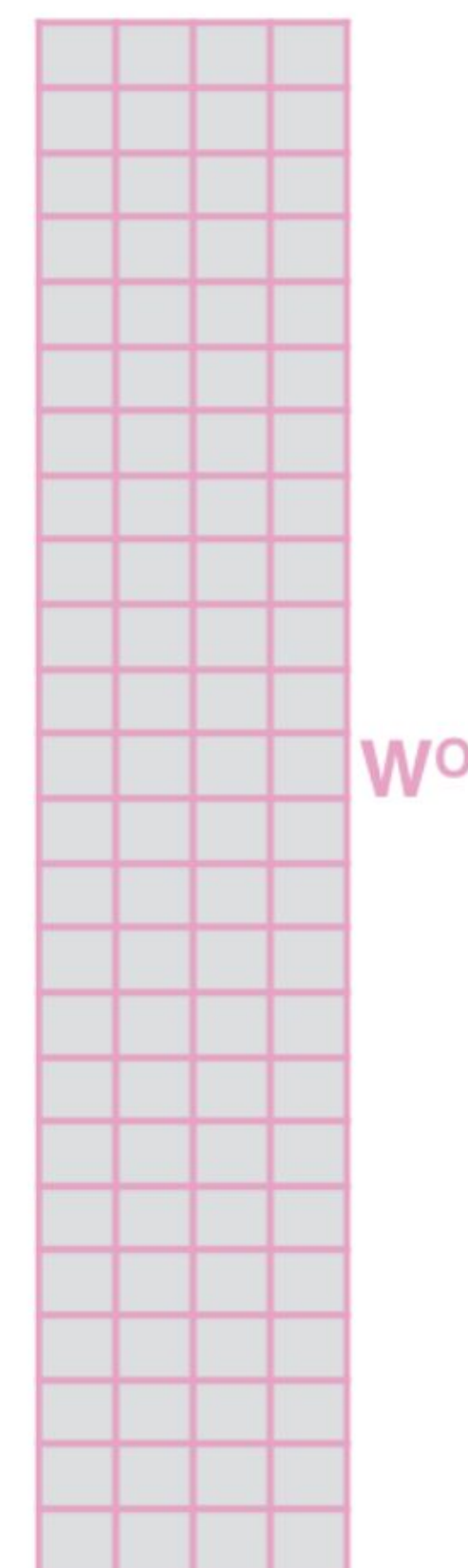
1) Concatenate all the attention heads



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

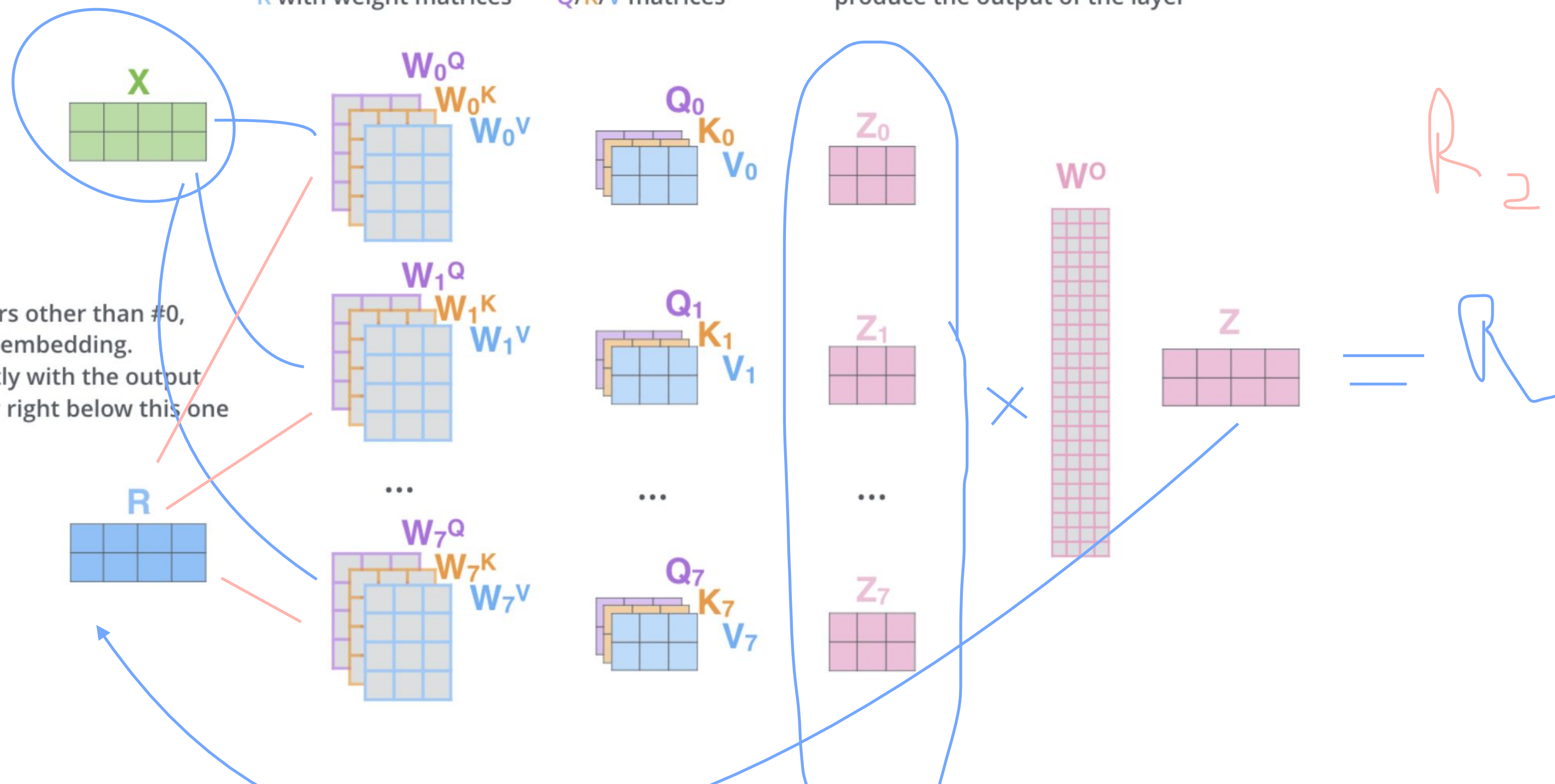


Змей Горыныч

