

Глубокое обучение

Дмитрий Никулин

30 июня 2021 г.

Неделя 14: Трансформеры

Agenda

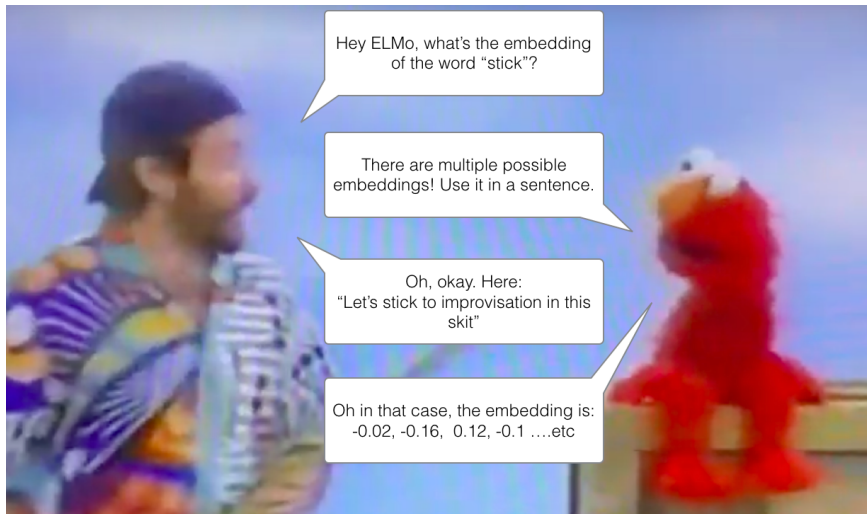
- Развитие идеи эмбедингов
- RNN и механизм внимания
- Attention is all you need
- Модификации трансформера

Развитие идеи эмбедингов

Серия вопросов в зал

- Как работают разные эмбединги?
- В чем, по вашему мнению, их главная проблема?

Embeddings from Language MOdelS (ELMO)

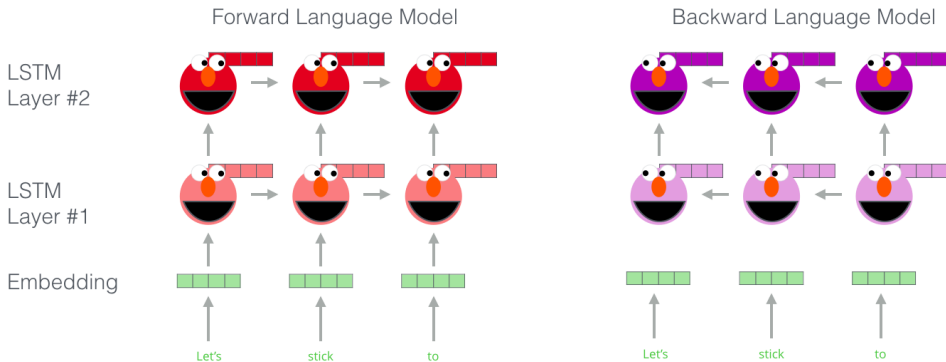


<https://arxiv.org/pdf/1802.05365.pdf>

ELMO

Обучаем две языковые модели: в одной LSTM предсказывает следующее слово, в другой предыдущее (т.е. это bidirectional LSTM):

Embedding of “stick” in “Let’s stick to” - Step #1



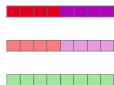
<http://jalammar.github.io/illustrated-bert/>

ELMO

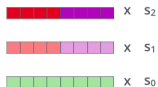
В качестве эмбединга используем взвешенную сумму начального эмбединга и скрытых состояний LSTM:

Embedding of "stick" in "Let's stick to" - Step #2

1- Concatenate hidden layers



2- Multiply each vector by a weight based on the task

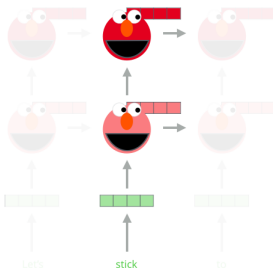


3- Sum the (now weighted) vectors

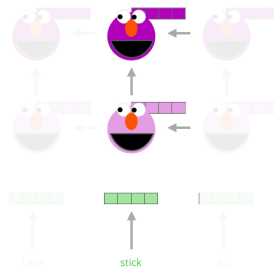


ELMo embedding of "stick" for this task in this context

Forward Language Model



Backward Language Model

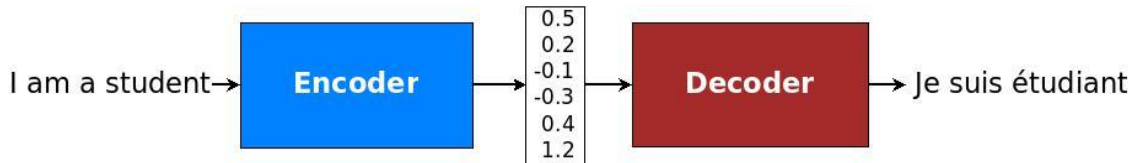


<http://jalamar.github.io/illustrated-bert/>

RNN и механизм внимания

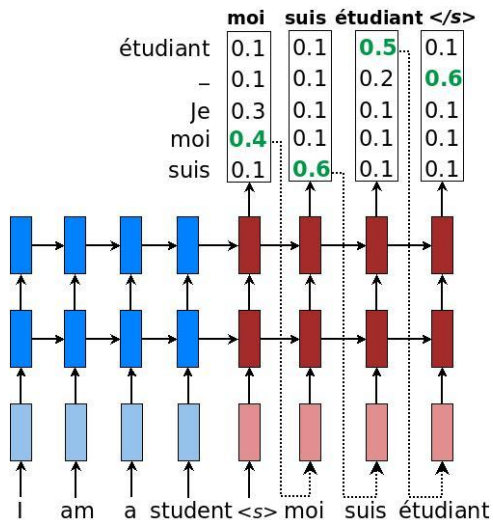
Проблема RNN

- При решении seq2seq задач предложения произвольной длины кодируются в вектор фиксированного размера
- В длинных предложениях теряется контекст, длинные предложения не зависят от начальных токенов
- LSTM и BiLSTM пытаются частично решить эту проблему



<https://github.com/tensorflow/nmt>

Автопереводы

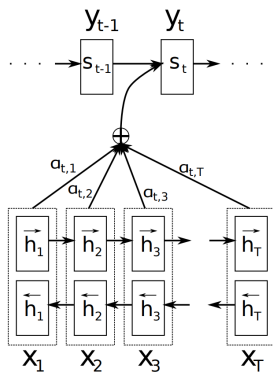


<https://github.com/tensorflow/nmt>

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

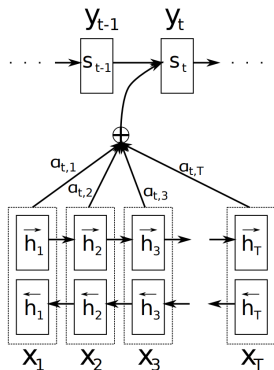
Механизм внимания



- На вход энкодеру подаём все скрытые состояния
- Хотим научить нейросеть смотреть в нужные места исходной последовательности

<https://arxiv.org/pdf/1409.0473.pdf>

Механизм внимания



<https://arxiv.org/pdf/1409.0473.pdf>

- Скрытое состояние декодера:

$$s_t = f(s_{t-1}, y_{t-1}, c_t)$$

- Релевантность j -го входного слова t -ому (обычно это полносвязный слой):

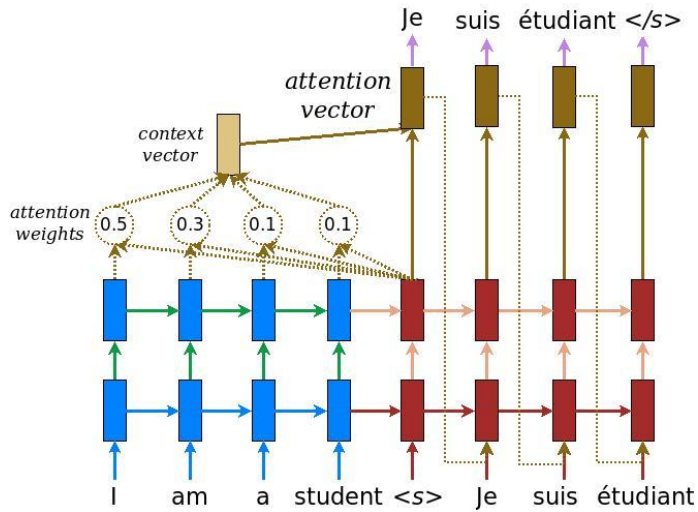
$$\text{sim}(s_{t-1}, h_j)$$

- Распределение на входных словах:

$$\alpha_{tj} = \text{Softmax}(\text{sim}(s_{t-1}, h_j))$$

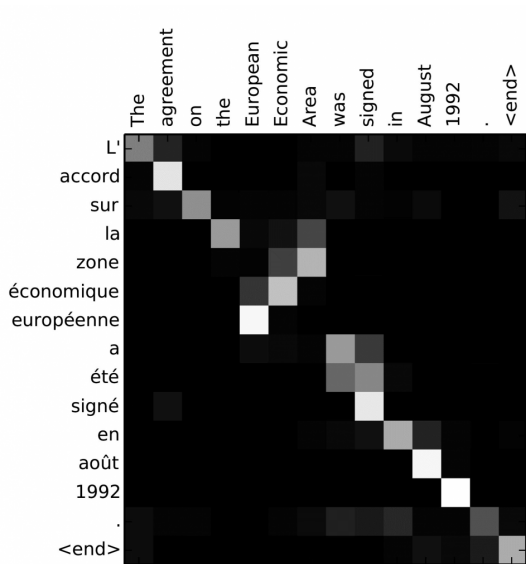
- Предсказываем, какие слова входной последовательности важны: $c_t = \sum_j \alpha_{tj} \cdot h_j$

Механизм внимания



<https://github.com/tensorflow/nmt>

Механизм внимания



Как посчитать sim?

- Скалярное произведение:

$$\text{sim}(s, h) = h^T \cdot s$$

- Additive attention:

$$\text{sim}(s, h) = W^T \cdot \tanh(W_h h + W_s s)$$

- Multiplicative attention:

$$\text{sim}(s, h) = h^T W s$$

- W, W_s, W_h - обучаемые параметры

Как улучшить seq2seq?

Attention is All You Need (2017)

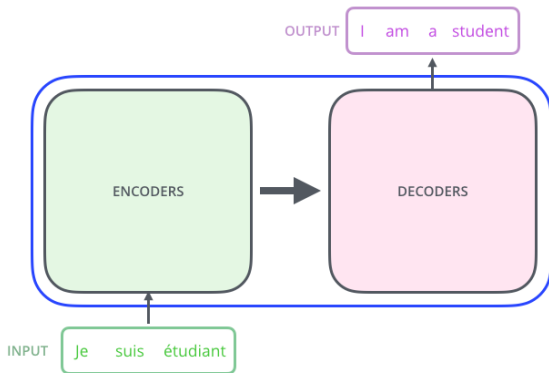
- Основная проблема RNN — модель помнит только ближайший контекст
- В первой статье про attention рекуррентная сеть была как в энкодере (для вычисления эмбеддингов), так и в декодере
- Хотелось бы сделать модель, в которой:
 - эмбеддинги каждого слова "знали" не только о ближайших эмбеддингах
 - декодер мог бы помнить всю историю

Attention is All You Need (2017)

- Это реализовано в Transformer — нейросетевой архитектуре для задач seq2seq, основанной исключительно на полносвязных слоях
- Превзошла существовавшие seq2seq архитектуры как по качеству, так и по скорости работы
- Основной элемент — multi-head self-attention

Transformer

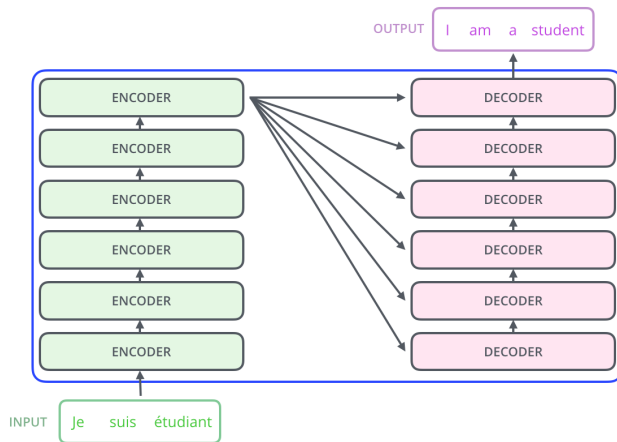
Верхнеуровнево это просто энкодер и декодер



