

# Лекция 8

## Машинный перевод и механизм внимания

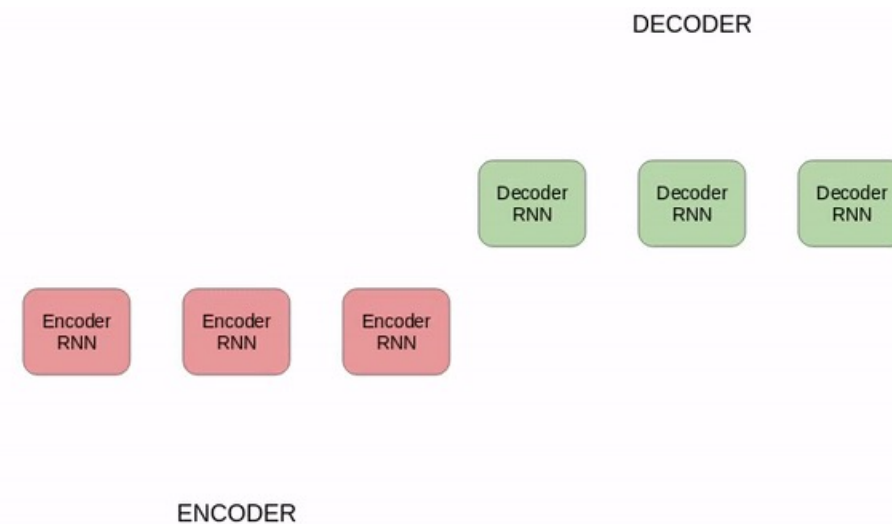
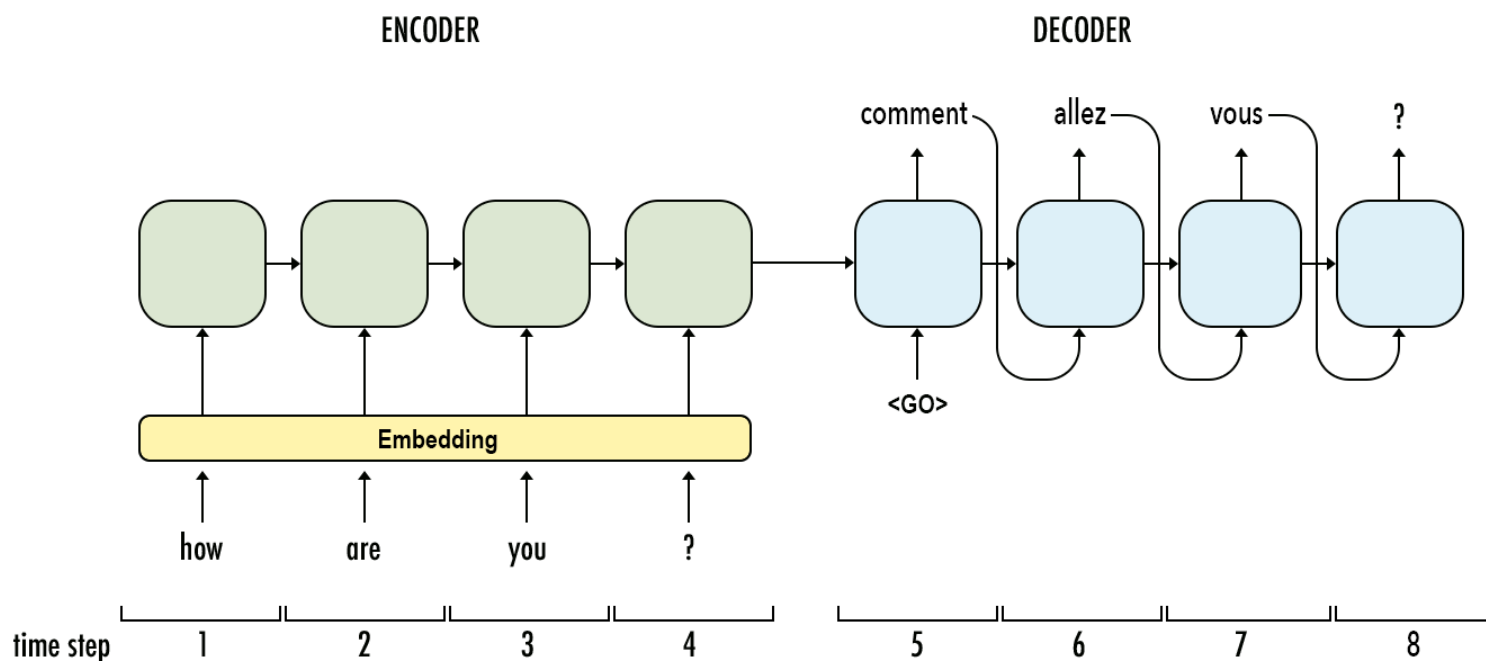
Разработка нейросетевых систем

Канев Антон Игоревич

# Seq2seq

- Для машинного перевода
- Состоит из двух частей, например две LSTM
- Можно добавить внимание

## Neural Machine Translation SEQUENCE TO SEQUENCE MODEL WITH ATTENTION



# Последовательность -> Последовательность

## Машинный перевод

### Пример:

ITA: Il gatto si e' seduto sul tappetino.



EN: The cat sat on the mat.

### Проблемы:

- Выравнивание: последовательности на входе / выходе могут иметь разную длину
- Неопределенность (отображение 1-ко-многим: множество возможных способов перевода)
- Метрика: как автоматически оценивать, означают ли предложения к одно и то же?

Последовательность -> Последовательность

## Машинный перевод

**Пример:**

ITA: Il gatto si e' seduto sul tappetino.



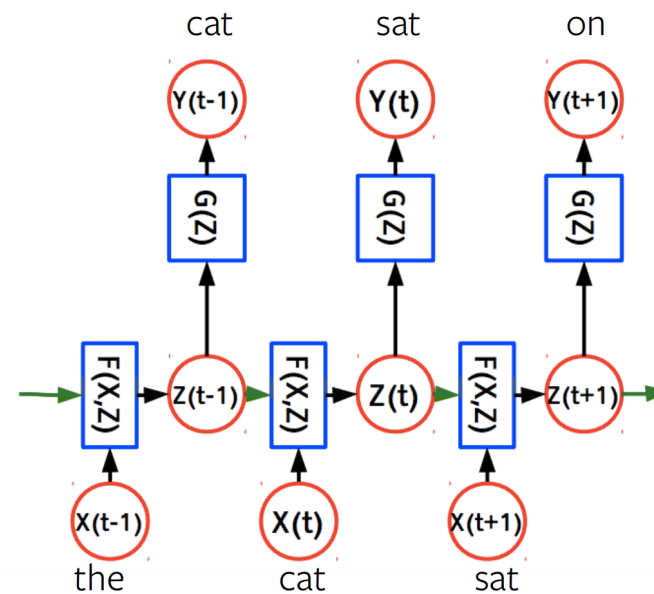
EN: The cat sat on the mat.

**Подход:**

Имеем один RNN для кодирования исходного предложения, а другой RNN - для предсказания целевого предложения. Целевой RNN учится (мягко) выравнивать предложение с помощью механизма внимания.