- 1) This is our input sentence*
- 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

Thinking Machines

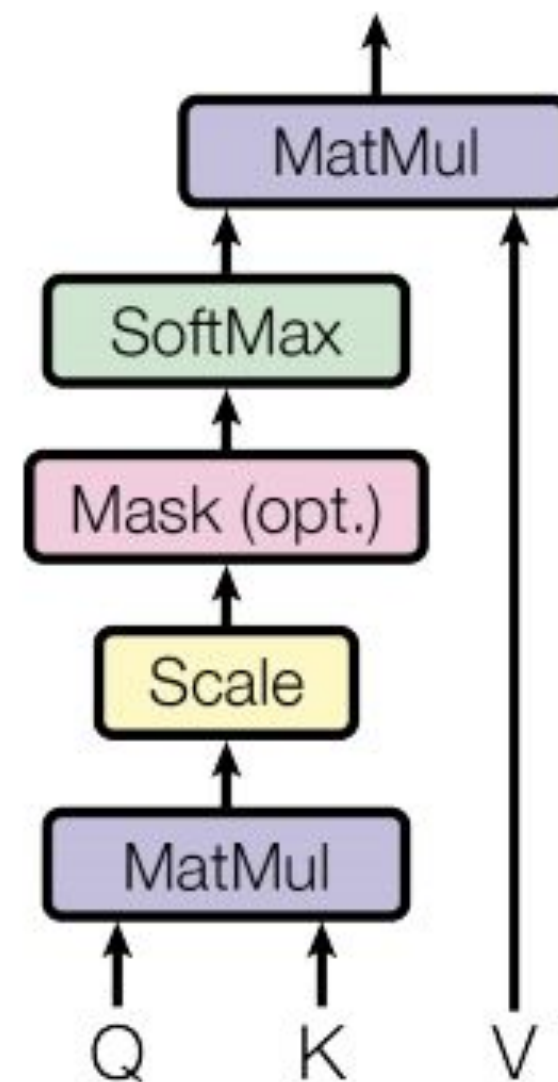
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one





