

# Глубокое обучение

Бекезин Никита

4 июня 2022

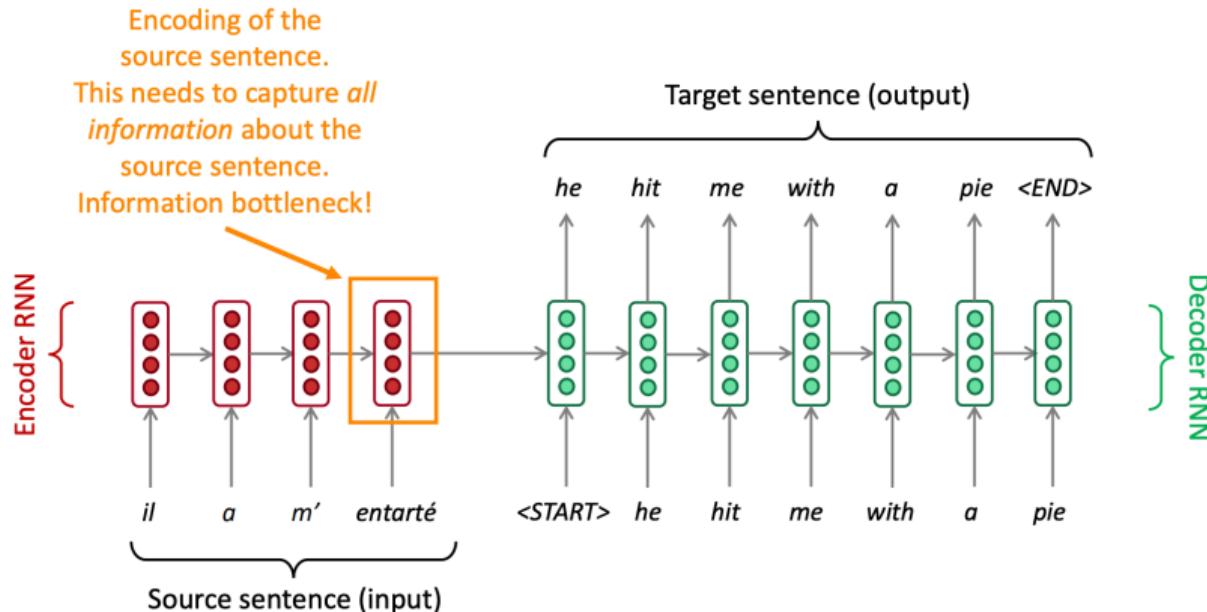
**Лекция 9:** Быстрое введение в Transformer + BERT

# Agenda

- Seq2seq recap
- Attention recap
- Self-Attention -> Transformer
- BERT

# Seq2seq recap

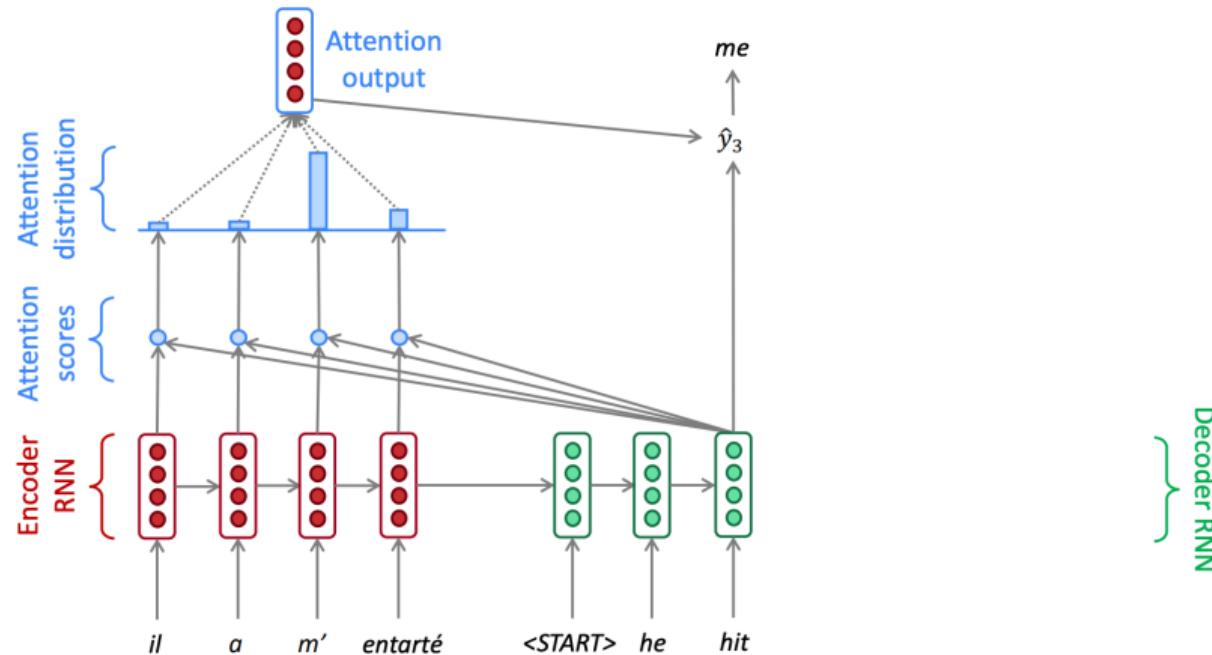
## Sequence-to-sequence: the bottleneck problem



Источник: <https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf>

# Attention recap

## Sequence-to-sequence with attention



Источник: <https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf>

# Attention recap

There are **several ways** you can compute  $e \in \mathbb{R}^N$  from  $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^{d_1}$  and  $s \in \mathbb{R}^{d_2}$ :

- **Basic dot-product attention:**  $e_i = s^T \mathbf{h}_i \in \mathbb{R}$ 
  - Note: this assumes  $d_1 = d_2$
  - This is the version we saw earlier
- **Multiplicative attention:**  $e_i = s^T \mathbf{W} \mathbf{h}_i \in \mathbb{R}$ 
  - Where  $\mathbf{W} \in \mathbb{R}^{d_2 \times d_1}$  is a weight matrix
- **Additive attention:**  $e_i = \mathbf{v}^T \tanh(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 s) \in \mathbb{R}$ 
  - Where  $\mathbf{W}_1 \in \mathbb{R}^{d_3 \times d_1}$ ,  $\mathbf{W}_2 \in \mathbb{R}^{d_3 \times d_2}$  are weight matrices and  $\mathbf{v} \in \mathbb{R}^{d_3}$  is a weight vector.
  - $d_3$  (the attention dimensionality) is a hyperparameter

Источник: <https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture08-nmt.pdf>

# Attention recap

- “Free” word alignment
- Better results on long sequences with attention

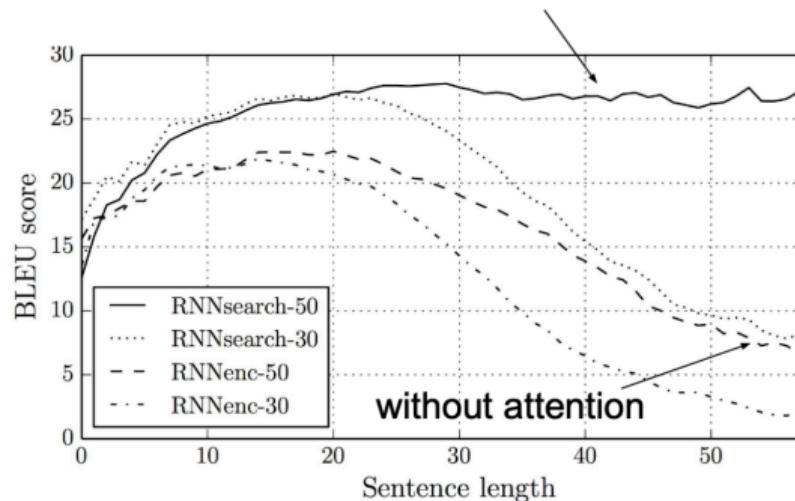
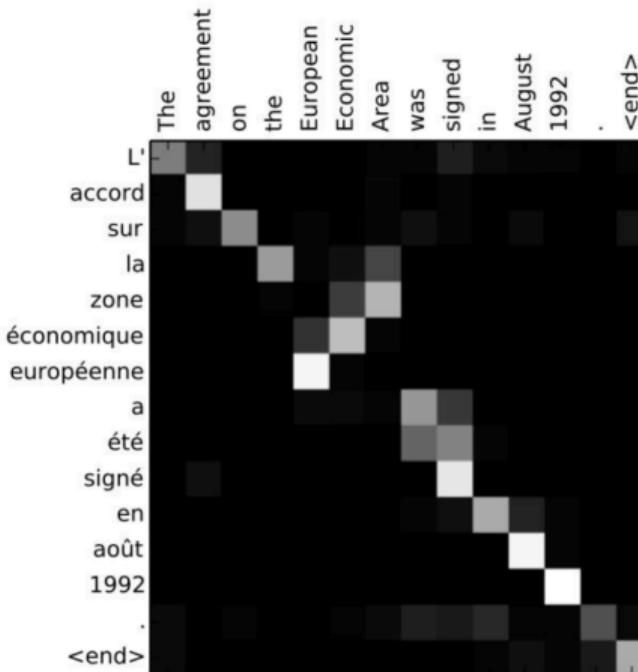


Image source: [Neural Machine Translation by Jointly Learning to Align and Translate](#)

