Глубокое обучение

Бекезин Никита

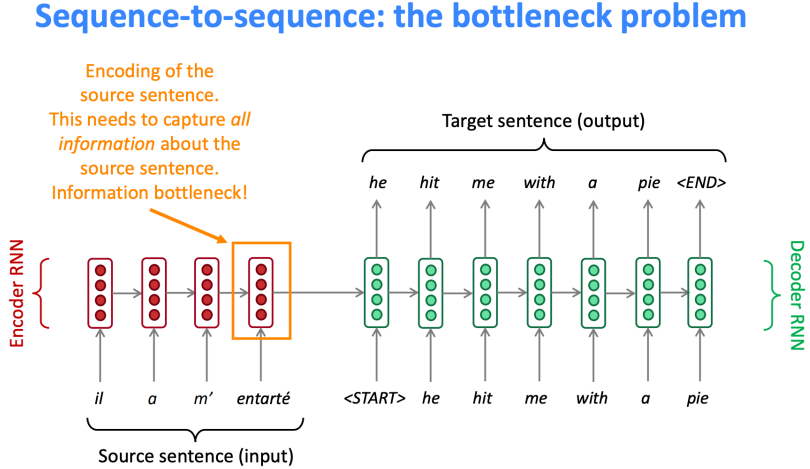
4 июня 2022

Лекция 9: Быстрое введение в Transformer + BERT

Agenda

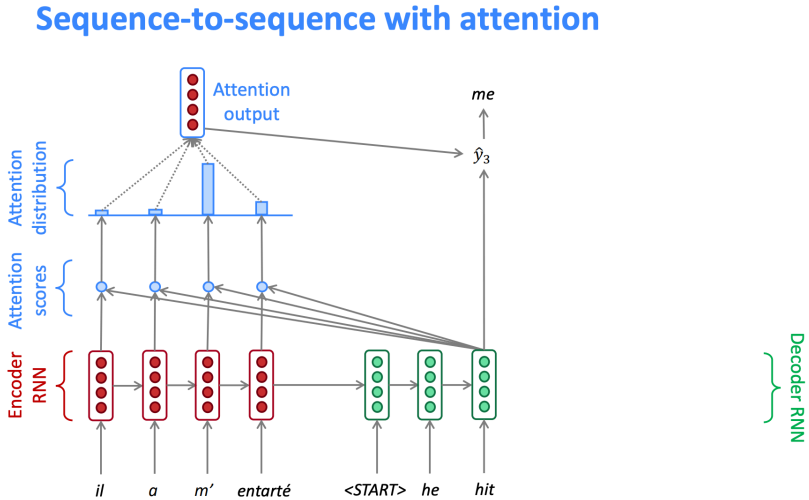
- Seq2seq recap
- Attention recap
- Self-Attention -> Transformer
- BERT

Seq2seq recap



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

Attention recap



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

Attention recap

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

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

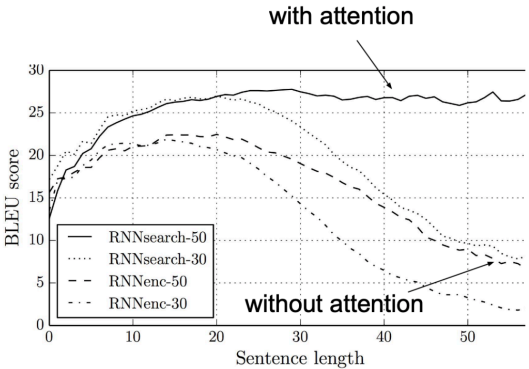
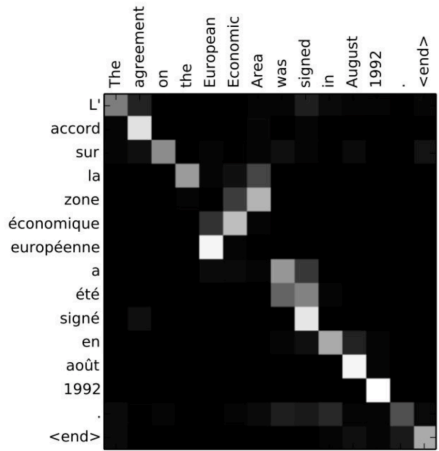


Image source: Neural Machine Translation by Jointly Learning to Align and Translate

Attention advantages

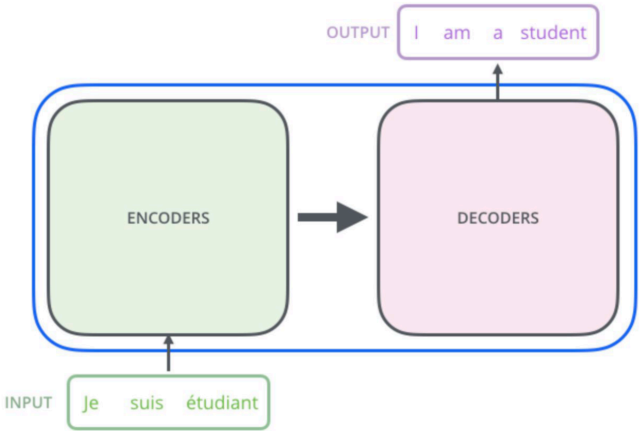


Attention is all you need!