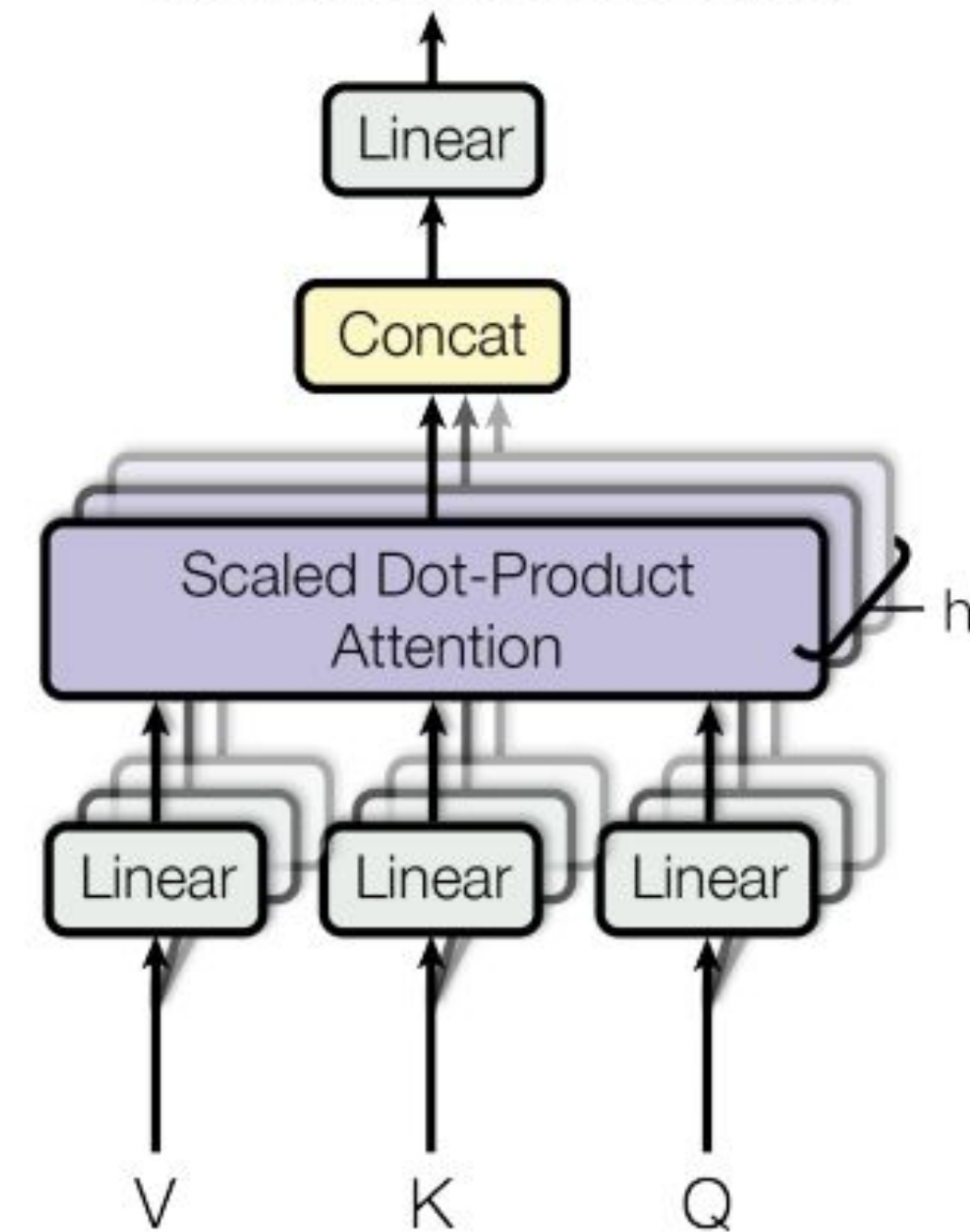
Scaled Dot-Product Attn vs Multi-Head Attn