## Attention advantages



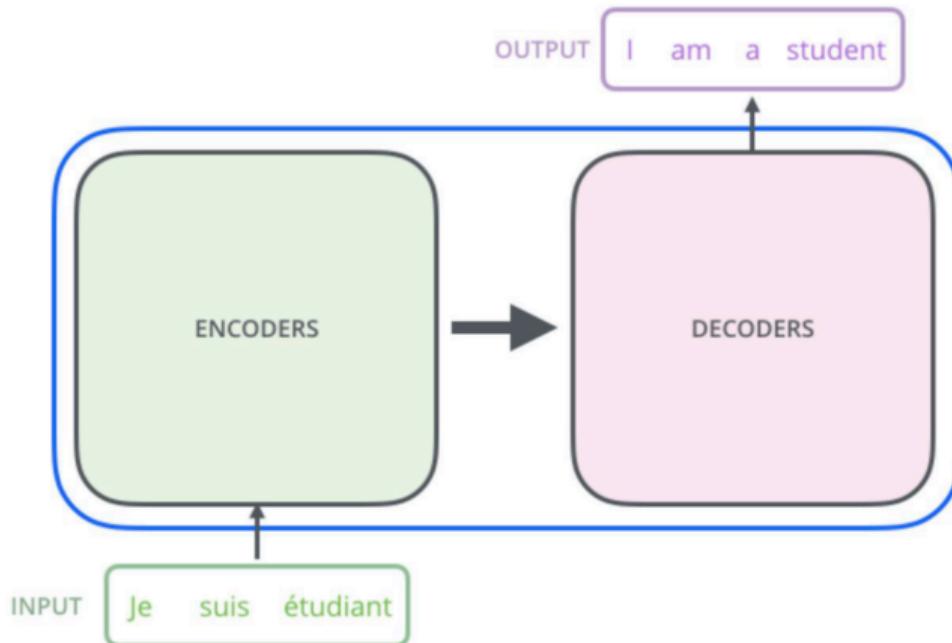
Attention is all you need!

# Attention is all you need

Данная статья - развитие идеи внимания. Статья вышла в 2017 году и стала истоком всех текущих SOTA моделей.

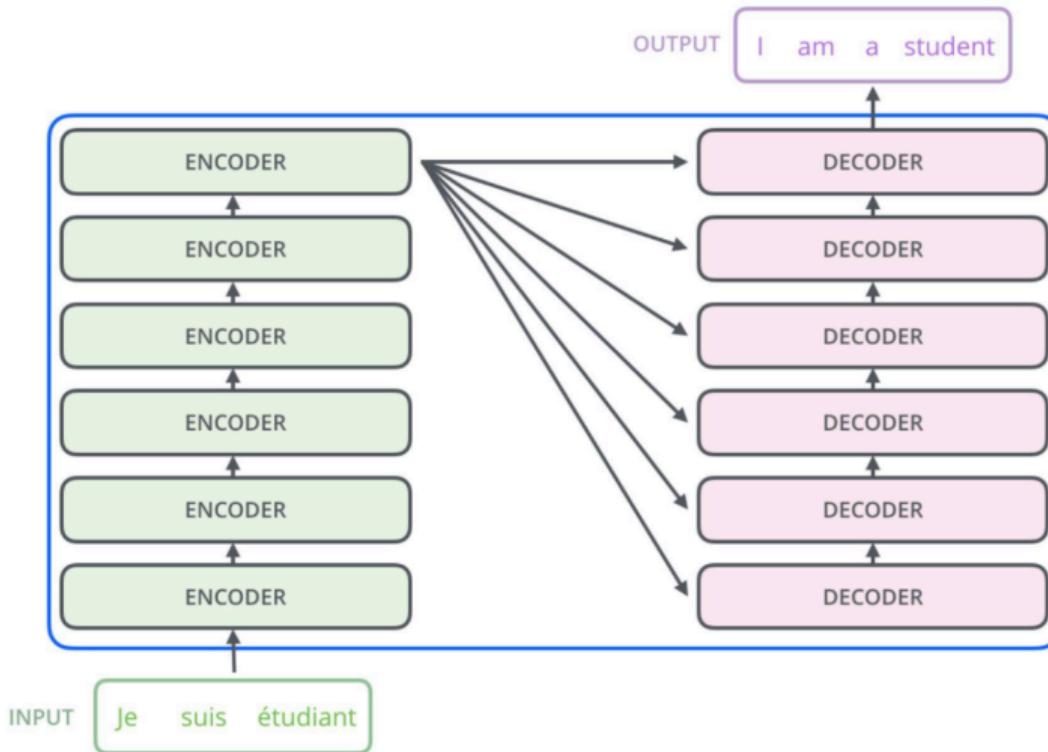
А зачем нам вообще что-то, кроме внимания? Идея в том, чтобы добавить в энкодер и декодер как можно больше внимания, что позволит уйти от рекуррентной структуры в Encoder-Decoder архитектуре

# Transformer



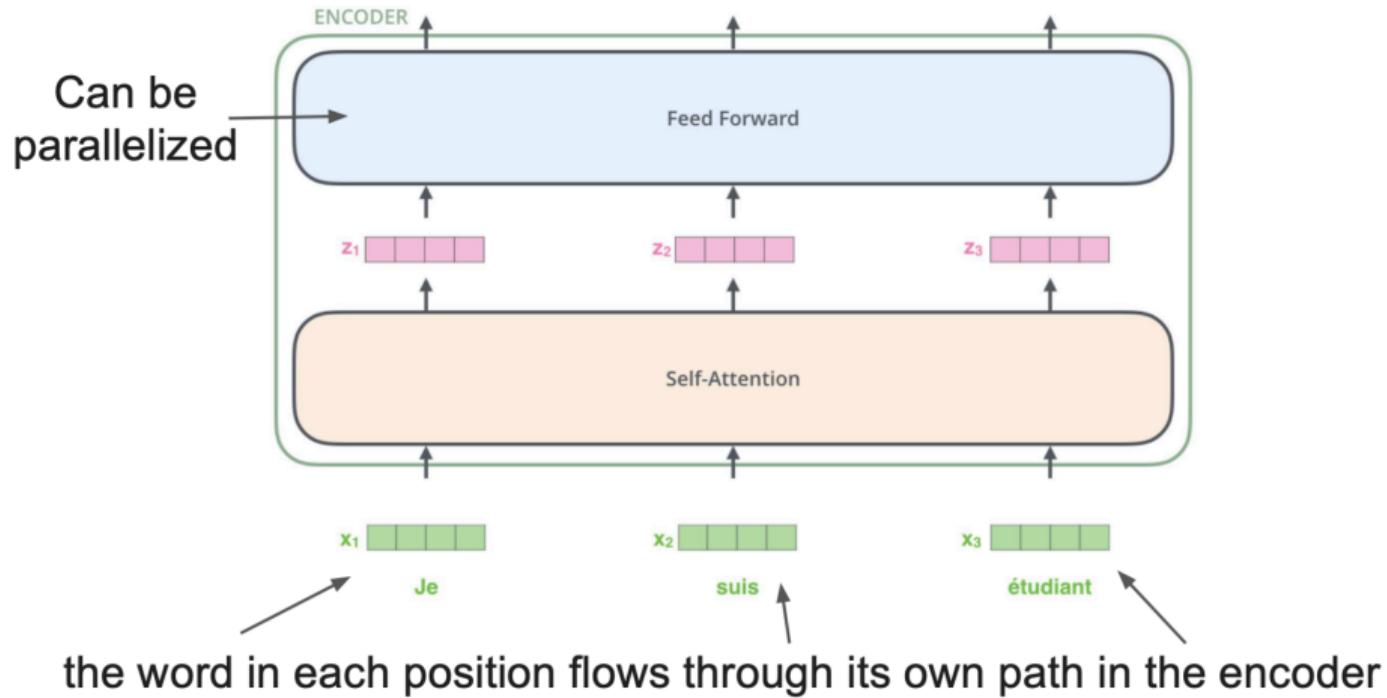
Источник: <https://jalammar.github.io/illustrated-transformer/>

# Transformer



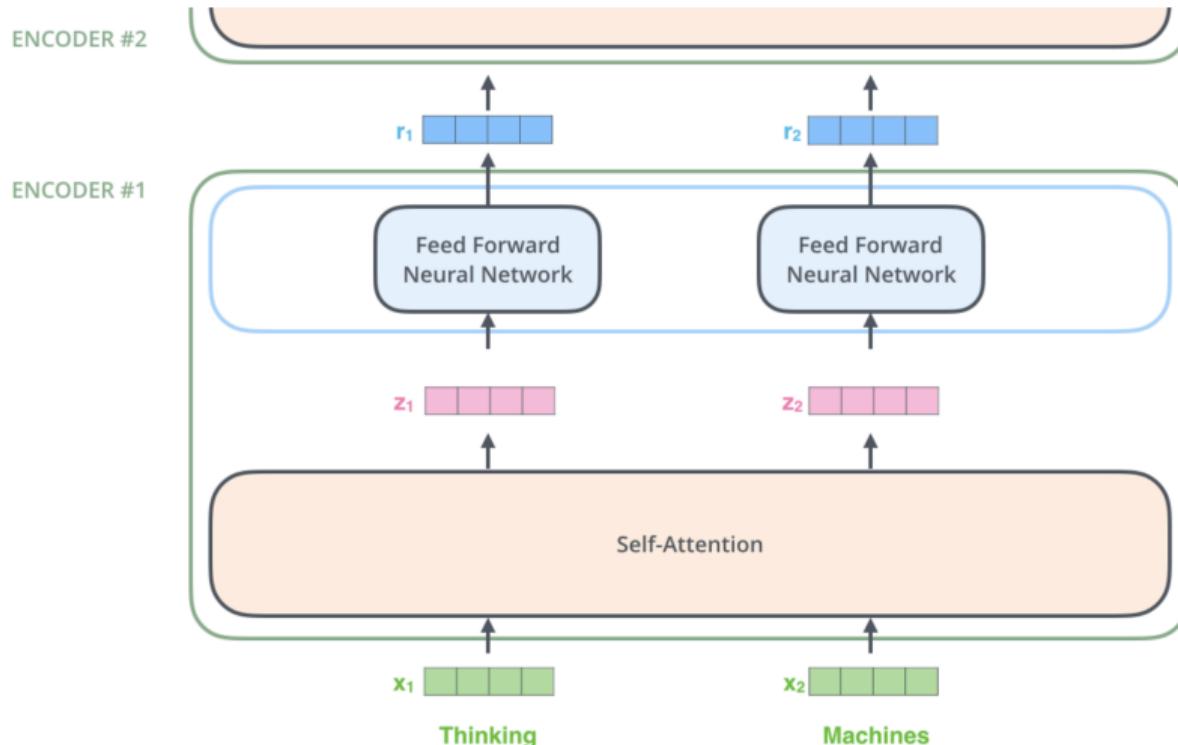
Источник: <https://jalammar.github.io/illustrated-transformer/>