Attention is all you need

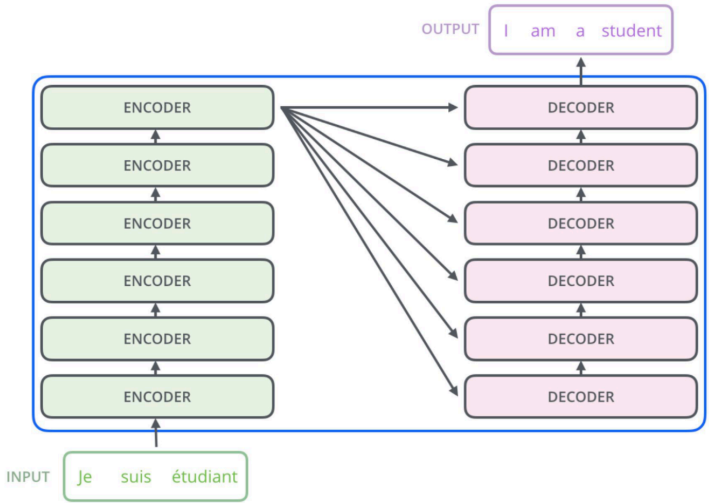
Данная статья - развитие идеи внимания. Статья вышла в 2017 году и стала истоком всех текущих SOTA моделей.
А зачем нам вообще что-то, кроме внимания? Идея в том, чтобы добавить в энкодер и декодер как можно больше внимания, что позволит уйти от реккурентной структуры в Encoder-Decoder архитектуре

Transformer



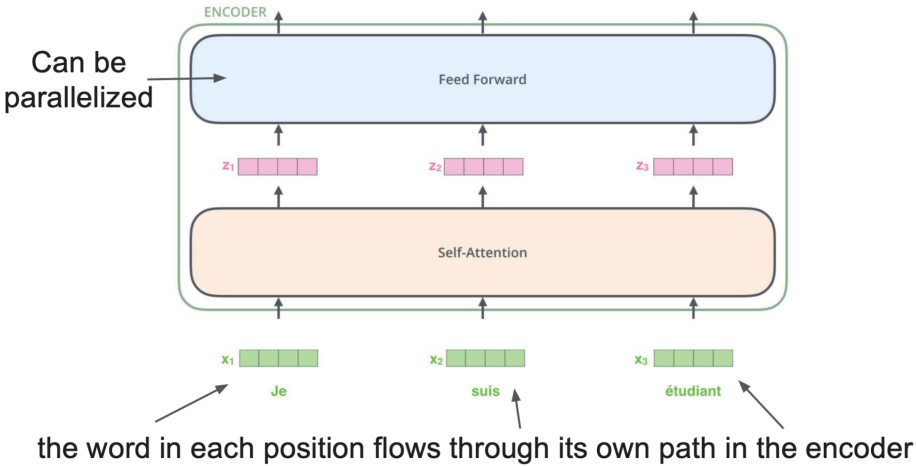
Источник: https://jalammar.github.io/illustrated-transformer/

Transformer



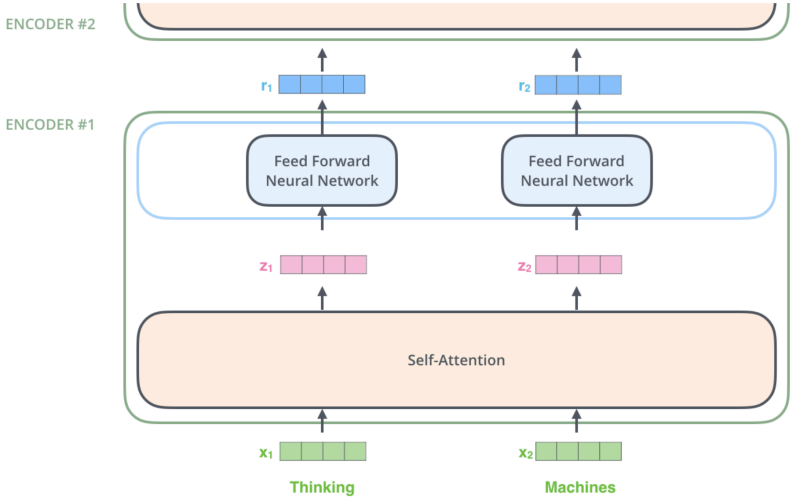
Источник: https://jalammar.github.io/illustrated-transformer/