# Последовательность -> Последовательность

## Машинный перевод



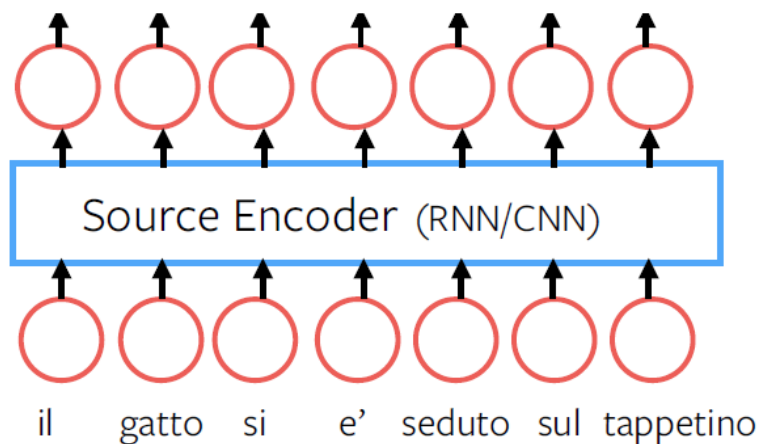
Y. LeCun's diagram

# Последовательность -> Последовательность

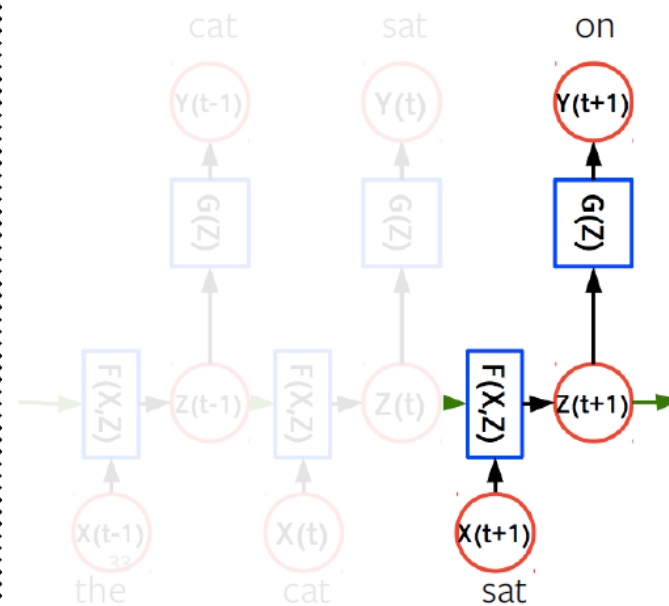
## Машинный перевод

Source

1) Represent source



Target



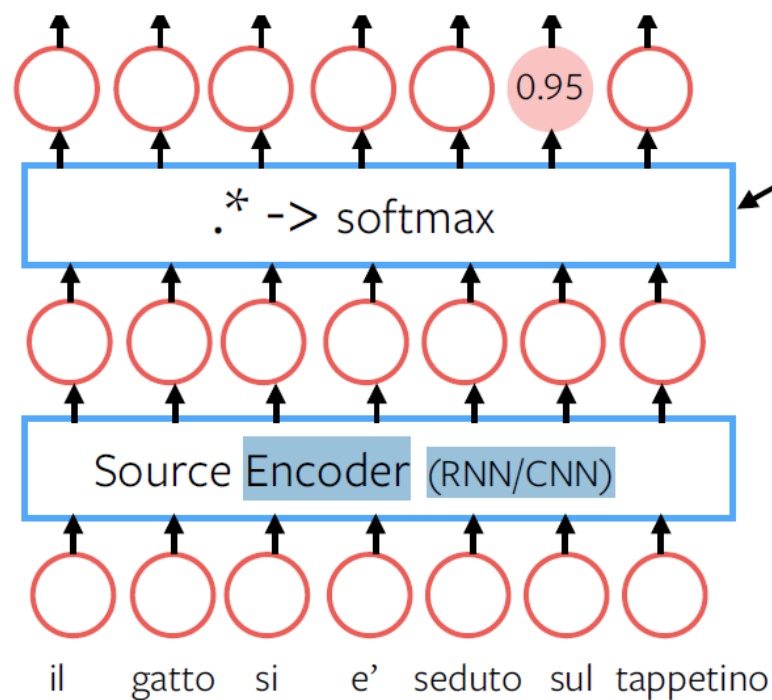
Y. LeCun's diagram++

# Последовательность -> Последовательность

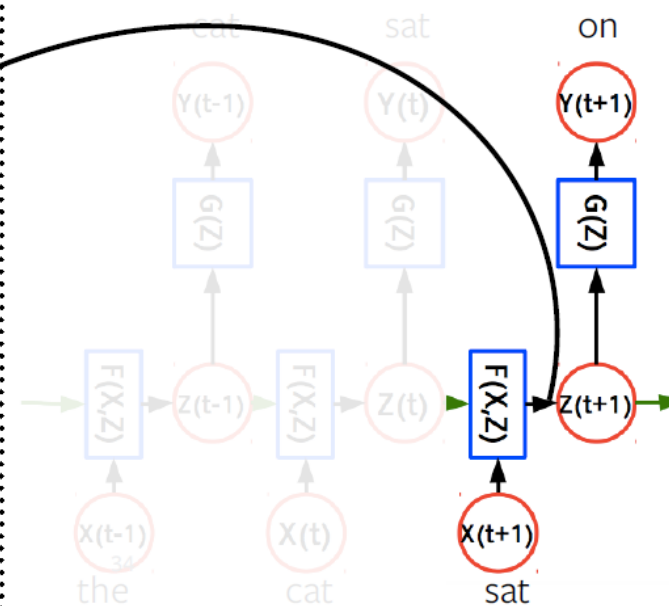
## Машинный перевод

Source

2) score each source word (attention)



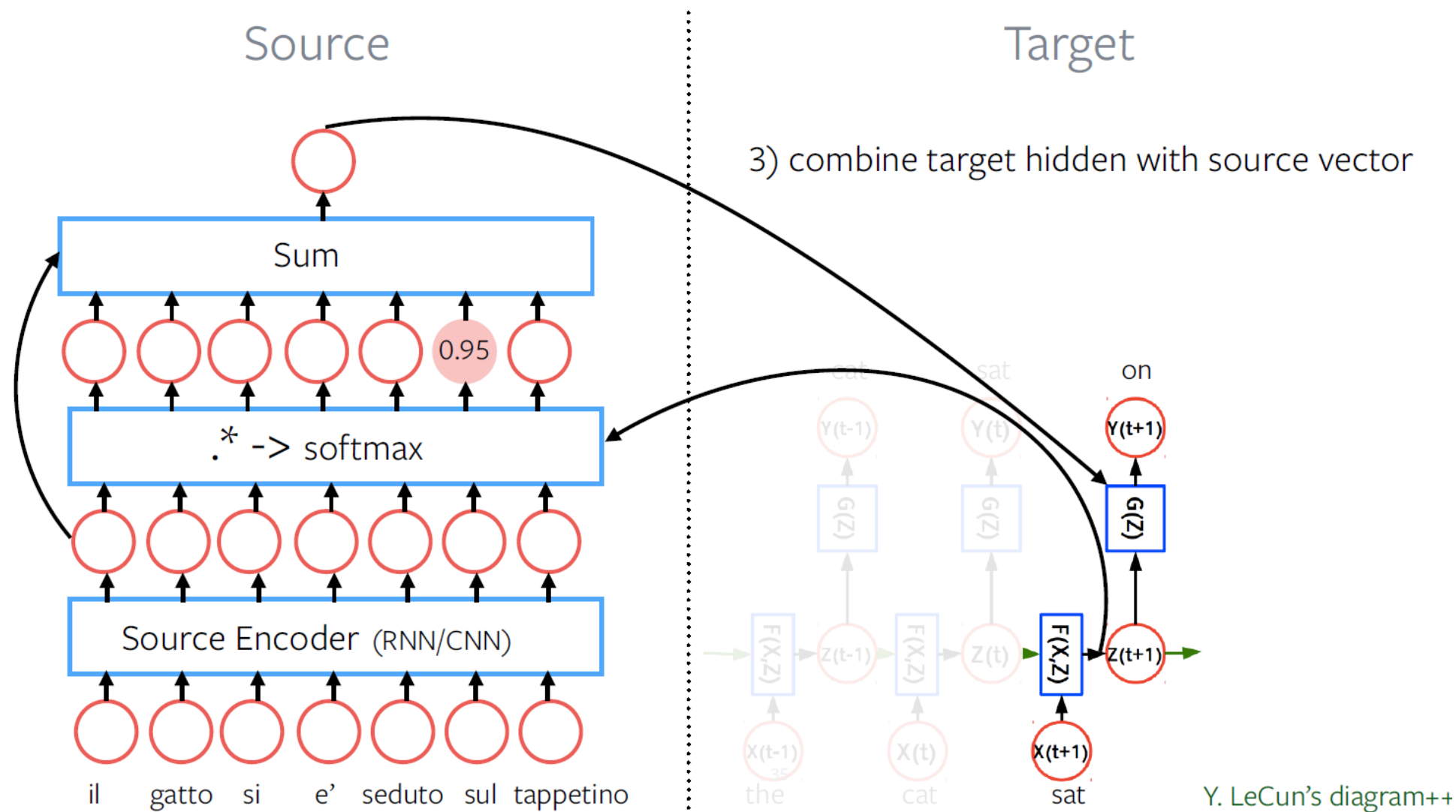
Target



Y. LeCun's diagram++

# Последовательность -> Последовательность

## Машинный перевод





# Последовательность -> Последовательность

## Машинный перевод

### Пример:

ITA: Il gatto si e' seduto sul tappetino.



EN: The cat sat on the mat.