# Transformer



Источник: <https://jalammar.github.io/illustrated-transformer/>

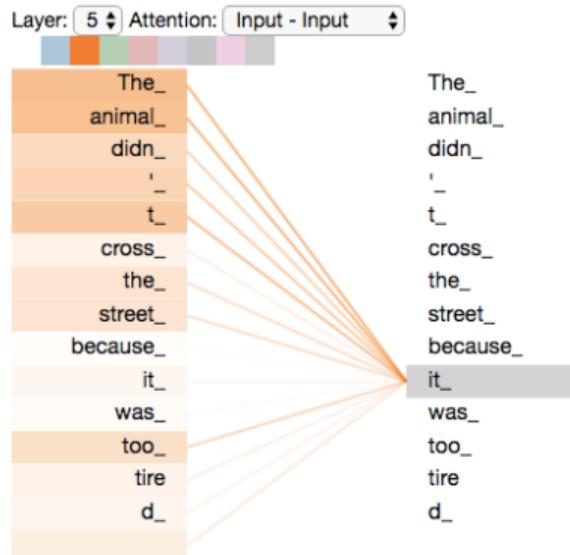
# Transformer



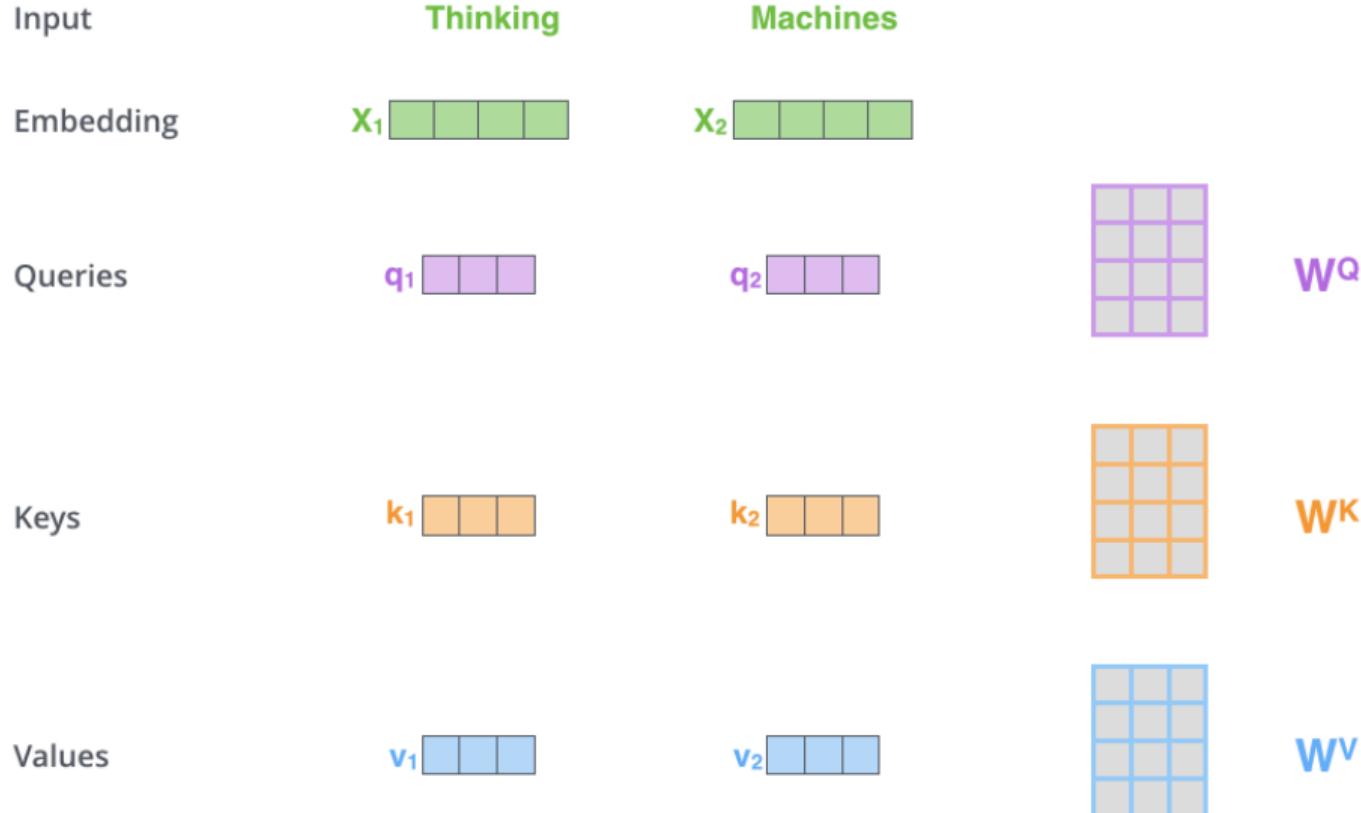
Источник: <https://jalammar.github.io/illustrated-transformer/>

# Self-attention или механизм внимания

Есть предложение: "The animal didn't cross the street because it was too tired"



# Шаг 1. Абстракции

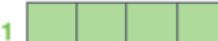
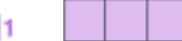
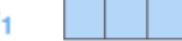


# Self-attention

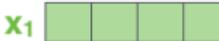
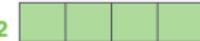
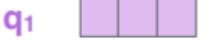
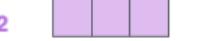
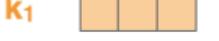
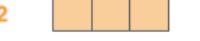
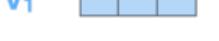
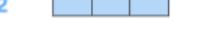
То же самое словами:

1. Query, key ищут связь между словами. Query - мое текущее слово, key - мое слово с которым я сравниваю себя.
2. Value - то, что мы знаем об этом слове

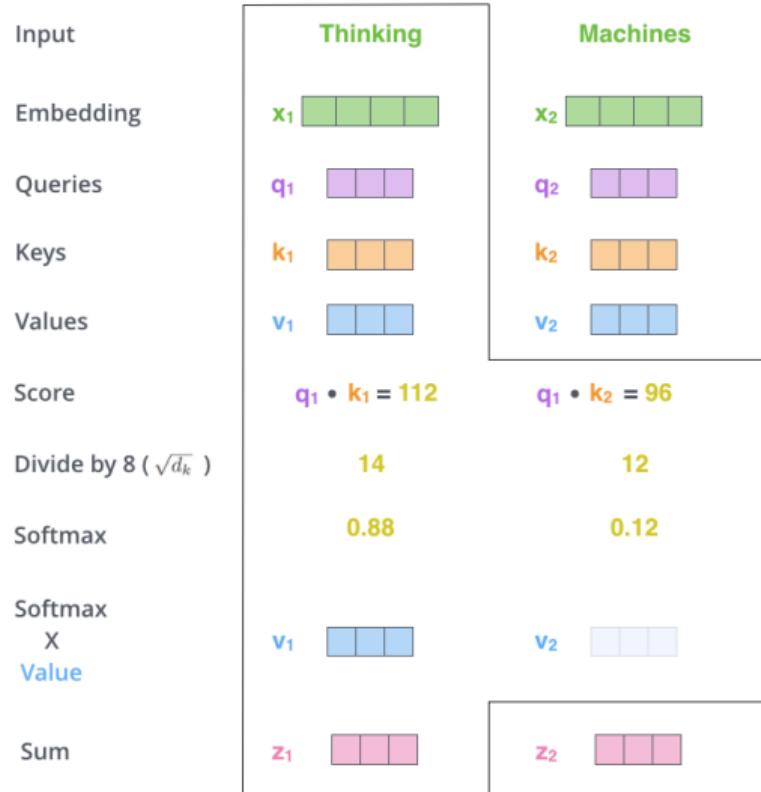
## Шаг 2. Расчет attention score

Input		
Embedding	$x_1$	
Queries	$q_1$	
Keys	$k_1$	
Values	$v_1$	
Score	$q_1 \cdot k_1 = 112$	$q_1 \cdot k_2 = 96$

## Шаг 3. Шкалирование и Softmax

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	

## Шаг 4. Сумма взвешенных векторов Value



# Матричные перемножения

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

A diagram illustrating matrix multiplication. On the left, a green 3x4 matrix labeled 'X' is multiplied by a purple 4x4 matrix labeled 'W<sup>Q</sup>'. The result is a purple 3x3 matrix labeled 'Q'. The matrices are represented as grids of colored squares.