Transformer



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

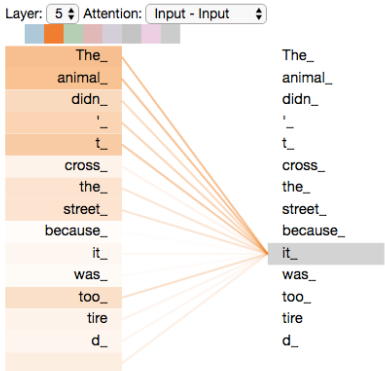
Transformer



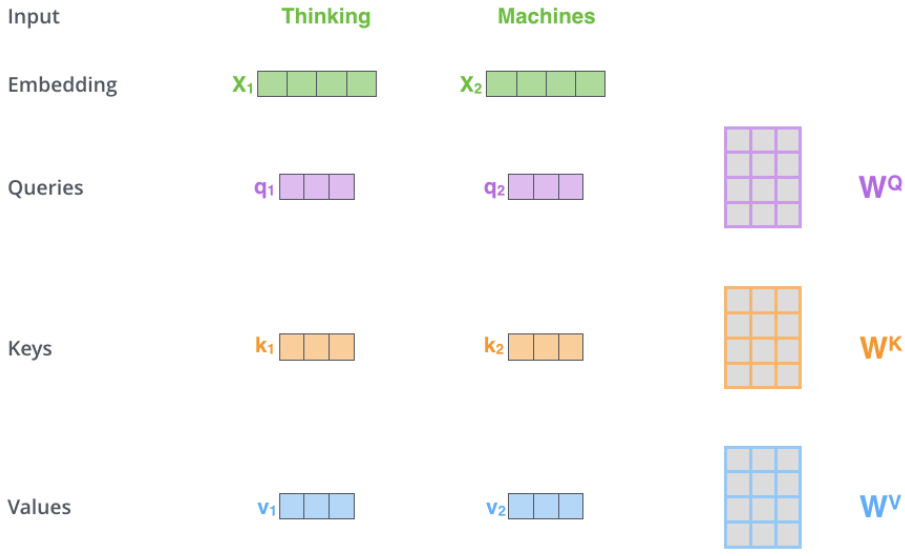
Источник: https://jalammar.github.io/illustrated-transformer/

Self-attention или механизм внимания

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



Шаг 1. Абстракции

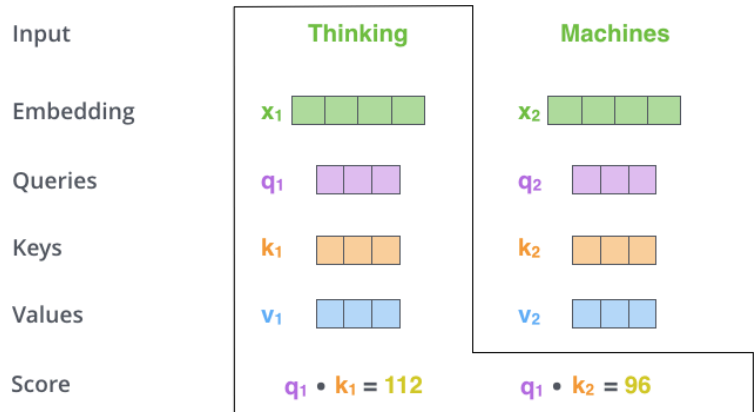


Self-attention

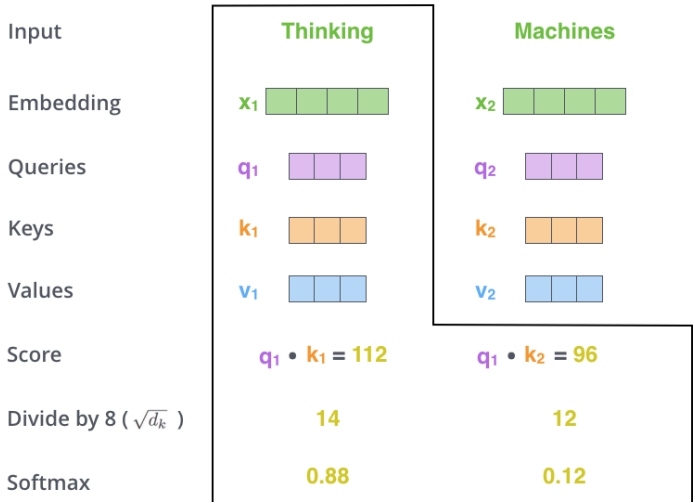
То же самое словами:

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

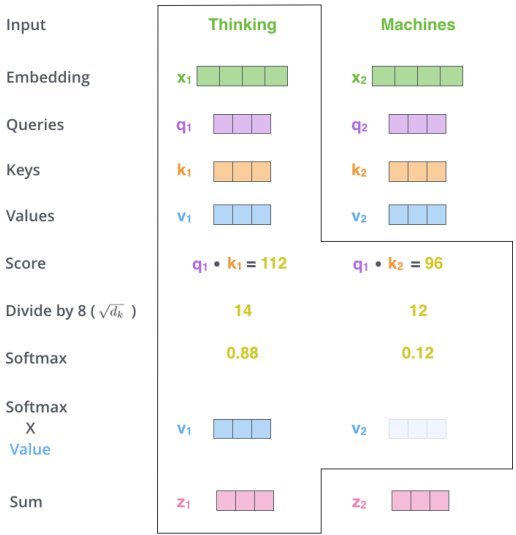
Шаг 2. Расчет attention score



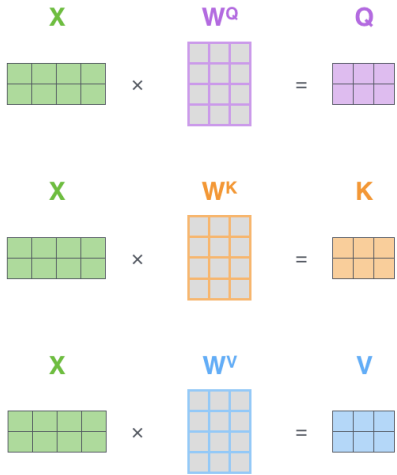
Шаг 3. Шкалирование и Softmax



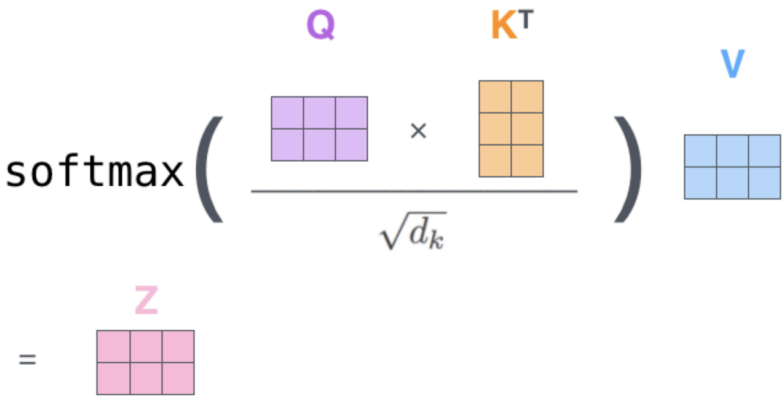
Шаг 4. Сумма взвешенных векторов Value



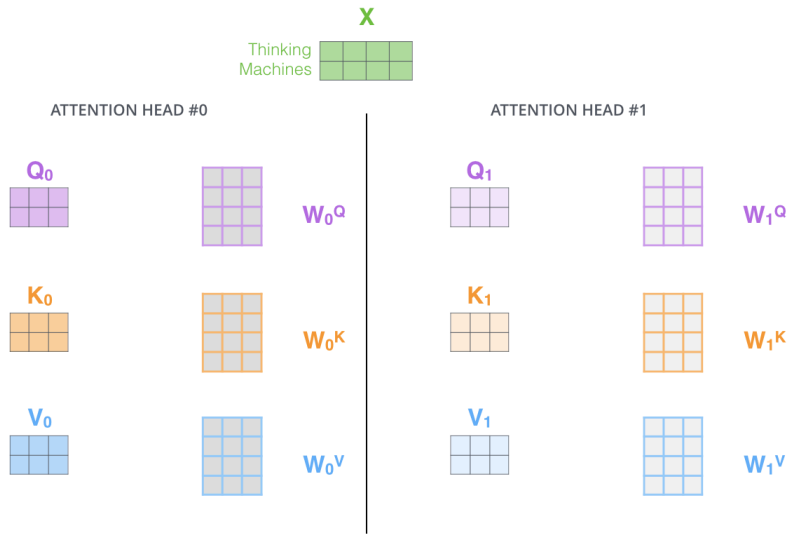
Матричные перемножения



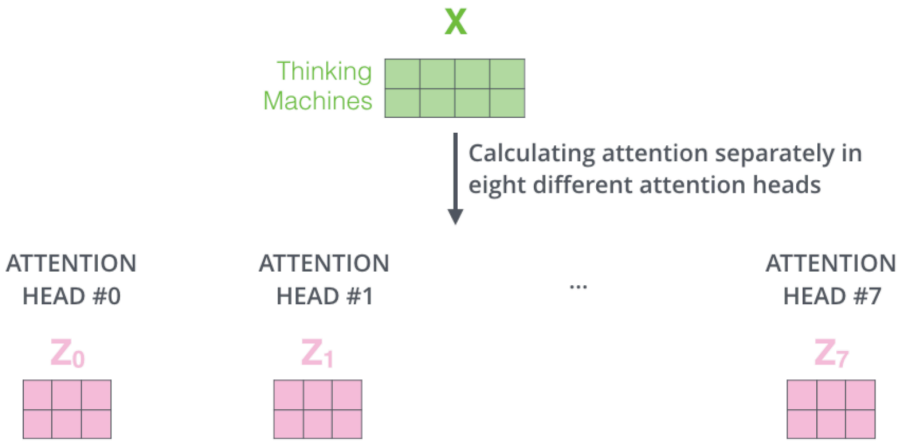
All together



Multi-Head Attention



Multi-Head Attention

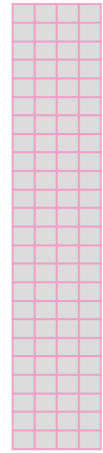


Соединяем!