<http://jalammar.github.io/illustrated-transformer/>

Transformer

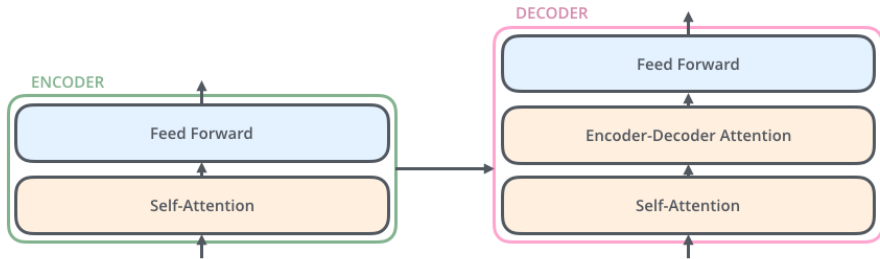
Энкодер и декодер состоят из одинаковых блоков; веса во всех блоках разные



<http://jalammar.github.io/illustrated-transformer/>

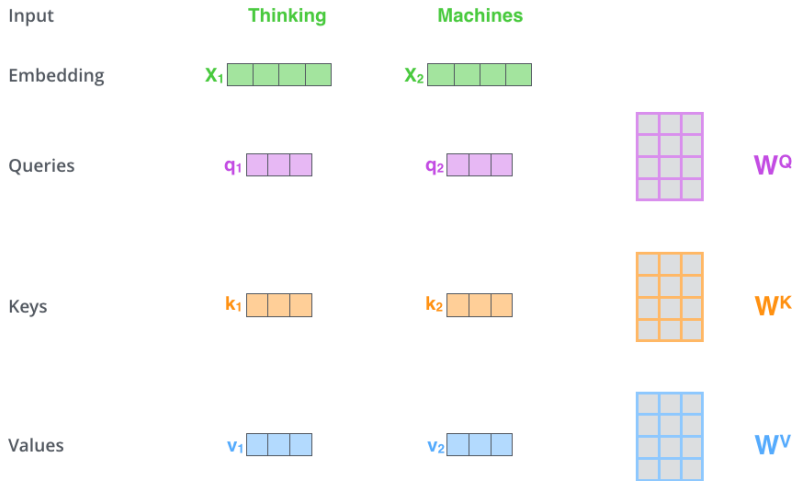
Transformer

В энкодере происходят две вещи: сначала вход прогоняется через self-attention, а затем — через полносвязный слой. В декодере помимо обычного self-attention есть ещё и attention из энкодера.



<http://jalammar.github.io/illustrated-transformer/>

Self-attention



<http://jalammar.github.io/illustrated-transformer/>

Абстракции

- Для эмбединга x_i каждого входного слова считаются три вектора:
 - Query (q_i) — представление слова для сравнения его с другими
 - Key (k_i) — представление слова для сравнения других с ним
 - Value (v_i) — представление слова, содержащее его смысл
- Матрицы W^Q, W^K, W^V обучаются вместе с моделью
- q_i, k_i, v_i получаются умножением x_i на соответствующие матрицы
- Функция релевантности: $\text{sim}(x_i, x_j) = \frac{q_i k_j^T}{\sqrt{d_k}}$, где d_k — размерность q_i и k_j