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

A diagram illustrating matrix multiplication. On the left, a green 3x4 matrix labeled 'X' is multiplied by an orange 4x4 matrix labeled 'W<sup>K</sup>'. The result is an orange 3x3 matrix labeled 'K'. The matrices are represented as grids of colored squares.

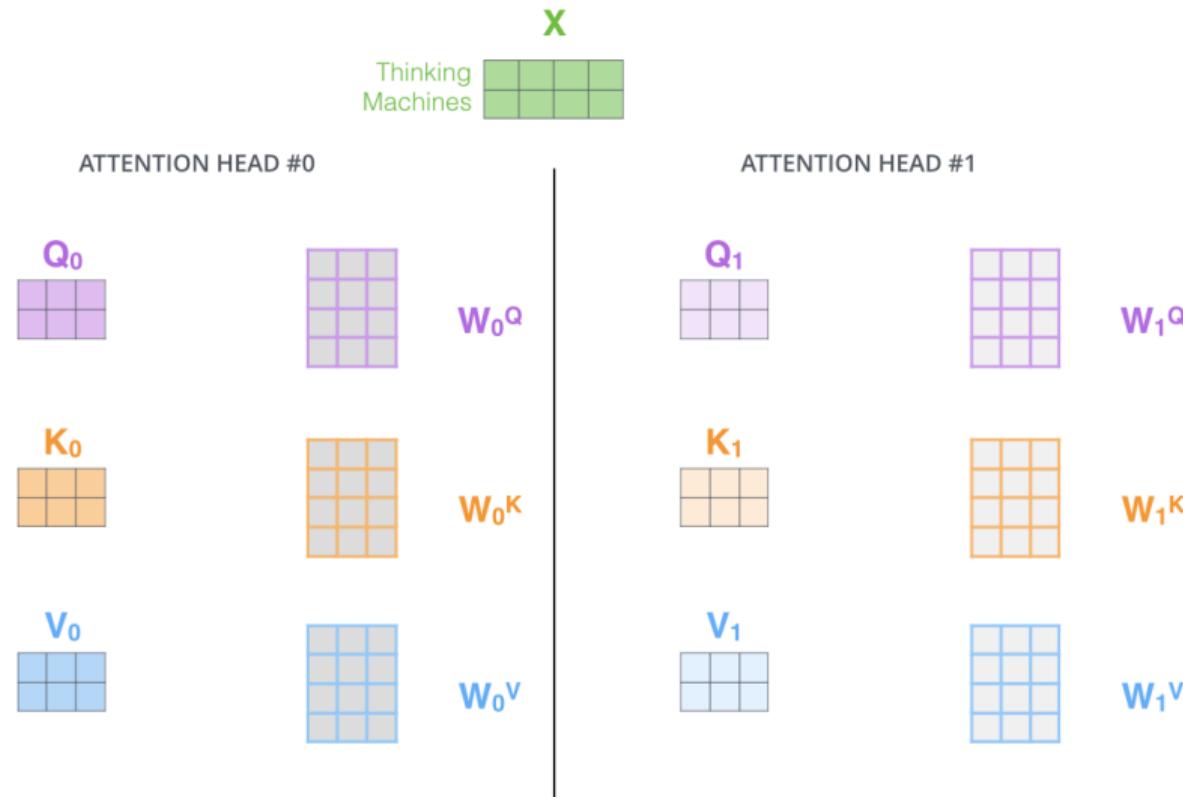
$$X \times W^V = V$$

A diagram illustrating matrix multiplication. On the left, a green 3x4 matrix labeled 'X' is multiplied by a blue 4x4 matrix labeled 'W<sup>V</sup>'. The result is a blue 3x3 matrix labeled 'V'. The matrices are represented as grids of colored squares.

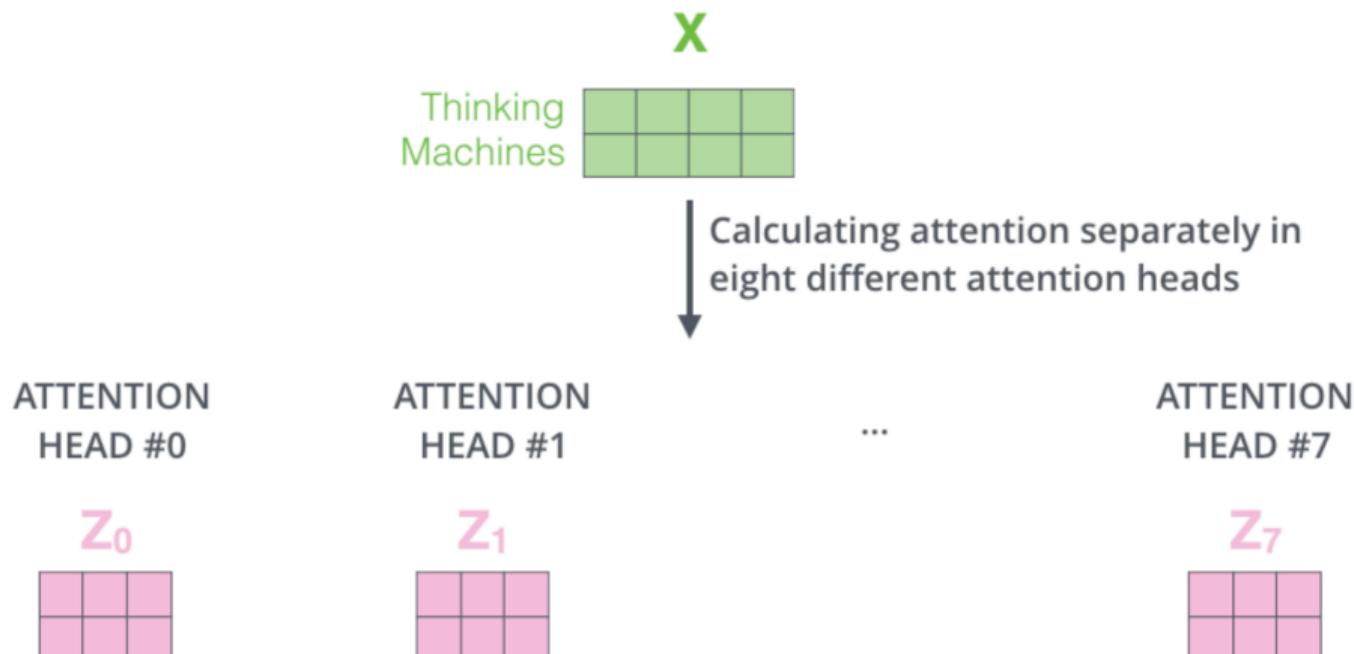
# All together

$$\text{softmax} \left( \frac{\begin{matrix} \mathbf{Q} & \times & \mathbf{K}^T \\ \begin{matrix} \text{---} \end{matrix} & \times & \begin{matrix} \text{---} \end{matrix} \end{matrix}}{\sqrt{d_k}} \right) \mathbf{V}$$
$$= \mathbf{z}$$

# Multi-Head Attention



# Multi-Head Attention



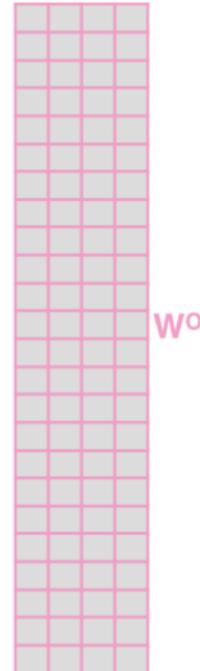
# Соединяем!

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

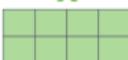
$$= \begin{matrix} Z \\ \hline \end{matrix}$$

# ИТОГО

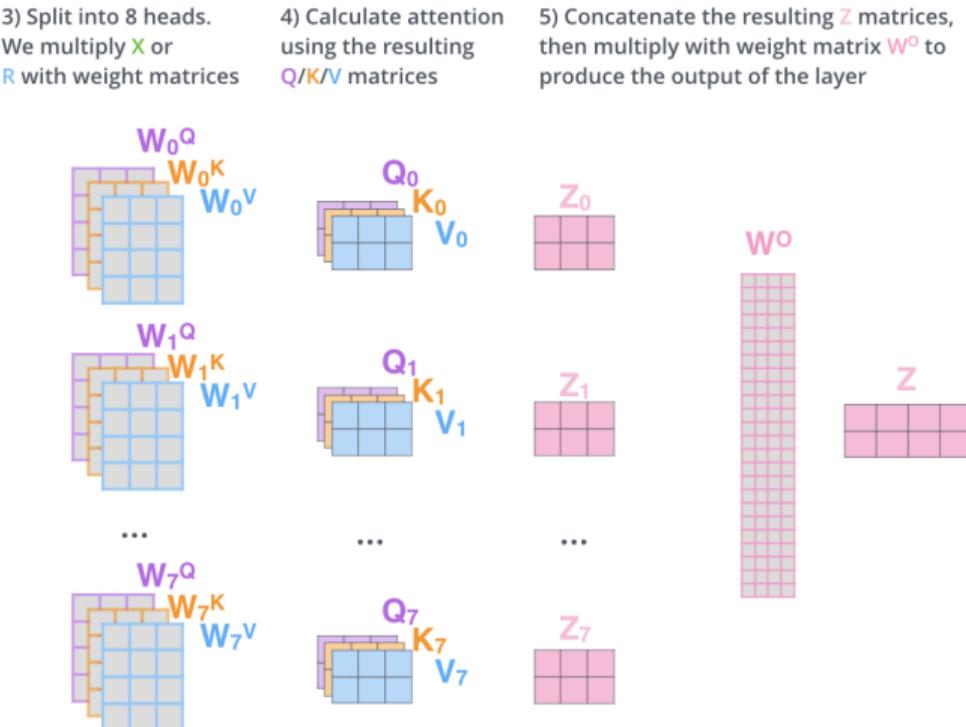
- 1) This is our input sentence\* each word\*
- 2) We embed
- 3) Split into 8 heads. We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting  $Q/K/V$  matrices
- 5) Concatenate the resulting  $Z$  matrices, then multiply with weight matrix  $W^o$  to produce the output of the layer