1) Concatenate all the attention heads

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

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

=

Итого

1) This is our input sentence*

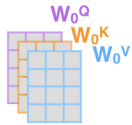
2) We embed each word*

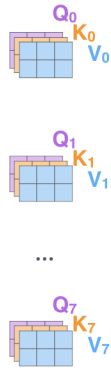
3) Split into 8 heads. We multiply X or R with weight matrices

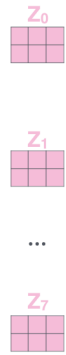
4) Calculate attention using the resulting Q/K/V matrices

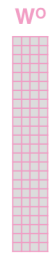
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

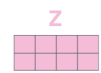
Thinking Machines





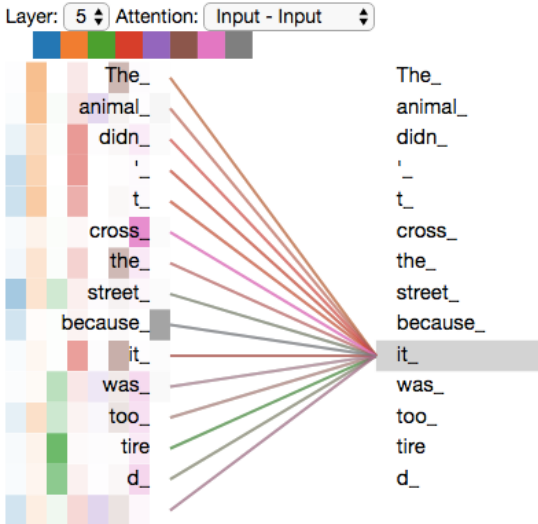




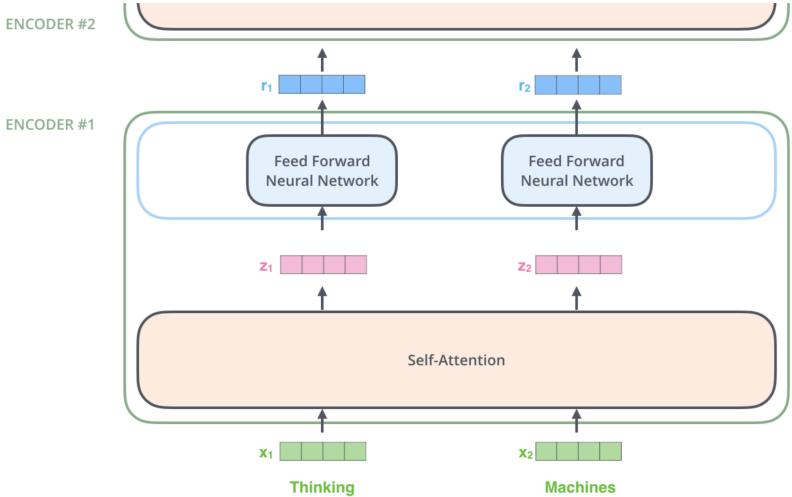


* In all encoders other than #0, we don't need embedding.
We start directly with the output of the encoder right below this one

Итого

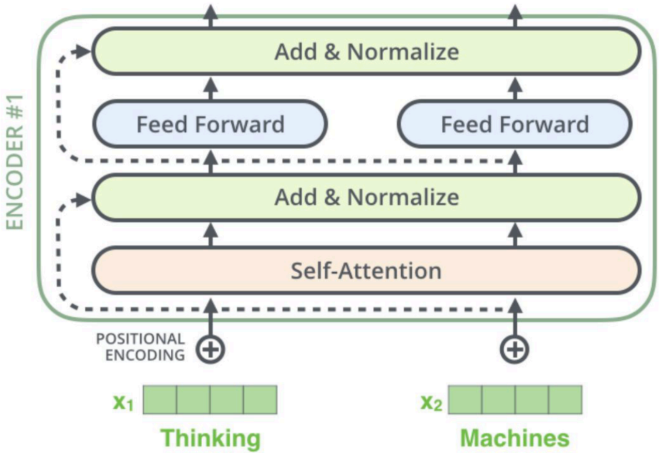

Layer: 5
Attention: Input - Input
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_
The_
animal_
didn_
'_
t_
cross_
the_
street_
because_
it_
was_
too_
tire
d_