Self-attention

Цель этого слоя — сложить Value с некоторыми весами

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

<http://jalammar.github.io/illustrated-transformer/>

Более детально

Input

Embedding

Queries

Keys

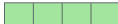
Values

Score

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

Softmax

Thinking

x_1 

q_1 

k_1 

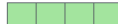
v_1 

$q_1 \cdot k_1 = 112$

14

0.88

Machines

x_2 

q_2 

k_2 

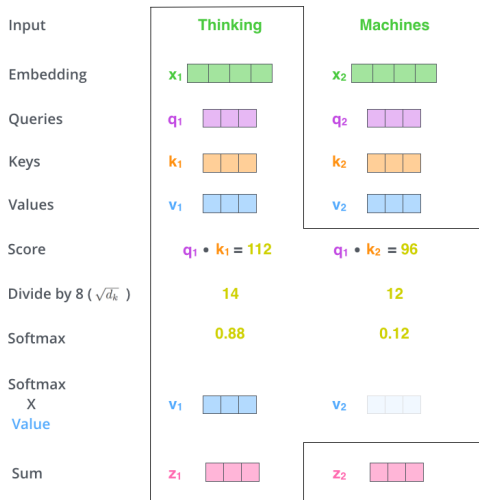
v_2 

$q_2 \cdot k_2 = 96$

12

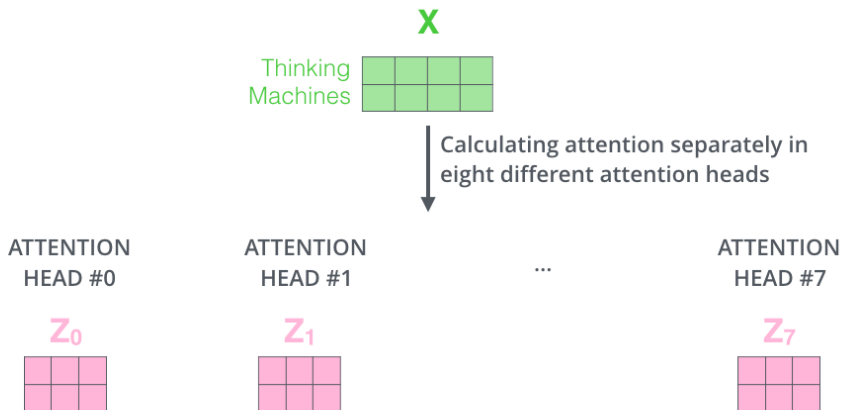
0.12

Более детально



Зверь с кучей голов

Несколько голов обеспечивают разное внимание



Ещё раз обзор multi-head self-attention

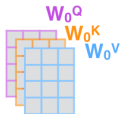
1) This is our input sentence*

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



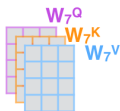
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



...

...

...



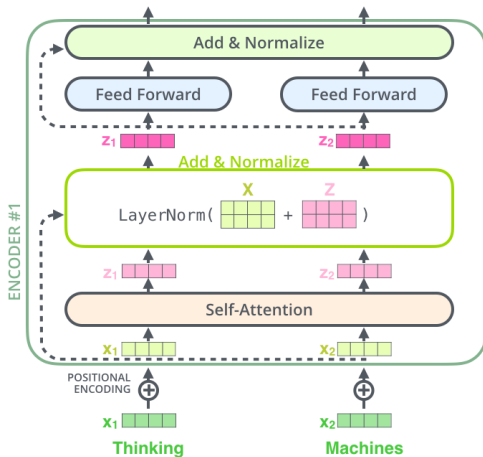
W^O



Z

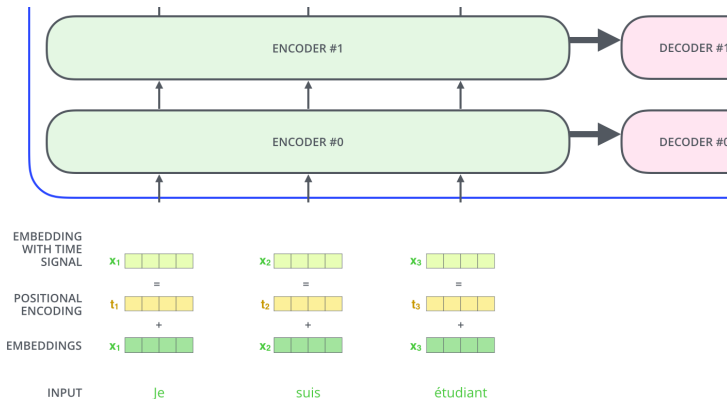


Один слой энкодера



Positional encoding

Для учёта позиции слова в предложении входные эмбеддинги можно преобразовывать каким-то способом, зависящим от позиции. Например, прибавлять какой-то тензор t_i (разный для разных i):



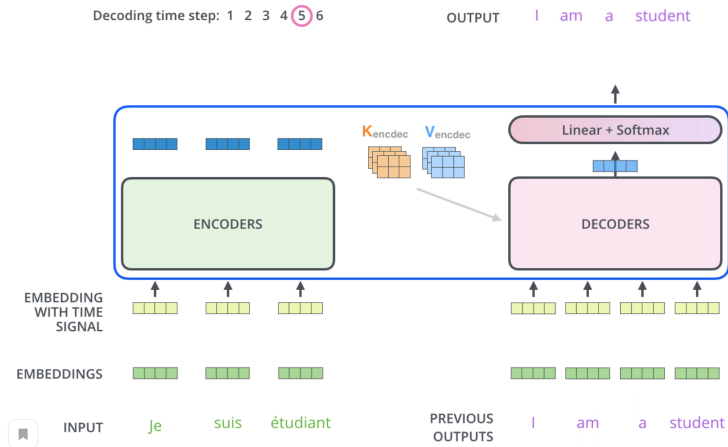
Positional encoding

- Самый простой вариант для t_i — это сделать их обучаемыми
- В оригинальной статье предлагалось сделать t_i из базисных векторов Фурье-базиса:

$$t_{i,2j} = \sin\left(\frac{i}{10000^{2j/d_{\text{model}}}}\right)$$
$$t_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d_{\text{model}}}}\right)$$

- Количество различных t_i задаётся при создании модели. Поэтому предобученную модель нельзя использовать для сколь угодно длинных последовательностей

Что происходит в декодере?