Thinking  
Machines

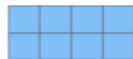
$X$



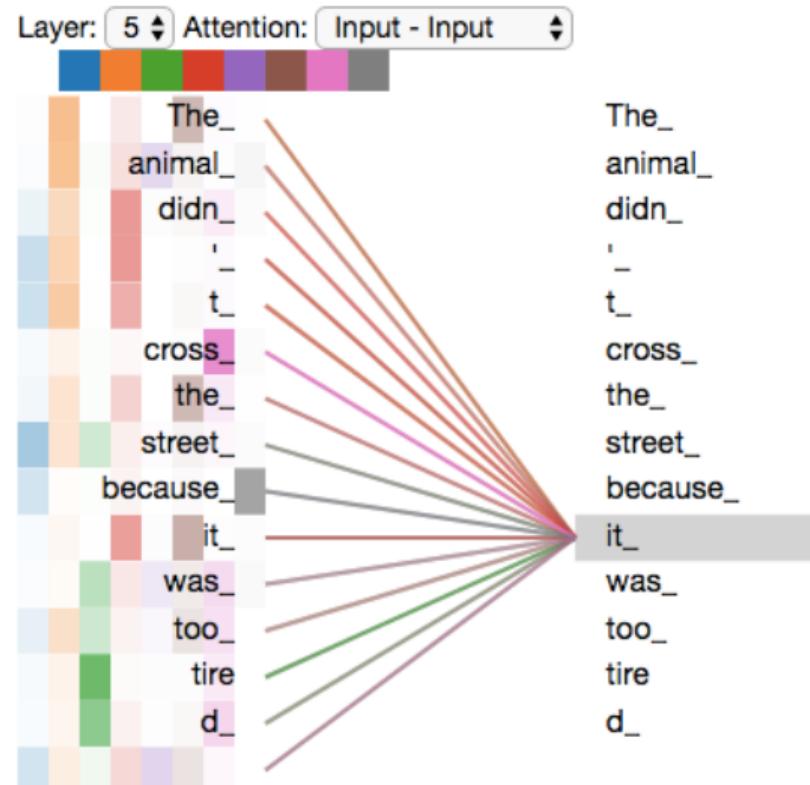
\* In all encoders other than #0, we don't need embedding.  
We start directly with the output of the encoder right below this one



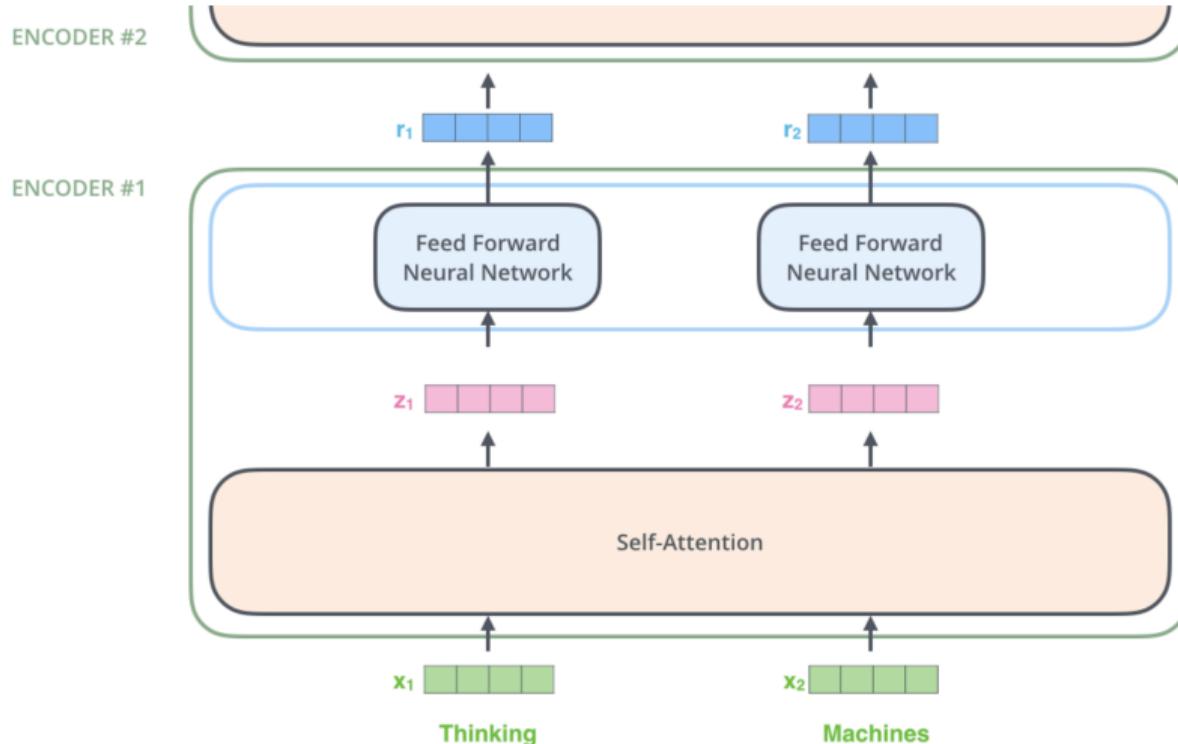
$R$



# ИТОГО

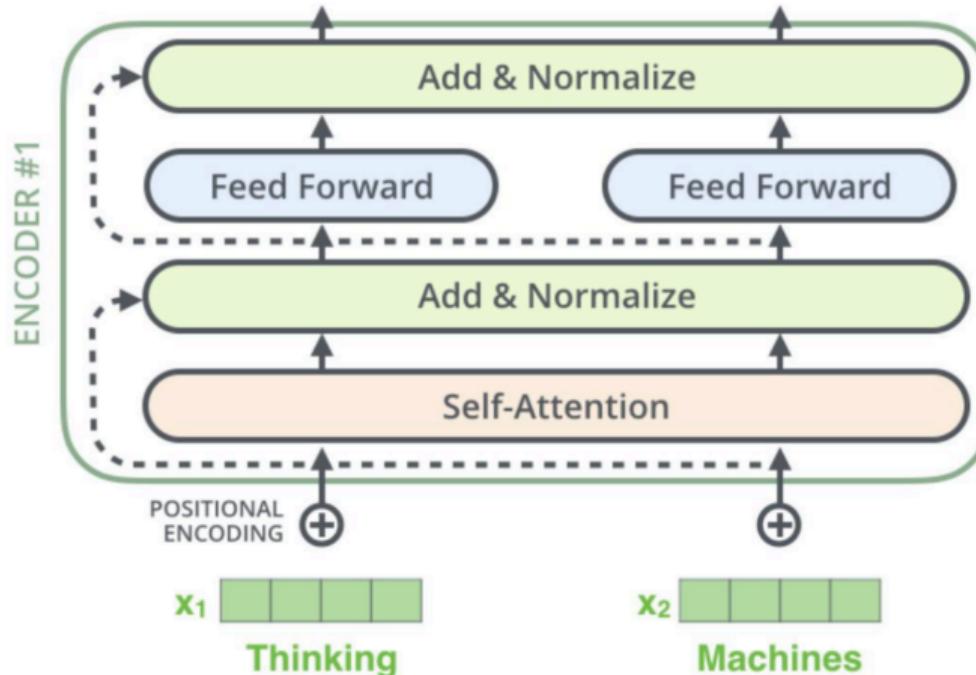


# До сих пор архитектура Transformer выглядела так



Источник: <https://jalammar.github.io/illustrated-transformer/>

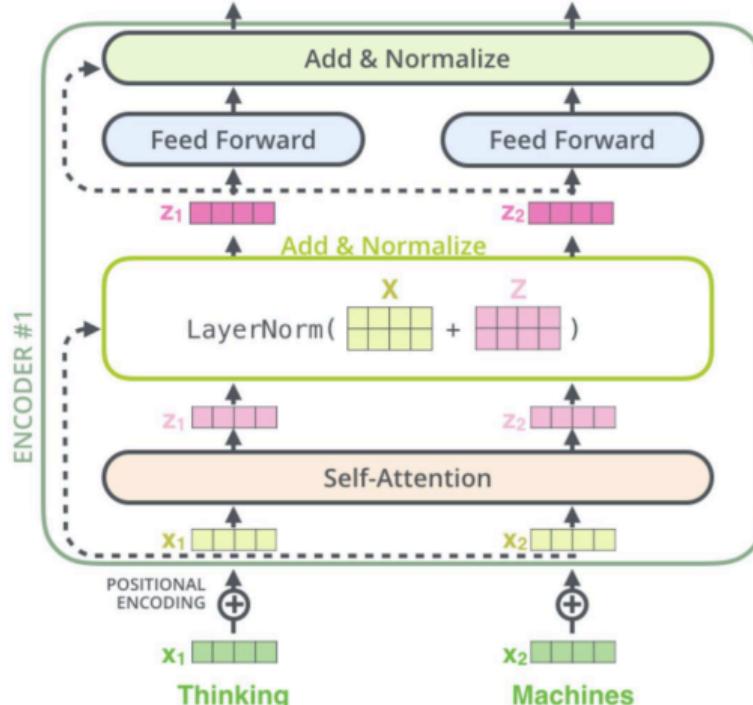
# Углубимся в детали: Layer normalization