До сих пор архитектура Transformer выглядела так



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

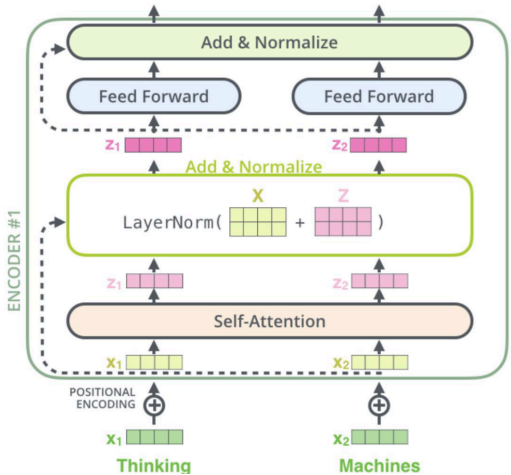
Углубимся в детали: Layer normalization



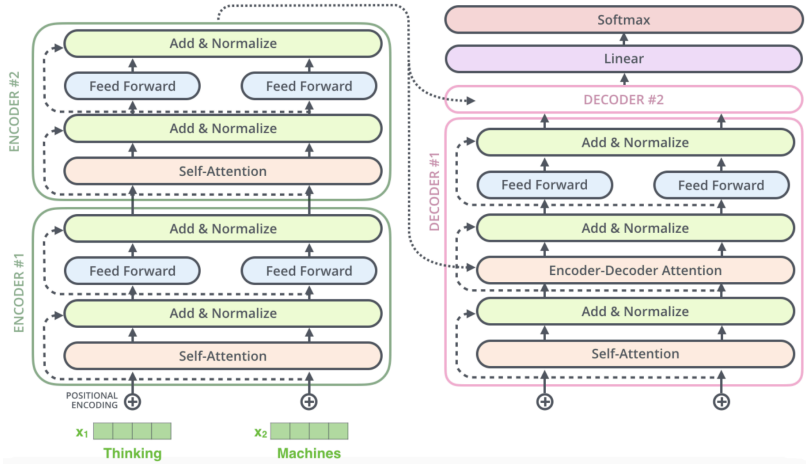
Layer normalization

Like BatchNorm

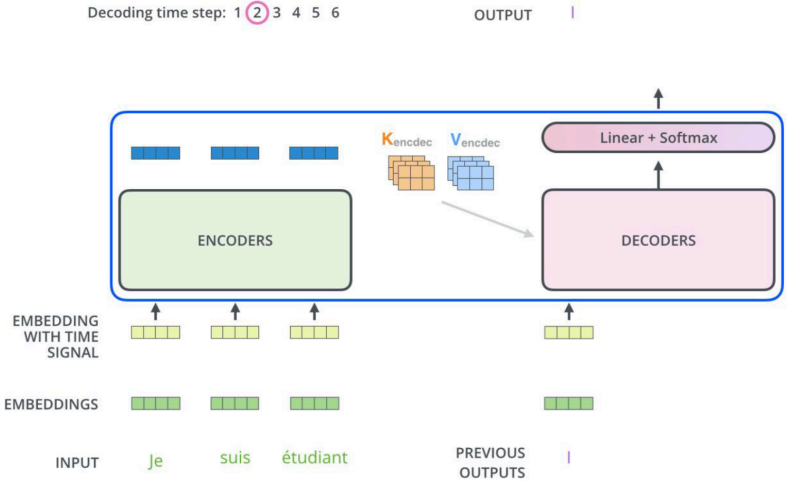
but normalize along all features representing latent vector



Transformer detailed



Encoder-Decoder Attention



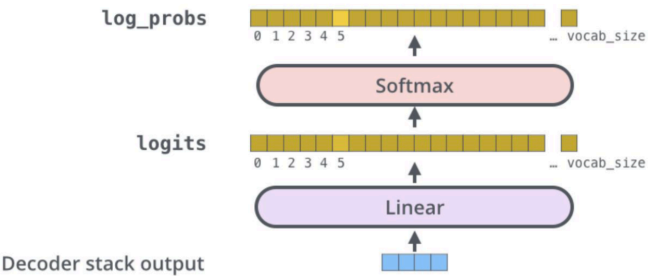
Final steps - Linear layer

Which word in our vocabulary is associated with this index?

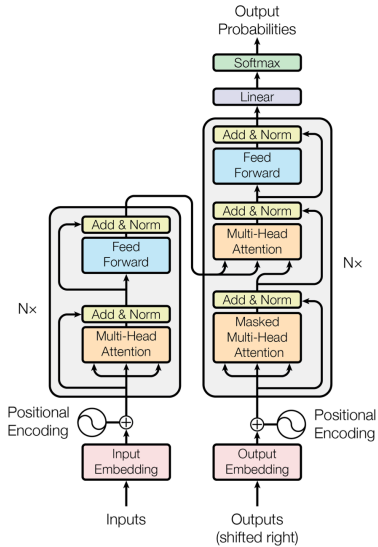
am

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

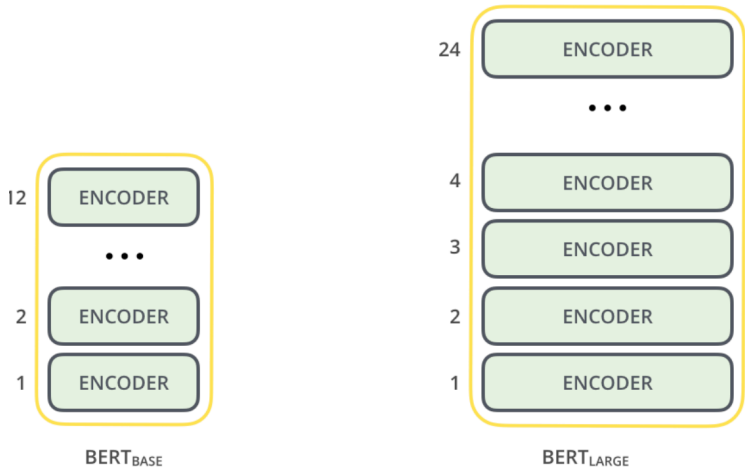
Крутой обзор с техническими деталями - живет в лекциях МФТИ. Многие моменты этой лекции заимствованы из него.