Scaled Dot-Product Attention



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

Multi-Head Attention

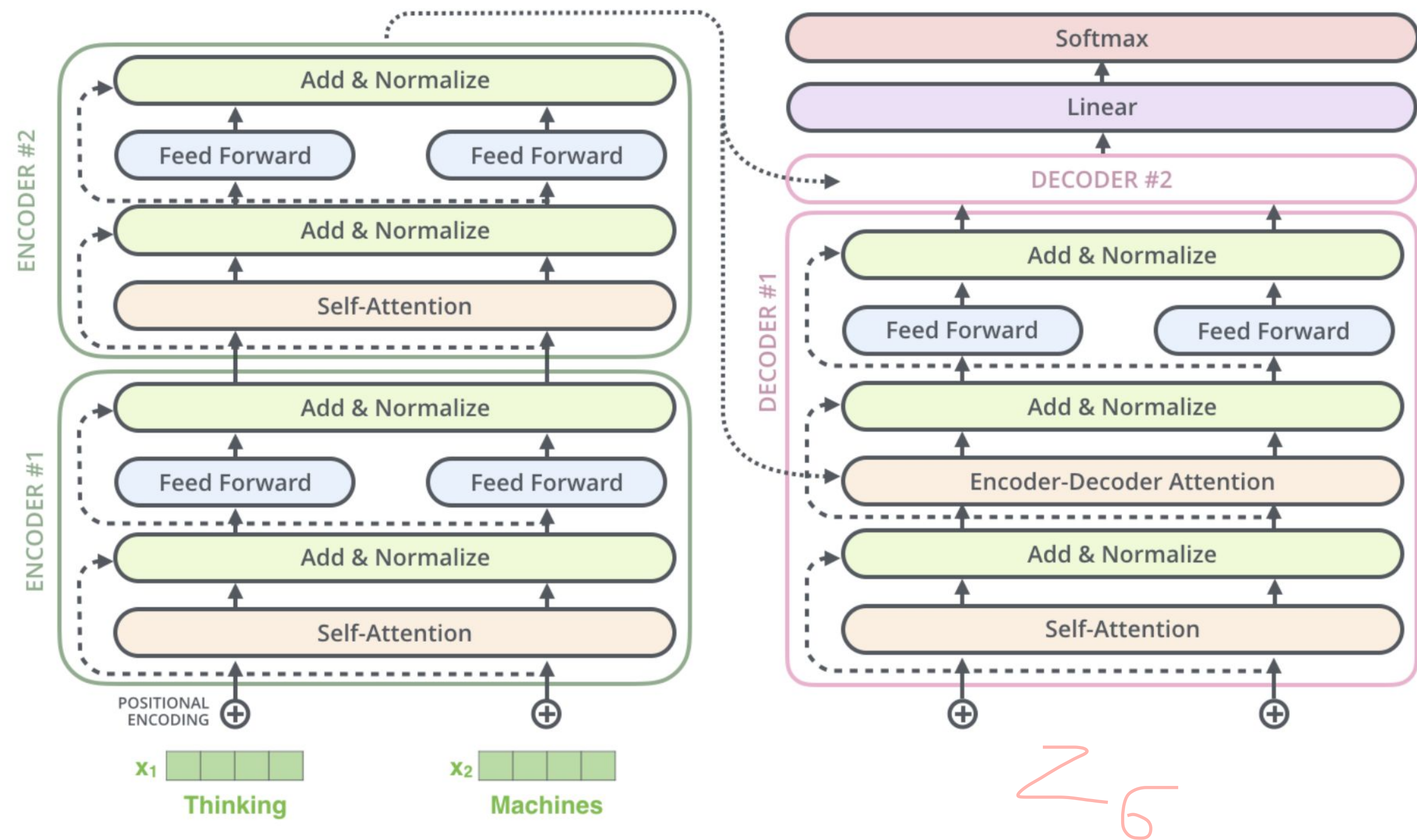


$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