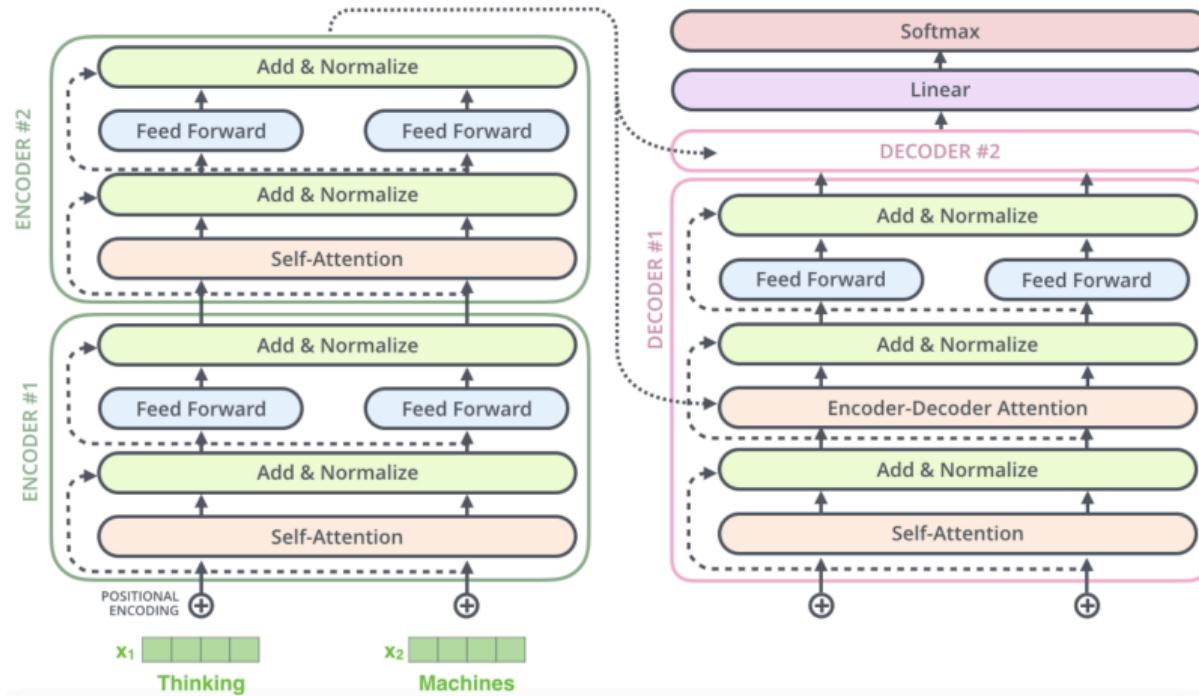
# Layer normalization

Like BatchNorm

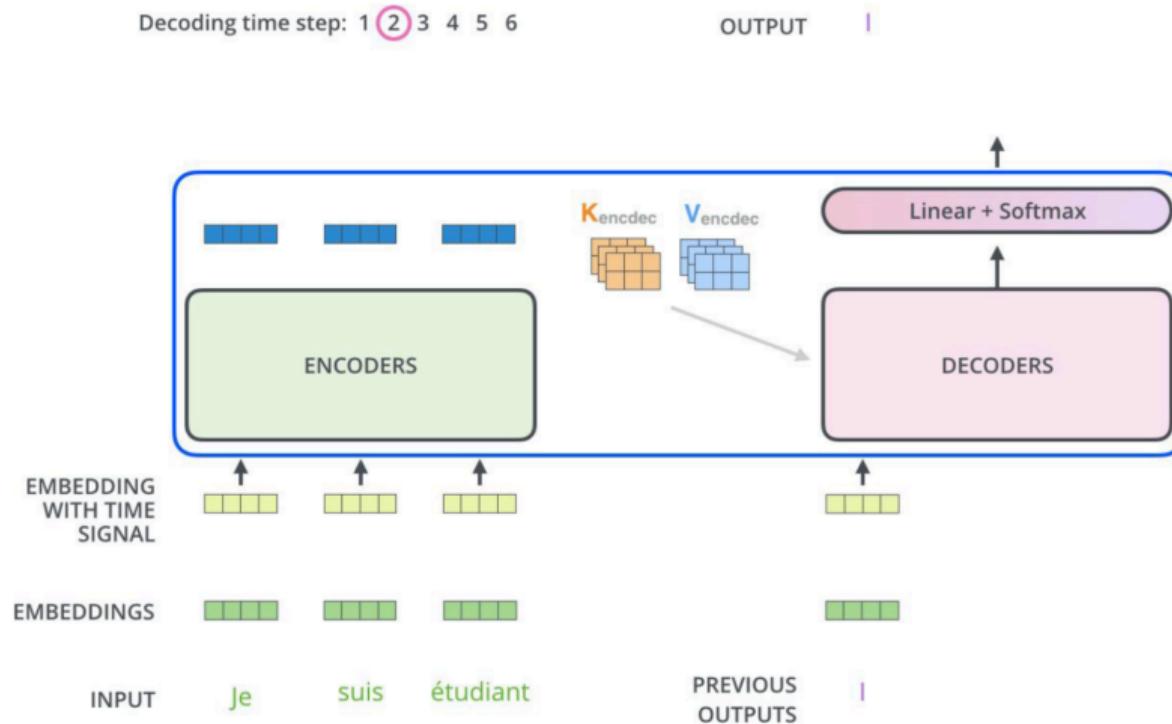
but normalize along  
all features  
representing latent  
vector



# Transformer detailed



# Encoder-Decoder Attention



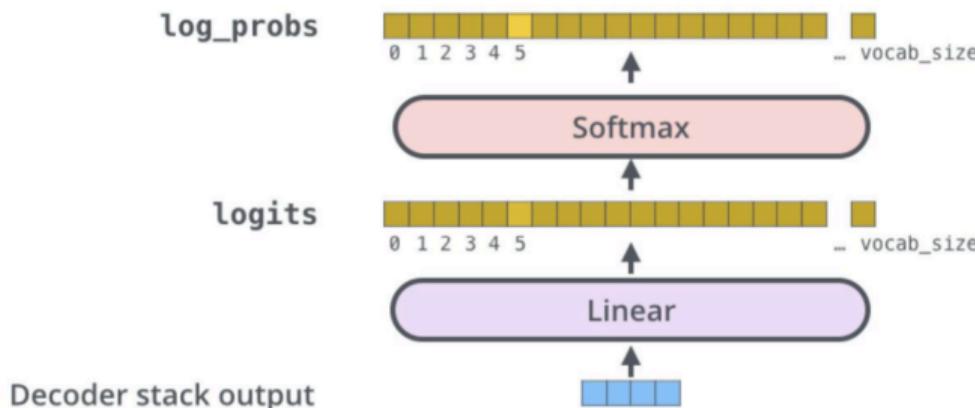
# Final steps - Linear layer

Which word in our vocabulary  
is associated with this index?

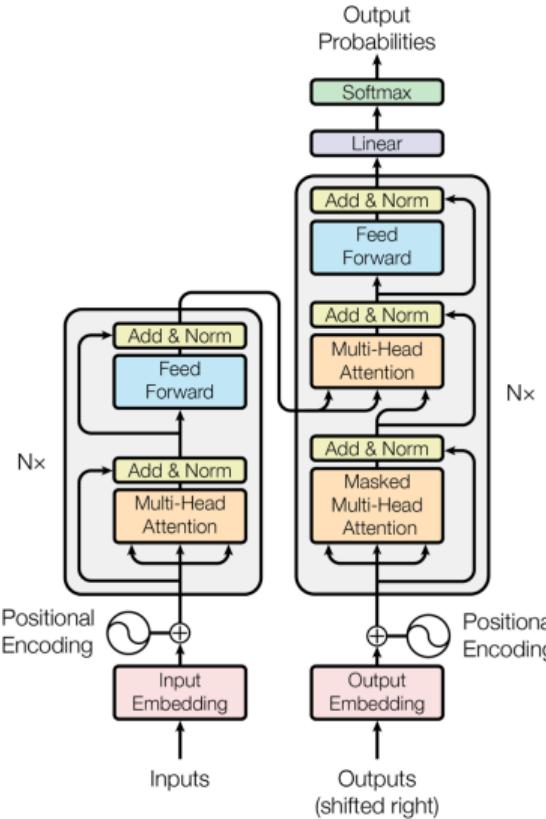
am

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

5



# Transformer full architecture once again



## Итого

1. Архитектура не содержит никаких слоев, кроме Dense
2. Архитектура хорошо обучается, находится множество взаимосвязей
3. Position encoding позволяет учитывать позицию в тексте



Дальнейшее развитие - это инженерные хитрости и усовершенствование желез

# BERT

Крутой обзор с техническими деталями - живет в лекциях МФТИ. Многие моменты этой лекции заимствованы из него.