Ссылка на лекцию:
https://www.youtube.com/playlist?list=PL4_hYwCyhAvY7k32D65q3xJVo8X8dc3Ye

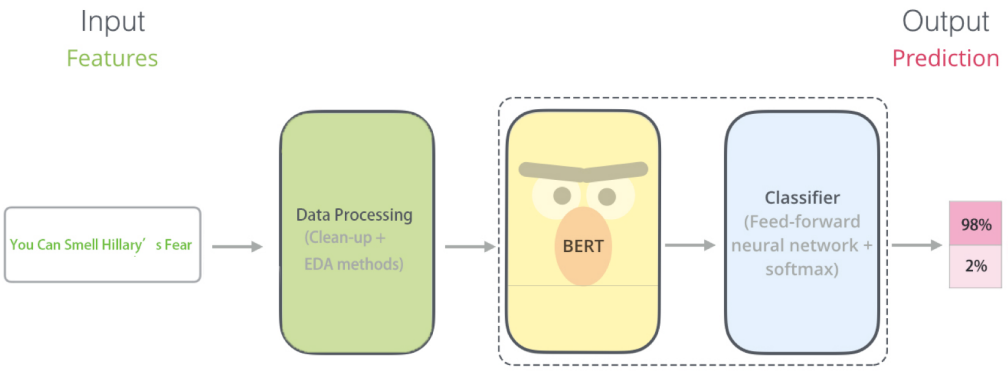
BERT

- Шел 2018 год и Google сказал - наши комьютеры 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

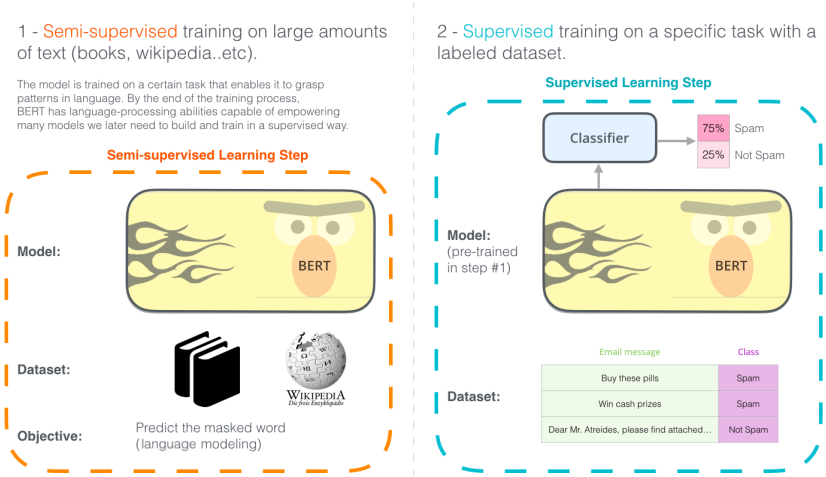


BERT, простейшее применение



2 фазы BERT

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



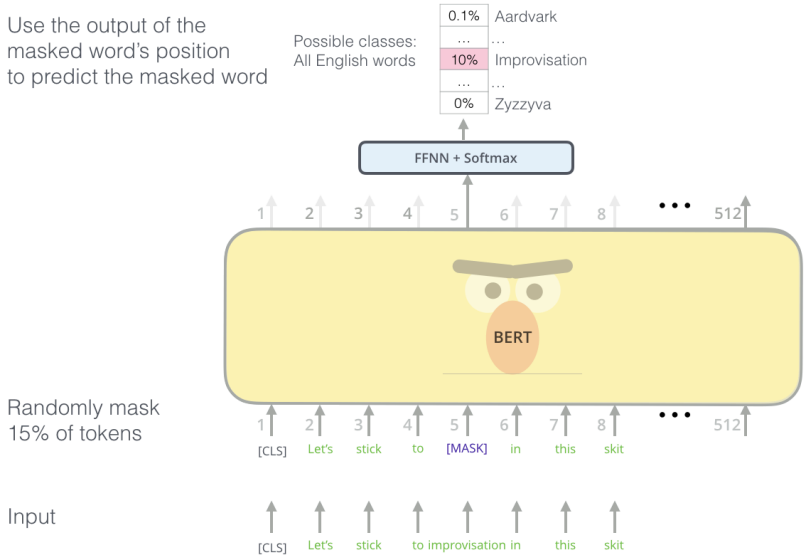
Источник: https://jalammar.github.io/illustrated-bert/