### Заметки:

- + Исходное и целевое предложение может иметь любую длину, хорошо работает и при длинных предложениях!
- + Учится неявно выравнивать
- + RNN можно заменить на CHC. *A convolutional encoder model for NMT, Gehring et al. 2016*
- + Генерирует беглые предложения
- Имеет проблемы с редко встречаемыми словами, точным выбором слов
- Обычно обучается как языковая модель (кросс-энтропия), хороша для вычисления оценки, но не для генерации

# Последовательность -> Последовательность

## Машинный перевод

### Заключение:

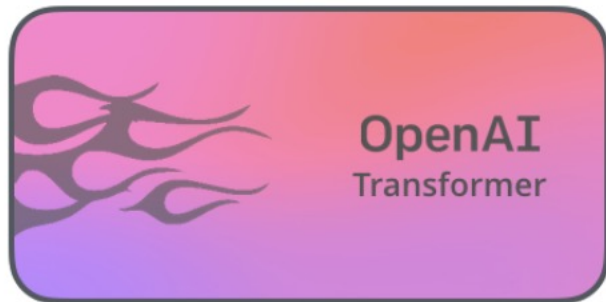
- + Механизм внимания является довольно общим и может использоваться для:
  - + Работает с входами переменной длины, поскольку «мягко выбирает один»
  - + Неявное выравнивание, которое обучается моделью, по мере необходимости
  - + Для выполнения раундов «объяснений» (например, «переходов» в сетях памяти)
- + Этот же механизм использовался для создания субтитров, суммирования и т. Д.
- Функция потерь на уровне слов (кросс-энтропия для предсказания следующего слова) является субоптимальной для задач генерации текстов

Sequence level training with RNNs, Ranzato et al. ICLR 2016

An actor-critic algorithm for sequence prediction, ICLR 2017

Sequence-to-sequence learning as beam-search optimization, EMNLP 2016

# Attention is all you need

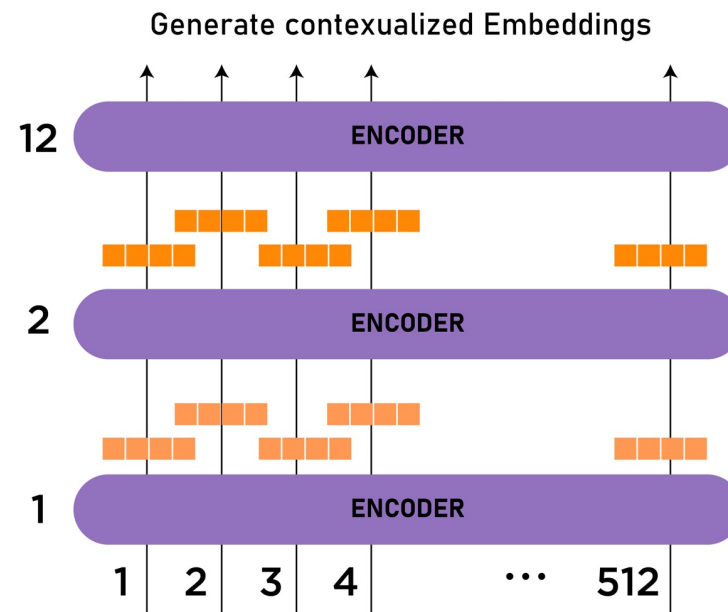
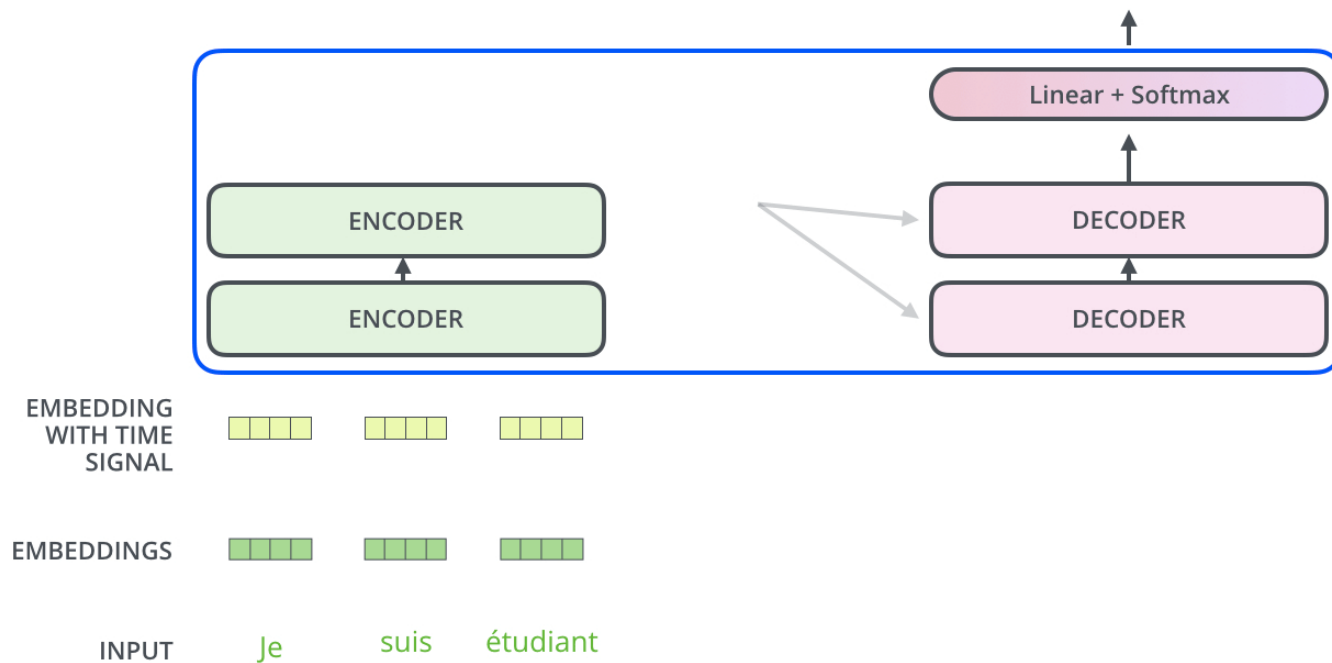