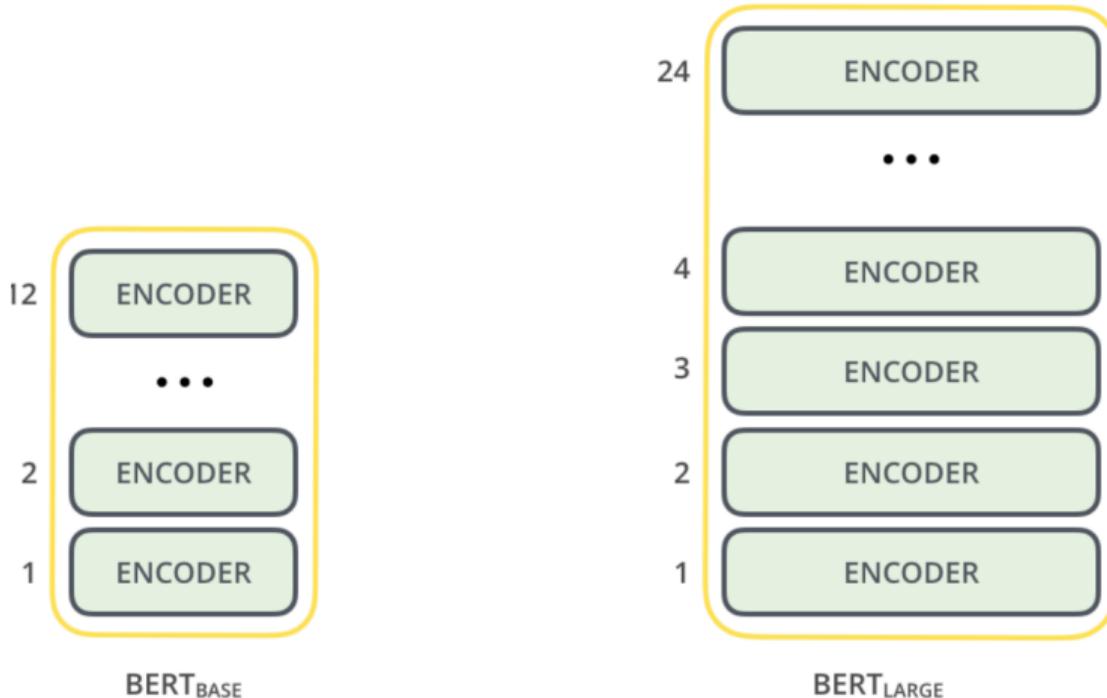
Ссылка на лекцию:

[https://www.youtube.com/playlist?list=PL4\\_hYwCyhAvY7k32D65q3xJVo8X8dc3Ye](https://www.youtube.com/playlist?list=PL4_hYwCyhAvY7k32D65q3xJVo8X8dc3Ye)

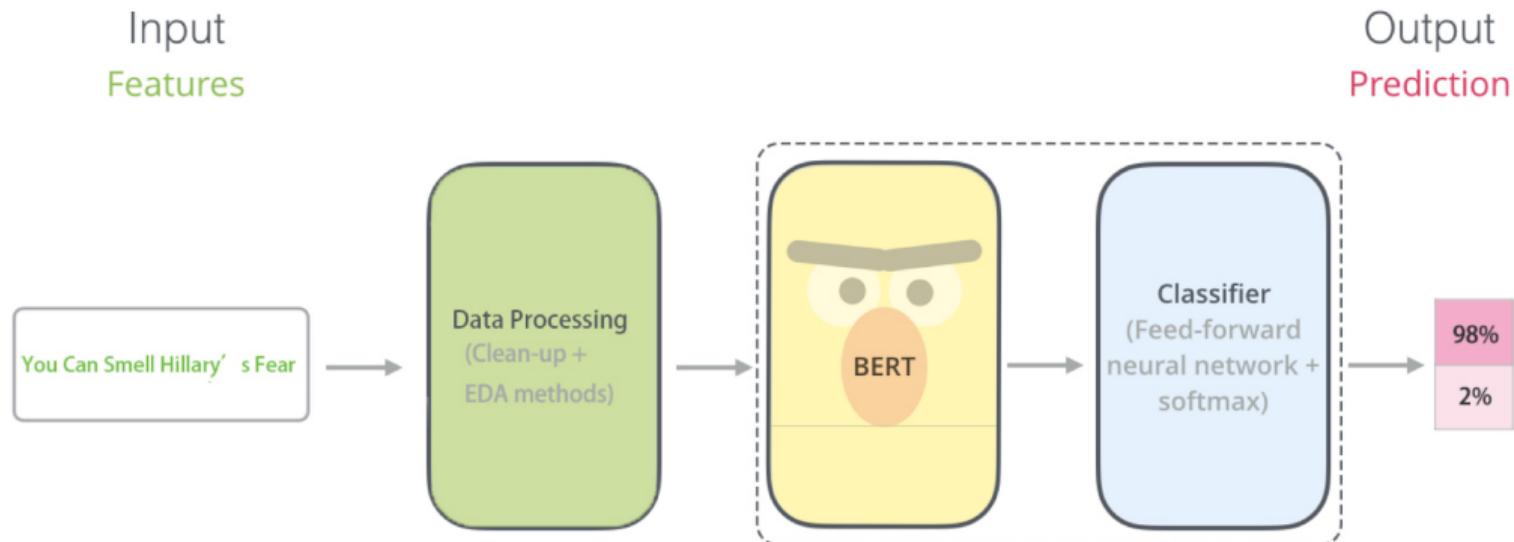
# BERT

- Шел 2018 год и Google сказал - наши компьютеры самые мощные, а данные самые большие!
- BERT - Bidirectional Encoder Representations from Transformers
- В чем прелесть - придумали как предобучать без учителя и потом переиспользовать веса!

# Архитектура BERT: Base, Large версии



# BERT, простейшее применение

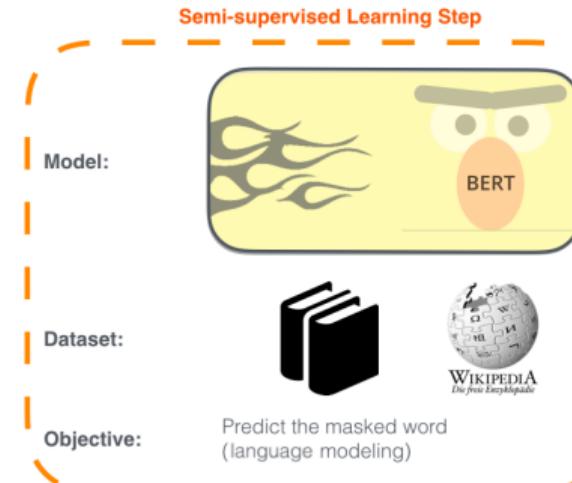


# 2 фазы BERT

BERT можно логически разделить на 2 фазы: Pre-training и Fine-tuning

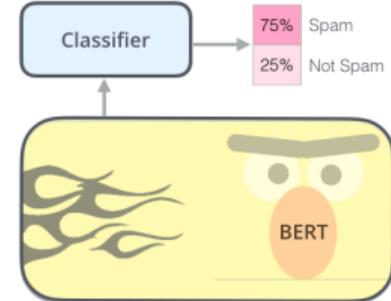
1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.

**Supervised Learning Step**



Dataset:

Email message	Class
Buy these pills	Spam
Win cash prizes	Spam
Dear Mr. Atreides, please find attached...	Not Spam

Источник: <https://jalammar.github.io/illustrated-bert/>

# Лежащие внутри идеи и почему он популярный

## 1. Pre-training по двум задачам:

- Masked LM: берем корпус текстов и маскируем часть предложений, тренируем предсказывать маску.
- Next sentence prediction: тренируем определять является ли следующее предложение истинно следующим для данного предложения

2. BERT из коробки знает язык, ему 1-2 эпохи надо подсказать, что с этим знанием делать
3. В готовых библиотеках лежат много предобученных под разные задачи Бертов - классификация, вопросно - ответные системы и тому подобное.
4. Опять же подаем слова кусочками (WordPiece), чтобы как-то решать проблему OOV.

Победа - нет необходимости размечать данные для обучения!

# Masked LM Pretraining

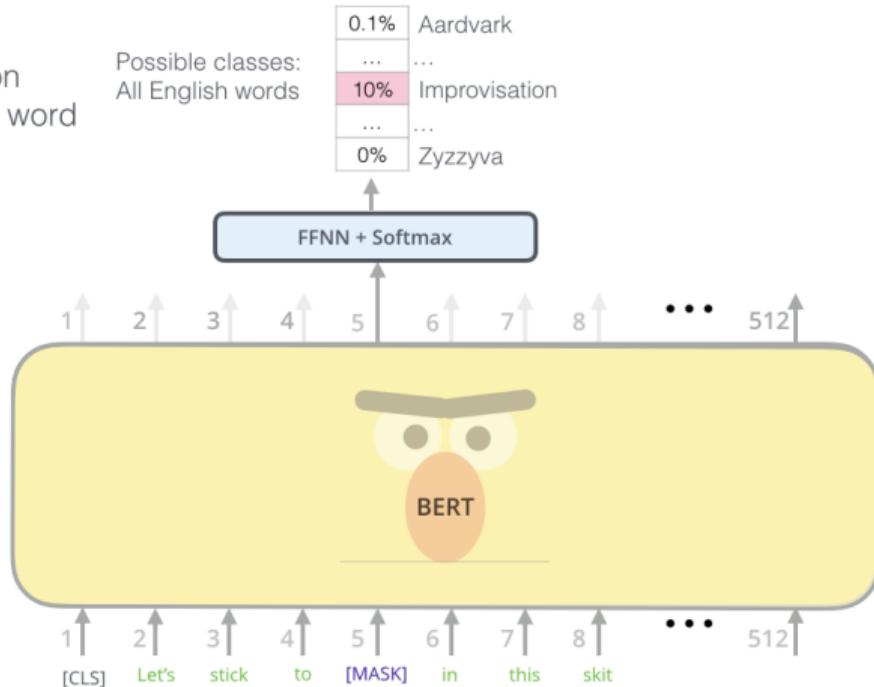
Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words