Лежащие внутри идеи и почему он популярный

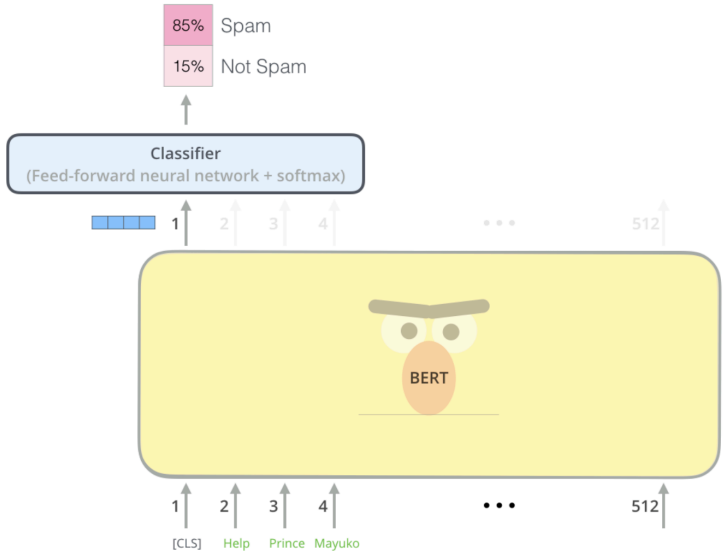
1. Pre-training по двум задачам:
   - Masked LM: берем корпус текстов и маскируем часть предложений, тренируем предсказывать маску.
   - Next sentence prediction: тренируем определять является ли следующее предложение истинно следующим для данного предложения
2. BERT из коробки знает язык, ему 1-2 эпохи 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



Fine-tuning


85%
Spam
15%
Not Spam
Classifier
(Feed-forward neural network + softmax)
1
2
3
4
512
BERT
1
2
3
4
512
[CLS]
Help
Prince
Mayuko

В каких задачах еще используется?

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

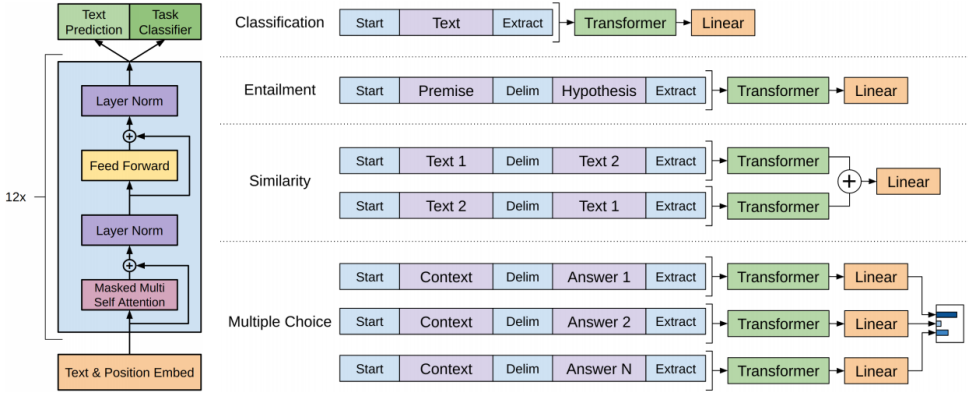
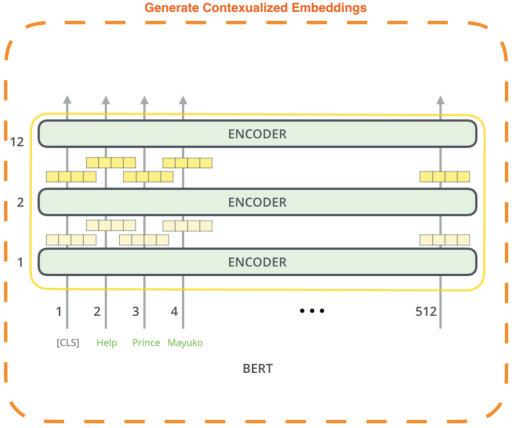


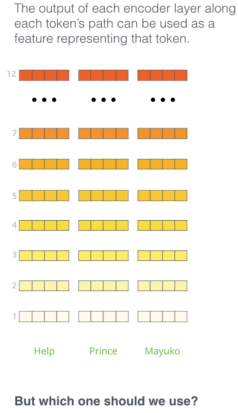
Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

BERT for feature extraction

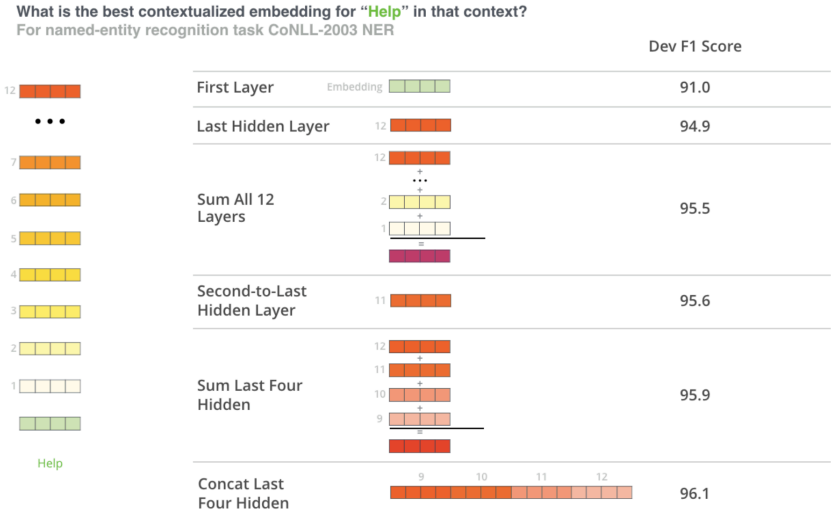
И это еще не все!

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



Немного статистики

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

Семинар