# Трансформер

- Трансформер с механизмом ВНИМАНИЯ
- BERT, GPT

Decoding time step: 1 2 3 4 5 6

OUTPUT

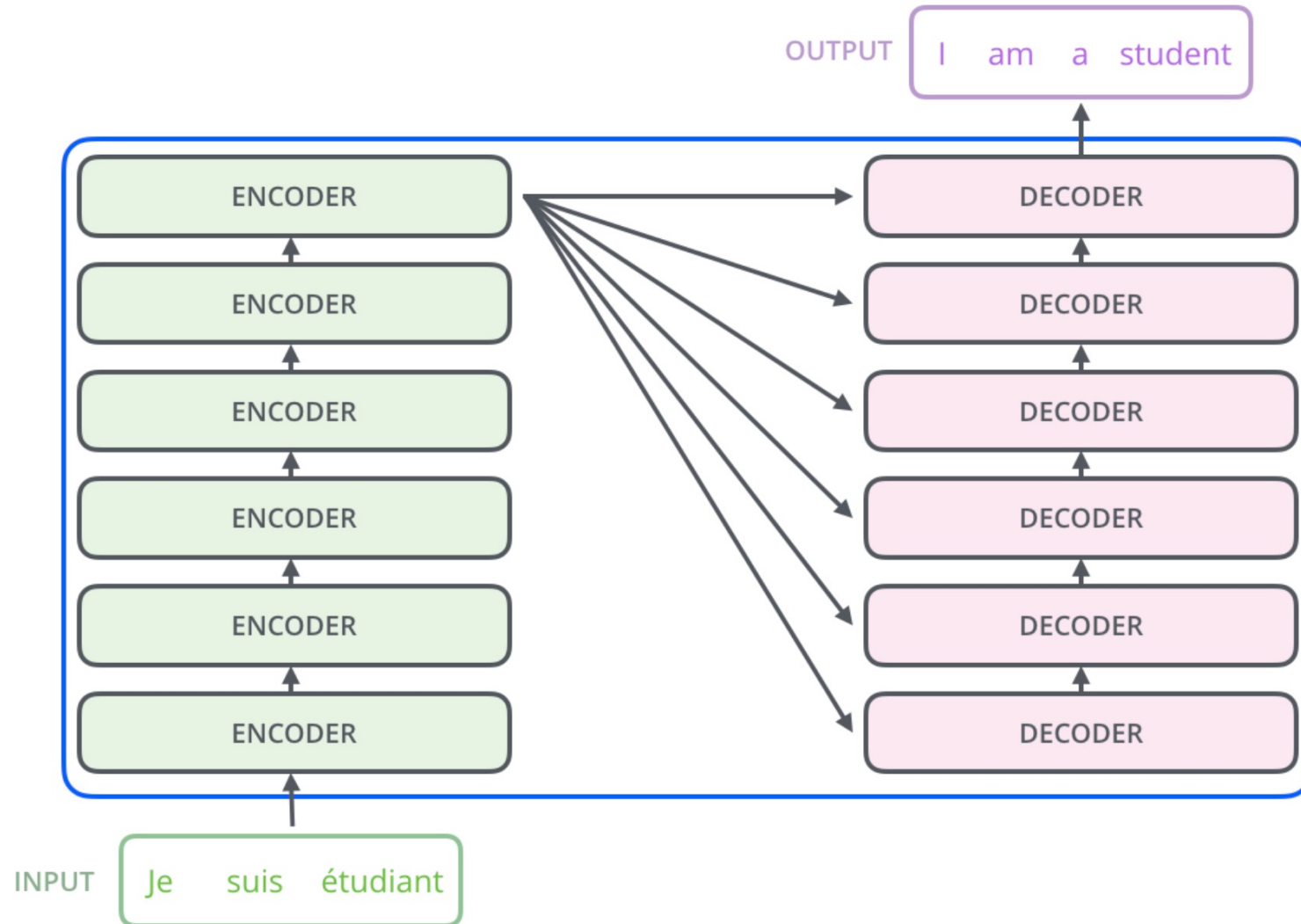


**BERT**

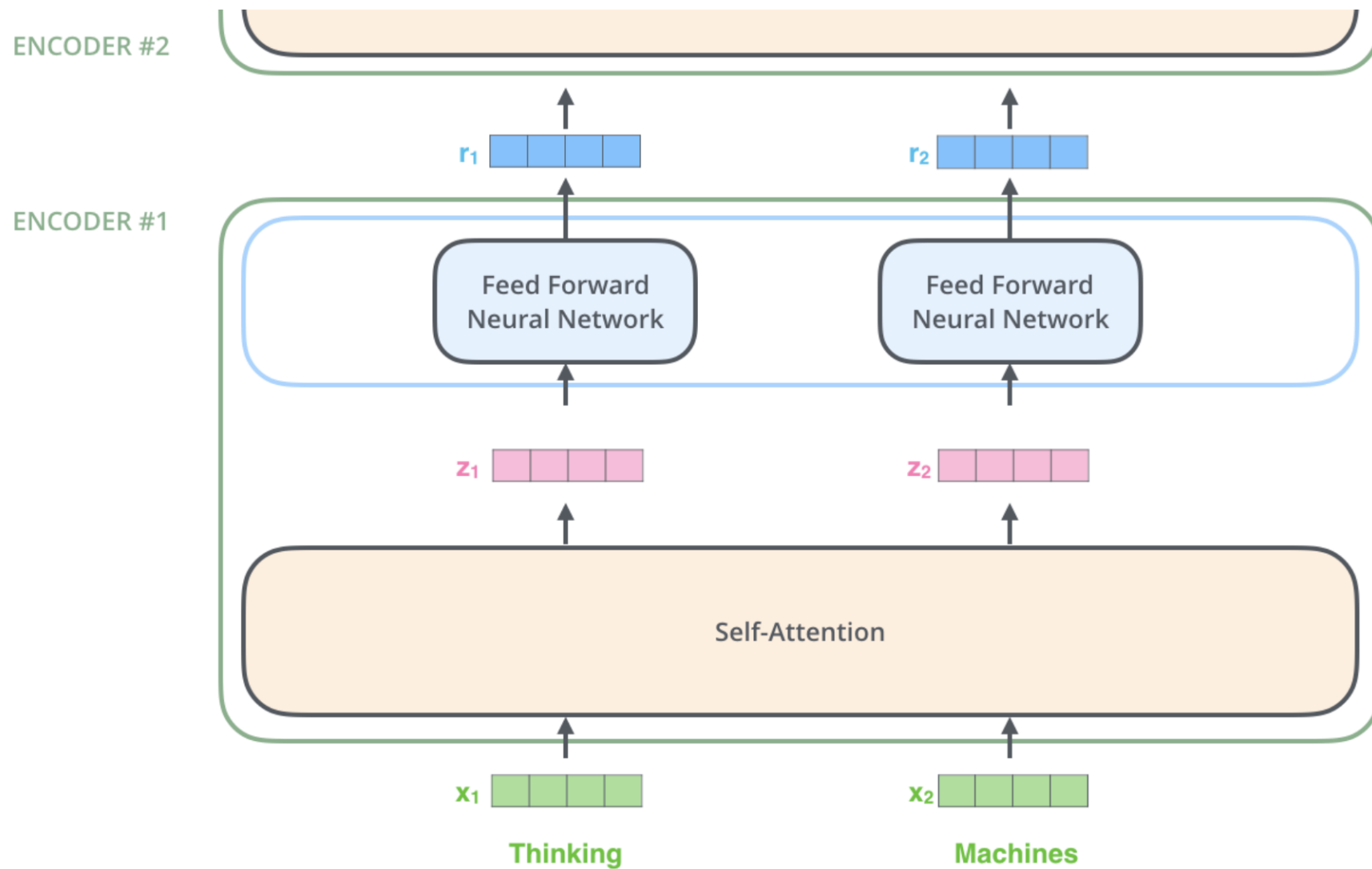
The output of each encoder layer can be used to represent the feature for that token



# Архитектура Transformer



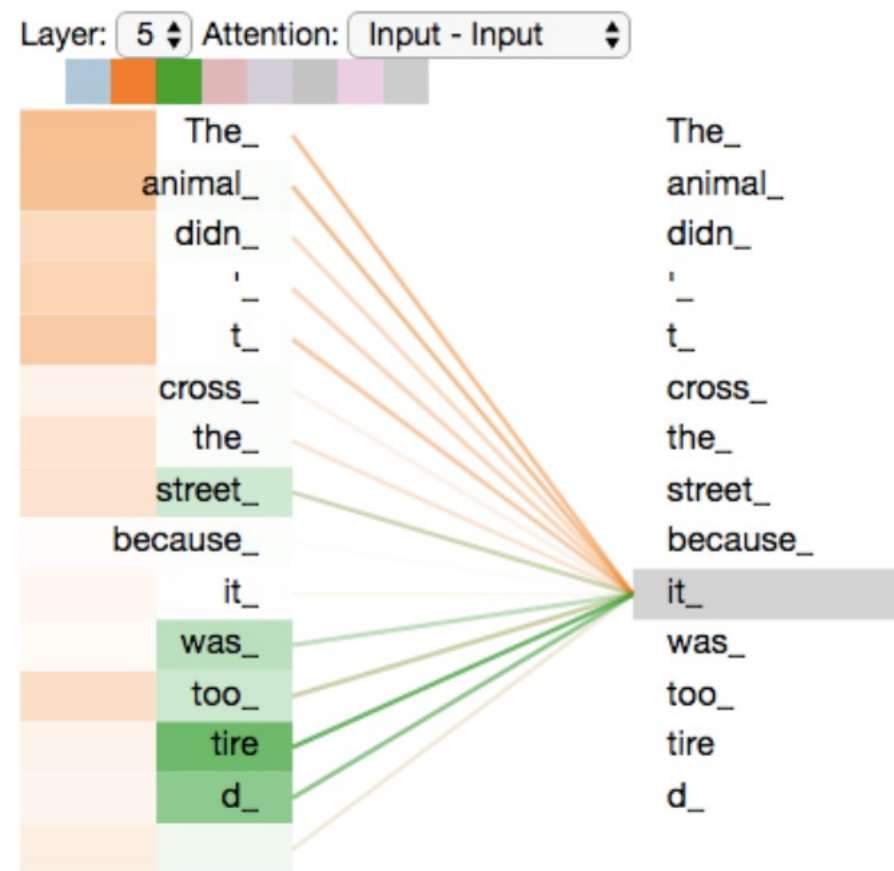
# Кодировщик



# Внутреннее внимание

As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" -- in a sense.