0.1%	Aardvark
...	...
10%	Improvisation
...	...
0%	Zyzyva

FFNN + Softmax

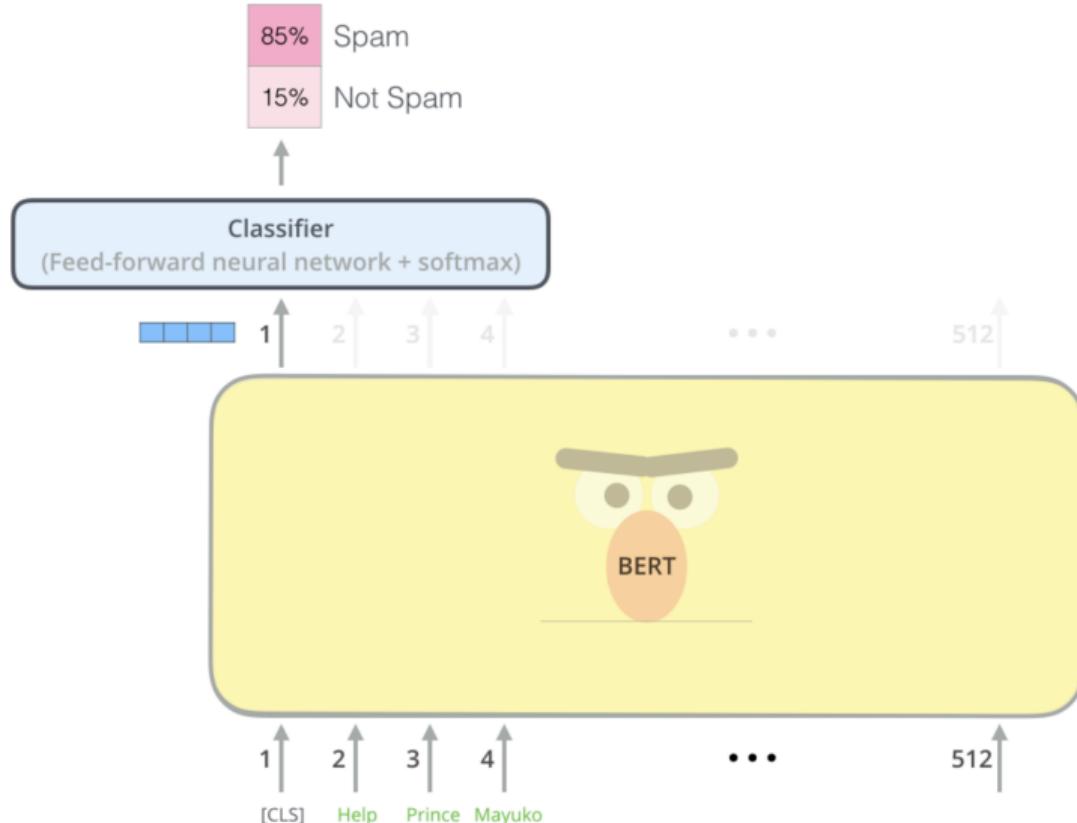
Randomly mask  
15% of tokens



Input

[CLS] Let's stick to improvisation in this skit

# Fine-tuning



# В каких задачах еще используется?

Spoiler: Картинка не про BERT. Взята из статьи про Open AI Transformer

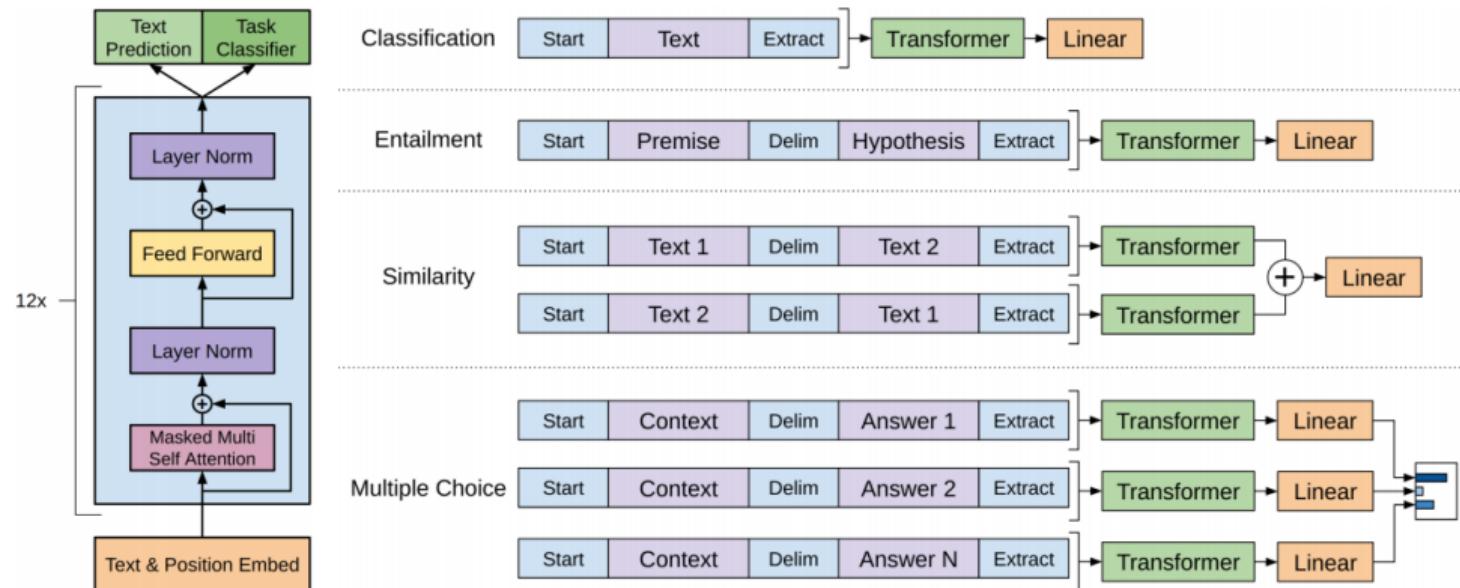
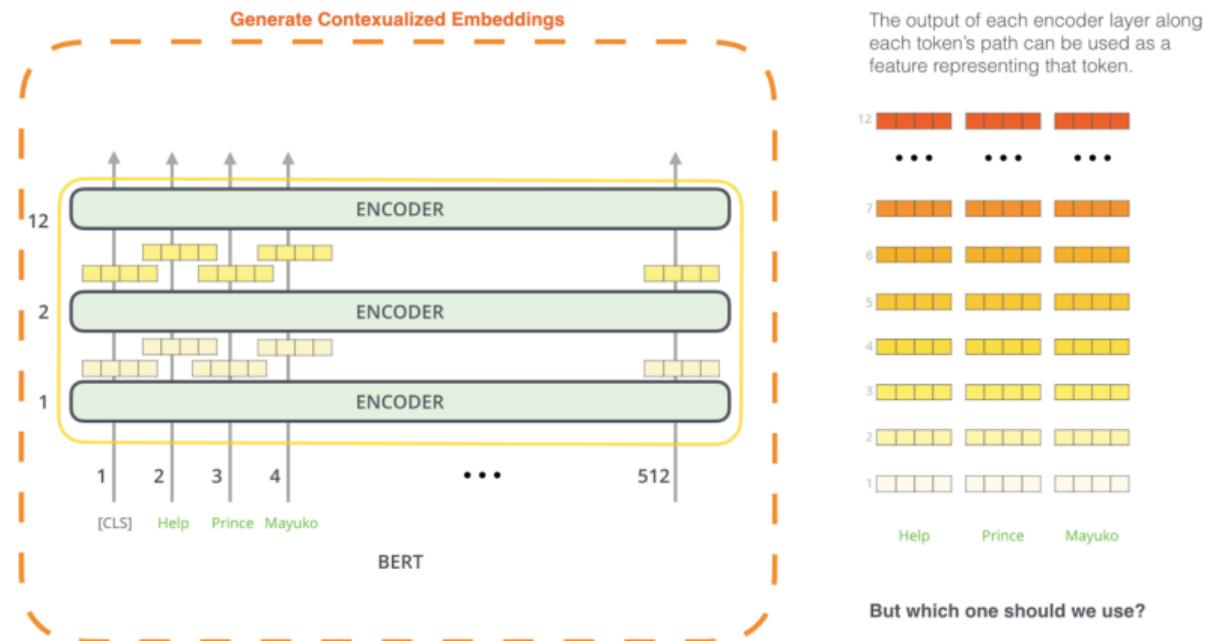


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

# BERT for feature extraction

И это еще не все!

The fine-tuning approach isn't the only way to use BERT. Just like ELMo, you can use the pre-trained BERT to create contextualized word embeddings.



# BERT for feature extraction

What is the best contextualized embedding for “Help” in that context?

For named-entity recognition task CoNLL-2003 NER

		Dev F1 Score
12	Embedding	91.0
• • •		
7		
6		
5		
4		
3		
2		
1		
Help		
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	12 + ... 2 + 1 =	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	12 + 11 + 10 + 9 =	95.9
Concat Last Four Hidden	9 10 11 12	96.1

# Немного статистики

<b>Model Type</b>	<b>Vocab Size</b>	<b>Hidden Dim</b>	<b># Params</b>	<b>Model Size (MB)</b>	<b>FLOPS ratio</b>
BERT <sub>DISTILLED</sub>	4928	48	1,775,910	6.8	1.3%
		96	5,665,926	22	1.32%
		192	19,169,094	73	4.49%
BERT <sub>BASE</sub>	30522	768	110,106,428	420	100%

# Семинар