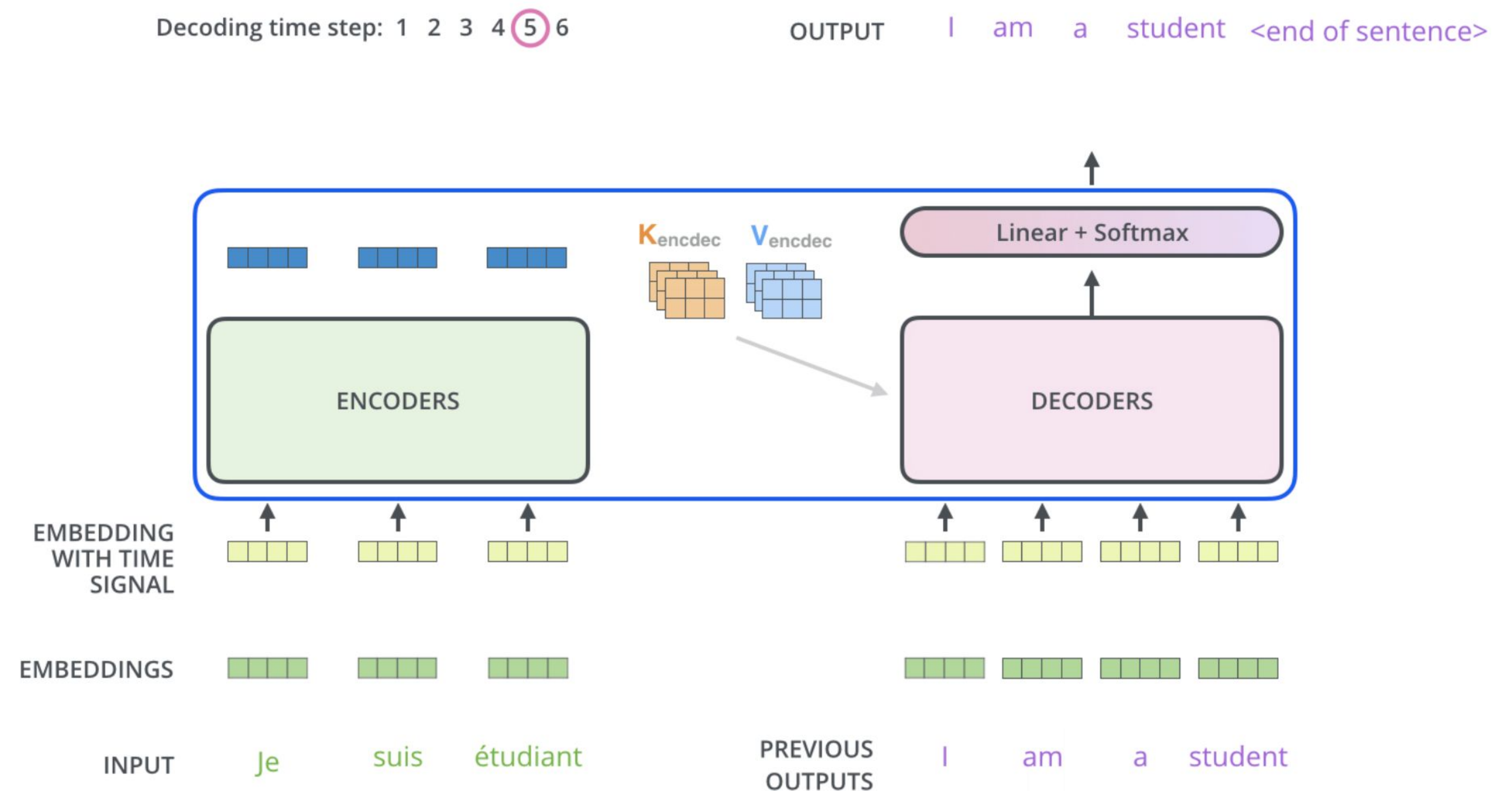
Декодер

- Encoder-Decoder Attention: декодер получает выход с последнего блока энкодера
- k, v (enc), q (dec)
- Хотим сформировать вектор контекста (по аналогии с RNN)