The model's representation of the word "it" bakes in some of the representation of both "animal" and "tired"



# Внутреннее внимание

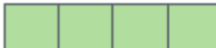
Multiplying  $x_1$  by the  $W^Q$  weight matrix produces  $q_1$ , the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

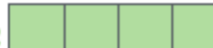
Input

Thinking

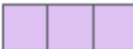
Machines

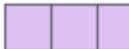
Embedding

$x_1$  

$x_2$  

Queries

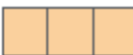
$q_1$  

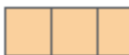
$q_2$  



$W^Q$

Keys

$k_1$  

$k_2$  



$W^K$

Values

$v_1$  

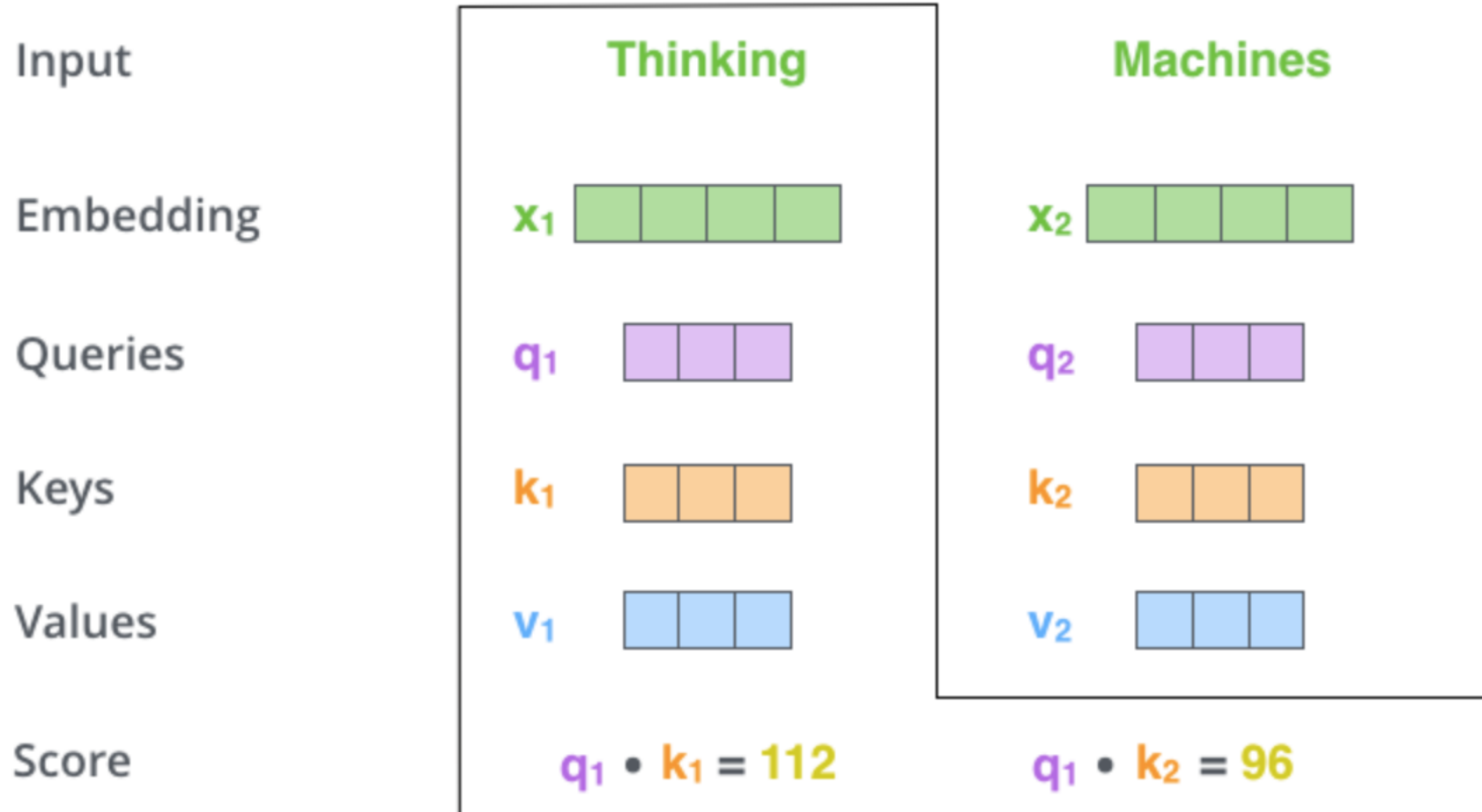
$v_2$  



$W^V$



# Query, Key, Value



# Value

Input

Embedding

Queries

Keys

Values

Score

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

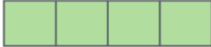
Softmax

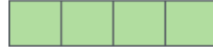
Softmax  
X  
Value

Sum

Thinking

Machines

$x_1$  

$x_2$  

$q_1$  

$q_2$  

$k_1$  

$k_2$  

$v_1$  

$v_2$  

$q_1 \cdot k_1 = 112$

$q_1 \cdot k_2 = 96$

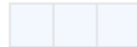
14

12

0.88

0.12

$v_1$  

$v_2$  

$z_1$  

$z_2$  

# Several words

Every row in the  $X$  matrix corresponds to a word in the input sentence



# Calculation

$$\text{softmax} \left( \frac{\begin{matrix} \text{Q} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix} \times \begin{matrix} \text{K}^T \\ \begin{array}{|c|c|} \hline & \\ \hline & \\ \hline & \\ \hline \end{array} \end{matrix} \right) \begin{matrix} \text{V} \\ \begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array} \end{matrix}$$

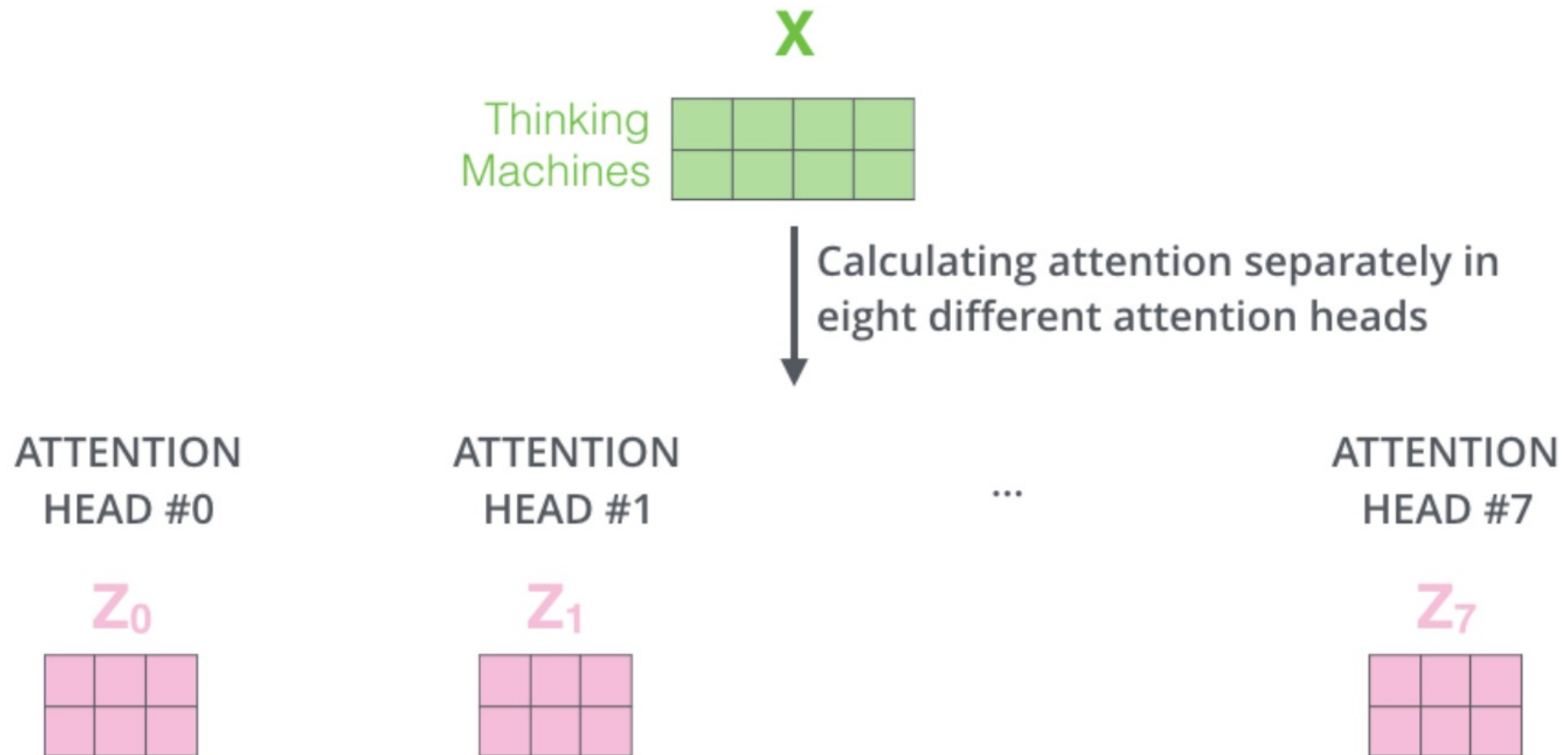
=

$\text{Z}$

$\begin{array}{|c|c|c|} \hline & & \\ \hline & & \\ \hline \end{array}$

The self-attention calculation in matrix form

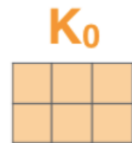
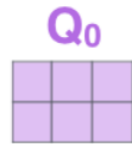
# Attention heads