Декодер

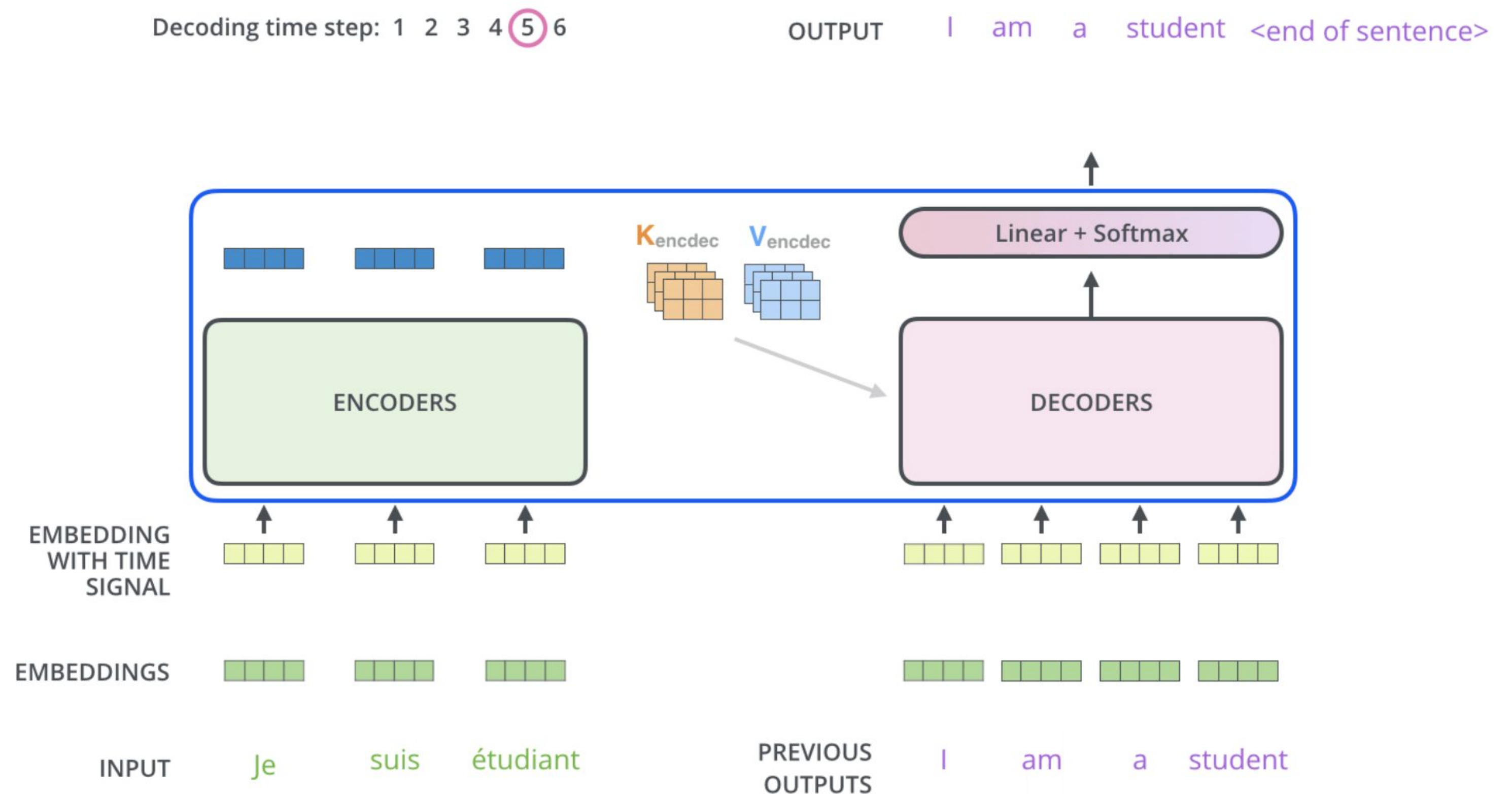
- Верим, что выход 6-ого блока энкодера есть хорошая репрезентация контекста
- Матрицы K_{encdec} и V_{encdec} обучаются, чтобы получить векторы k и v для декодера (общие для каждого блока декодера)





Декодер

- Декодер генерирует элементы последовательности друг за другом (как ЯМ)
- Энкодер может видеть все элементы последовательности, в то время как декодер в момент времени видит подмножество элементов (маска)



Декодер

- Для каждого предсказываемого элемента последовательности после линейного преобразования получаем набор чиселок – logits (по самому правому элементу в момент времени)
- Берем софтмакс

Which word in our vocabulary is associated with this index?