# Attention heads

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

ATTENTION HEAD #0



$W_0^Q$



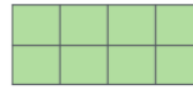
$W_0^K$



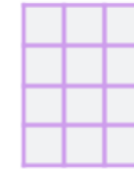
$W_0^V$

$X$

Thinking  
Machines



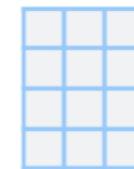
ATTENTION HEAD #1



$W_1^Q$



$W_1^K$



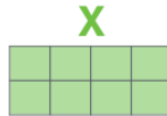
$W_1^V$

# Attention result

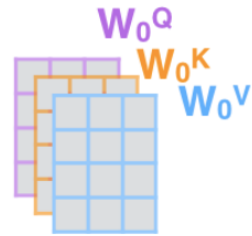
1) This is our input sentence\*

Thinking  
Machines

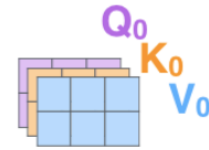
2) We embed each word\*



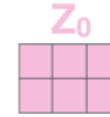
3) Split into 8 heads.  
We multiply  $X$  or  $R$  with weight matrices



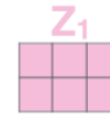
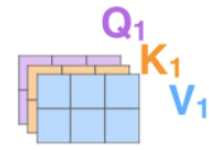
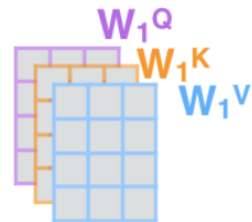
4) Calculate attention using the resulting  $Q/K/V$  matrices



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



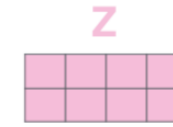
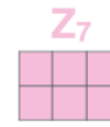
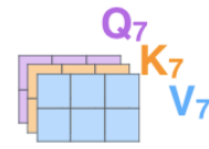
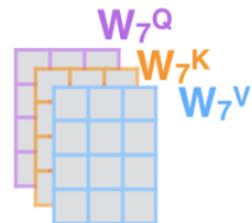
\* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



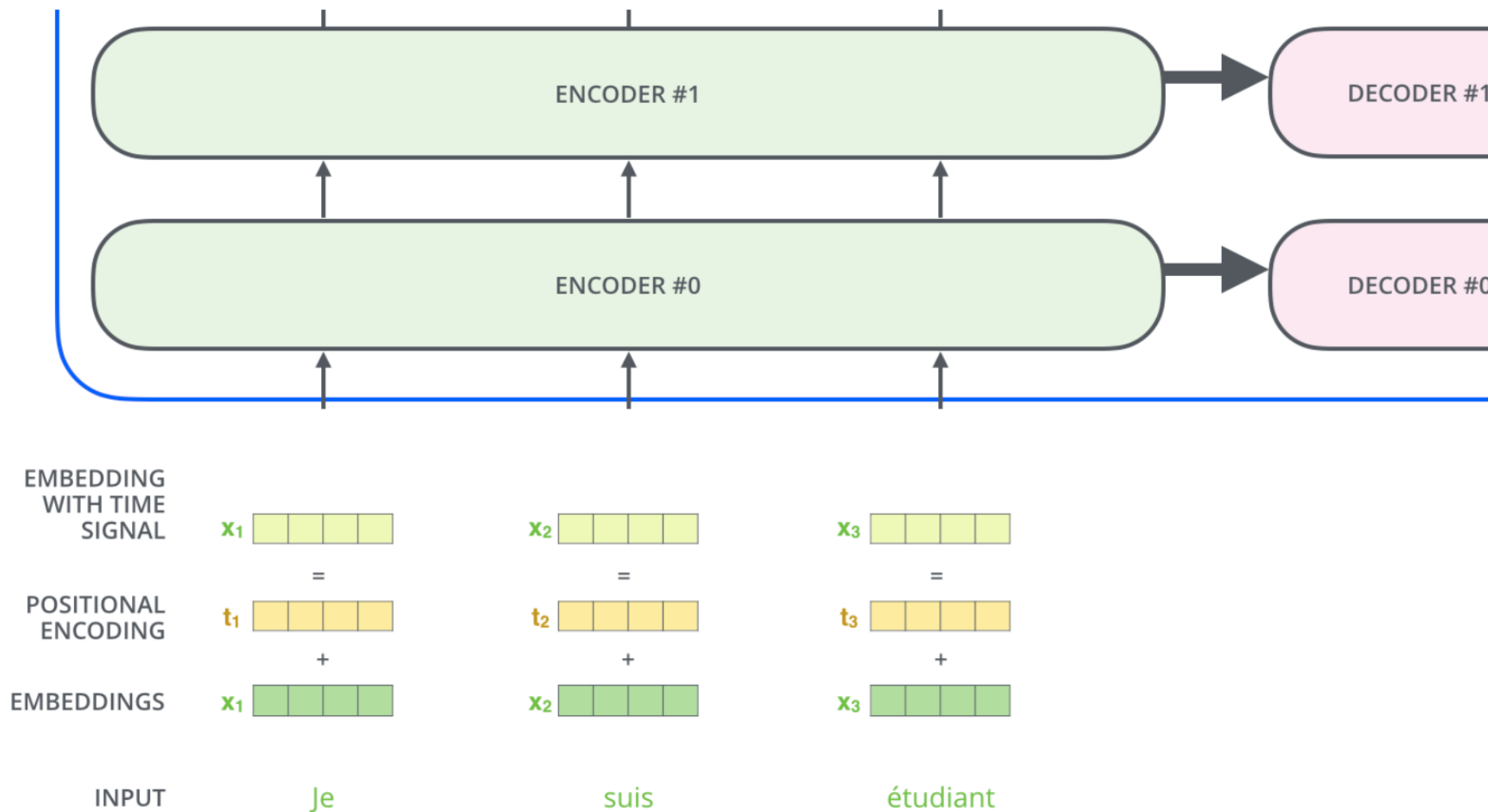
...

...

...



# Positions



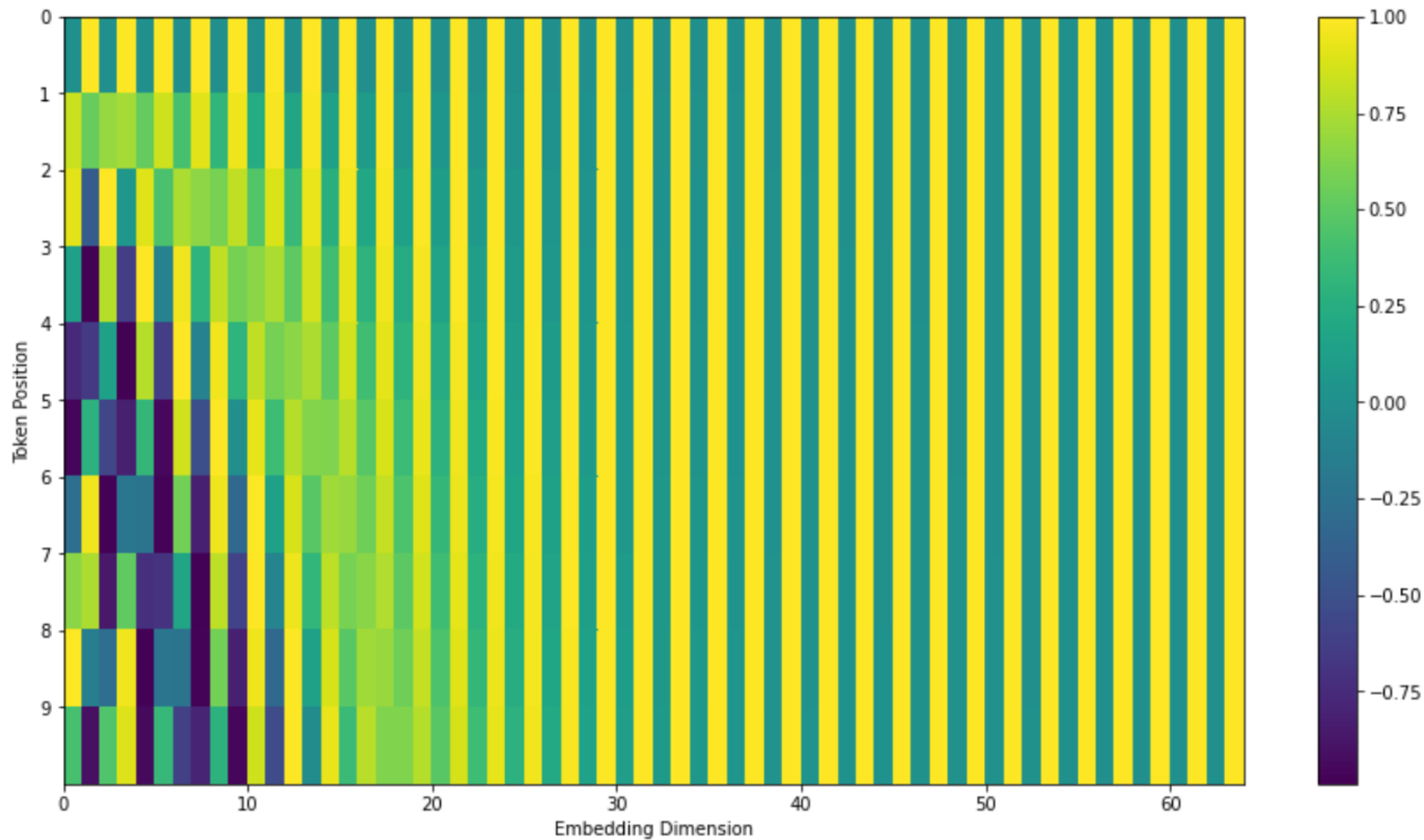


# Positional encoding

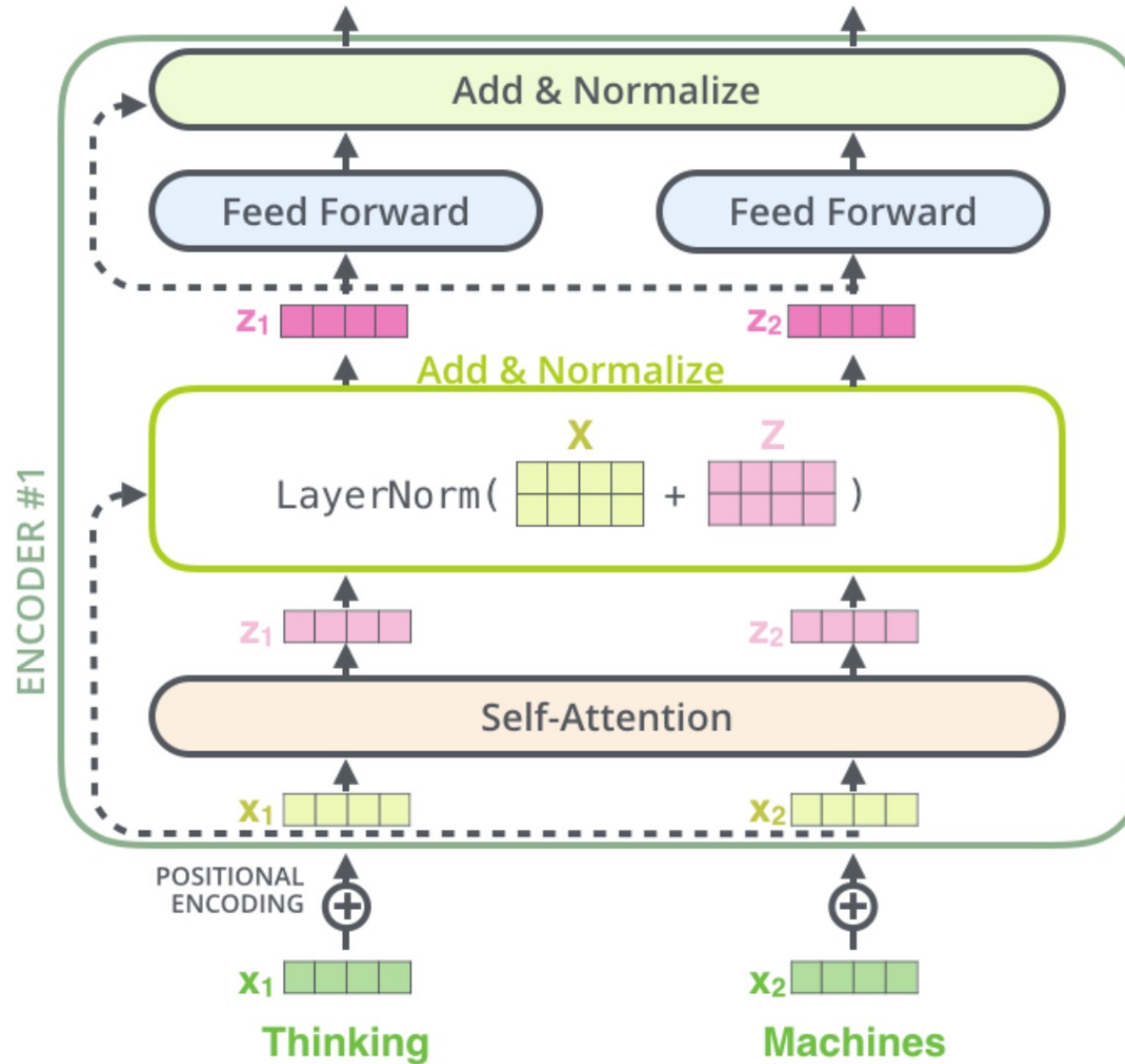
A real example of positional encoding  
with a toy embedding size of 4