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

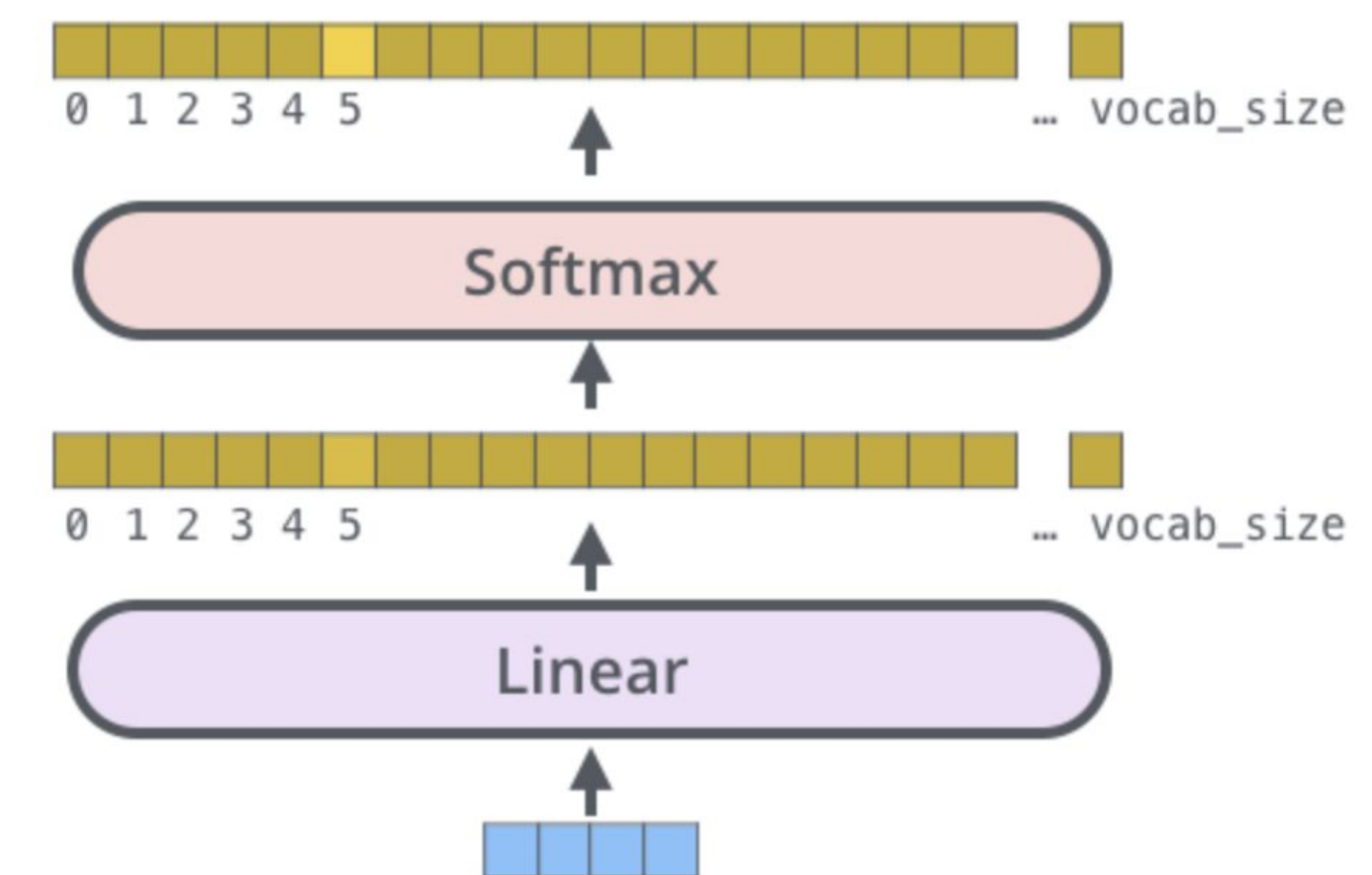
log_probs

am

5

logits

Decoder stack output



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.



Трансформер

- Non-sequential: обрабатываем элементы входа одновременно
- Информация о позиции элемента: позиционное кодирование
- Можно распараллелить вычисления
- Self-attention
- Кодирование длинных последовательностей и отношений между их элементами

RNN

- Sequential: обрабатываем элементы входа последовательно
- Информация о позиции элемента: последовательная обработка
- Нельзя распараллелить вычисления
- Кодирование элемента в момент времени имеет большое влияние на кодирование следующих элементов