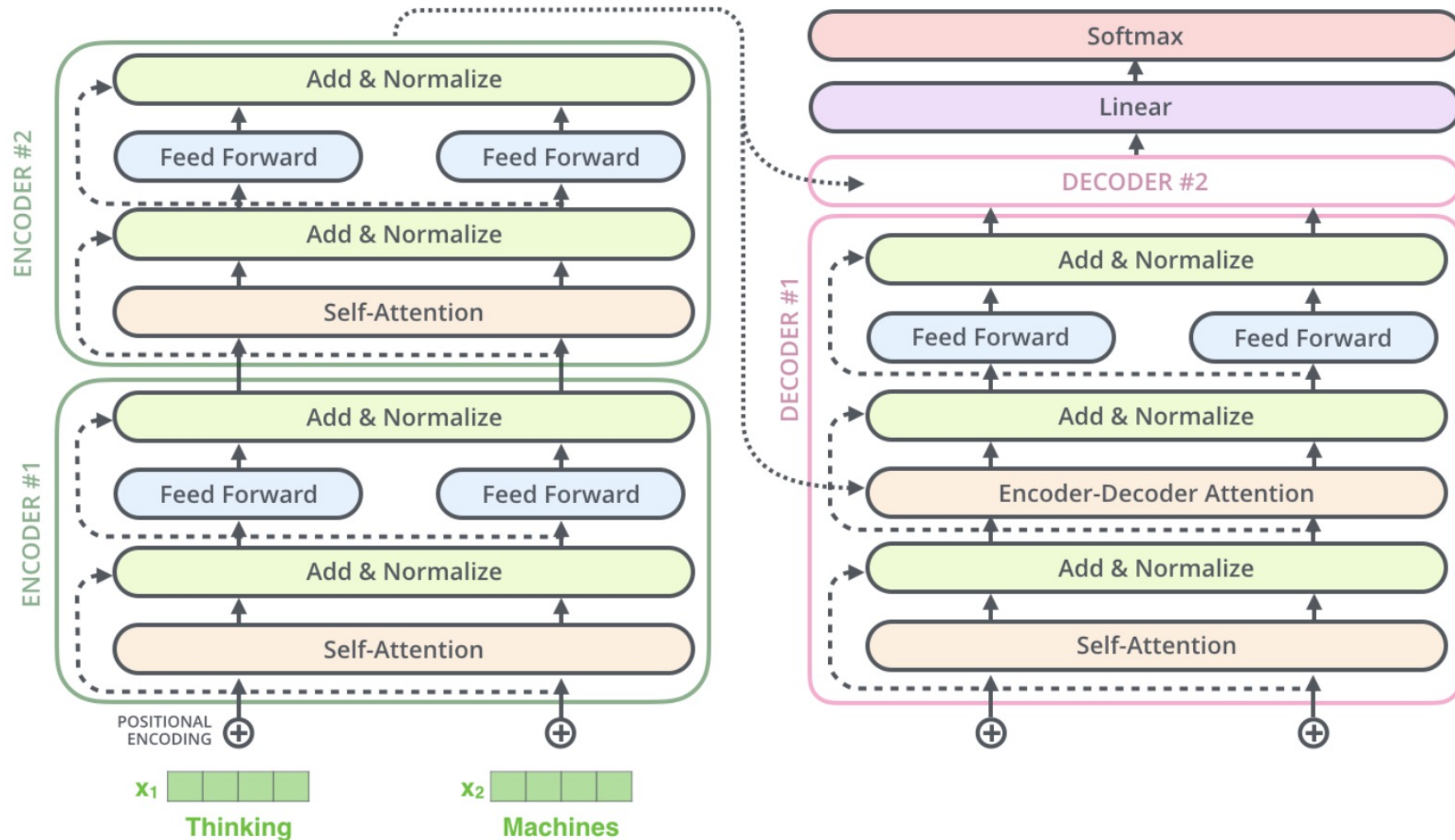
# Positional encoding



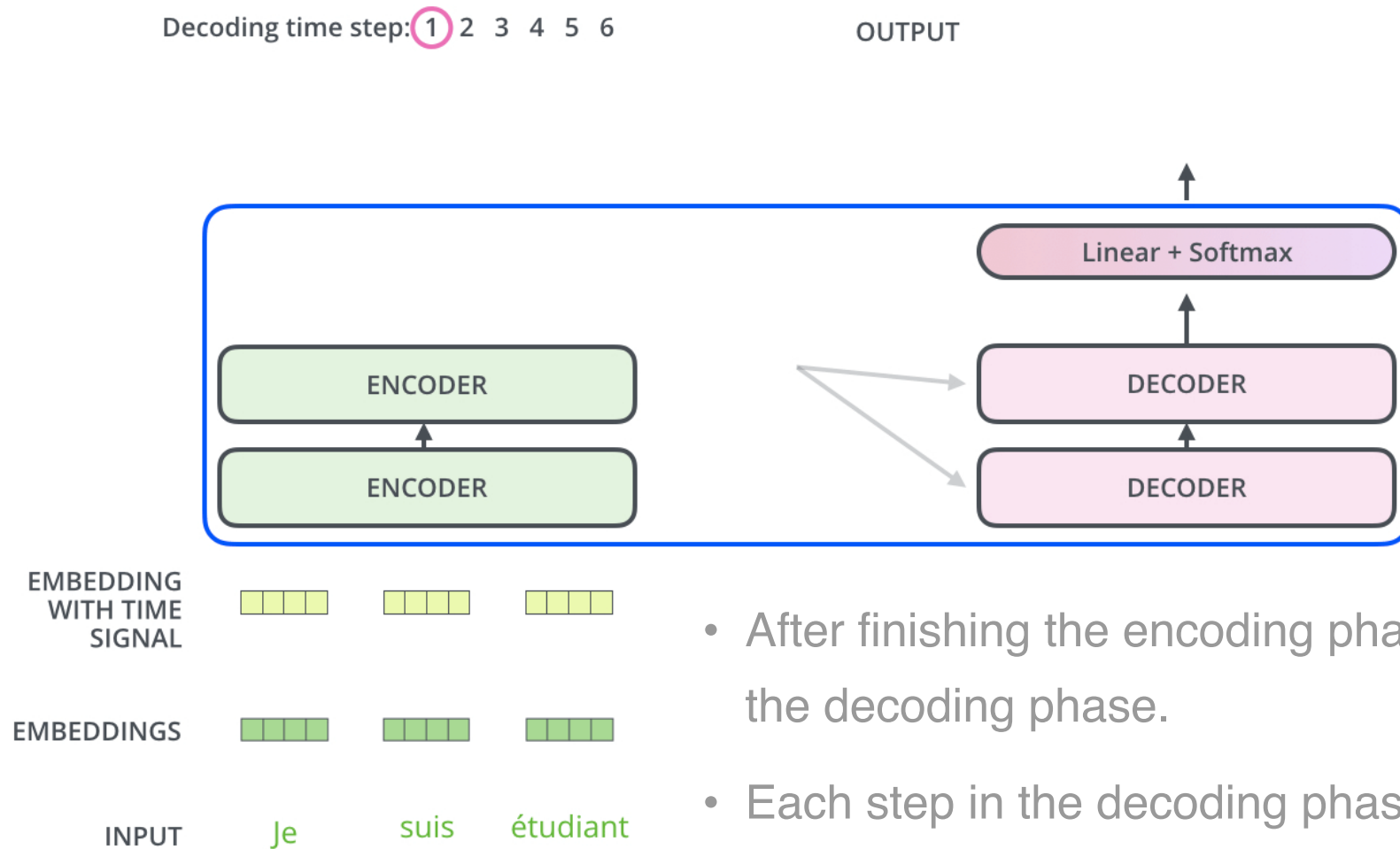
# Add and Normalize



# Encoder-decoder connection



# Decoder



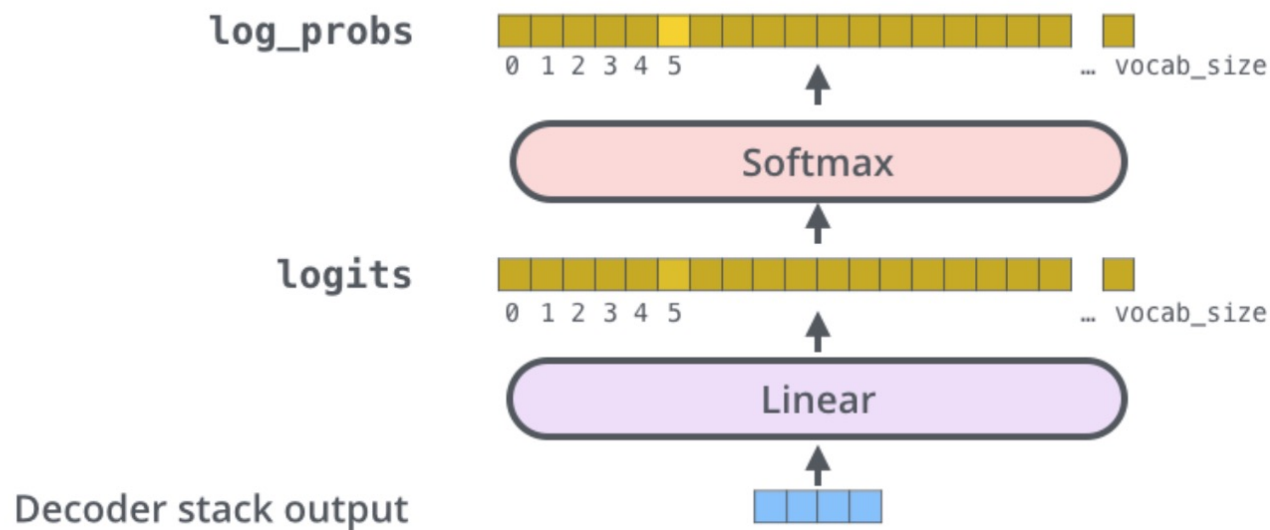
- After finishing the encoding phase, we begin the decoding phase.
- Each step in the decoding phase outputs an element from the output sequence (the English translation sentence in this case)

# ИТОГОВОЕ СЛОВО

- Let's assume that our model knows 10,000 unique English words (our model's "output vocabulary") that it's learned from its training dataset.
- This would make the logits vector 10,000 cells wide – each cell corresponding to the score of a unique word.
- That is how we interpret the output of the model followed by the Linear layer.

Which word in our vocabulary  
is associated with this index?

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



# Example

Output Vocabulary						
WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

One-hot encoding of the word "am"

0.0	1.0	0.0	0.0	0.0	0.0
-----	-----	-----	-----	-----	-----

Example: one-hot encoding of our output vocabulary

# Model output