<http://jalammar.github.io/illustrated-transformer/>

Полная архитектура

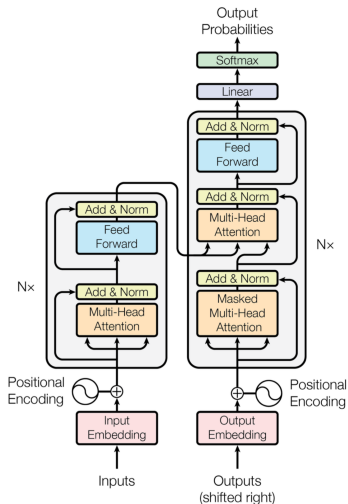


Figure 1: The Transformer - model architecture.

Модификации трансформера

Учить трансформер под каждую задачу сложно

- Большие объёмы неразмеченных данных в интернете в разных доменах (книги, новости, Википедия, иные тексты из интернет-страниц)
- Размеченных данных мало. Качественная разметка дорогая и долгая
- Много вычислительных ресурсов, GPU, TPU, фреймворки распределённых вычислений
- Можем ли мы как-то заиспользовать имеющиеся ресурсы?

Модификации

- Да, можем! И использовать будем semi-supervised learning
- Обучаем большой трансформер на какой-нибудь unsupervised задаче на очень больших данных (очень долго, порядка нескольких недель на 64 GPU);
- Дообучаем трансформер на малом корпусе размеченных данных (очень быстро, порядка 1 часа на одной GPU).

Сколько стоит аренда 1 сервера с 8 GPU

Рассчитать стоимость

Compute Cloud

Object Storage

Kubernetes®

PostgreSQL

ClickHouse

MongoDB

MySQL®

Redis™

Elasticsearch

SpeechKit

Translate

Техническая поддержка

Compute Cloud

Операционная система ?

Ubuntu 20.04 LTS

Платформа ?

Intel® Cascade Lake with Nvidia Tesla V100

Количество GPU

1248

Количество vCPU ?

64

Объём RAM ?

384 ГБ

Привязываемая VM ?

☐

Публичный IP-адрес

☐

Исходящий трафик ?

0

 ГБ

Диск – 1 (Загрузочный)

☒ HDD

☐ SSD

Размер диска

3 ГБ

3 ГБ4096 ГБ

Добавить диск

Все цены указаны с НДС.

Итого:

Compute Cloud × 1987 487,98 Р ×

Intel Cascade Lake with Nvidia Tesla v100, 100% vCPU34 449,41 Р

Intel Cascade Lake with Nvidia Tesla v100, GPU898 289,28 Р

Intel Cascade Lake with Nvidia Tesla v100, RAM54 743,04 Р

Стандартное сетевое хранилище (HDD)6,25 Р

Исходящий трафик в интернет0,00 Р

987 487,98 Р

в месяц

Перейти в консоль

<https://cloud.yandex.ru/prices>

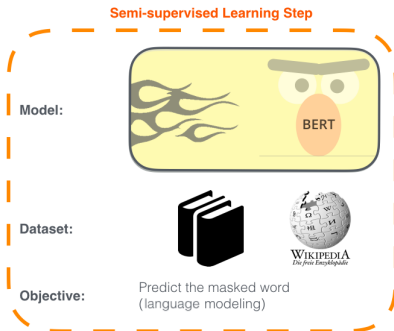
BERT (2018)

- BERT - Bidirectional Encoder Representations from Transformers
- **Идея BERT:** предобучать энкодер из трансформера на двух искусственных задачах:
 - Masked Language Modeling
 - Next Sentence Prediction
- После того, как мы предобучили BERT, мы можем доучивать слои для решения конкретной задачи

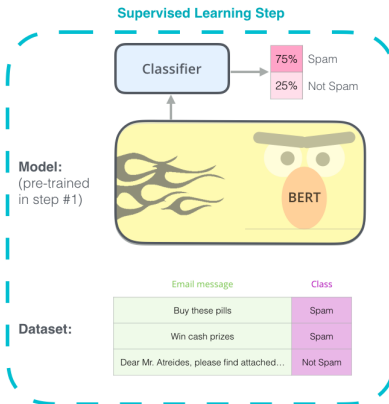
BERT (2018)

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.



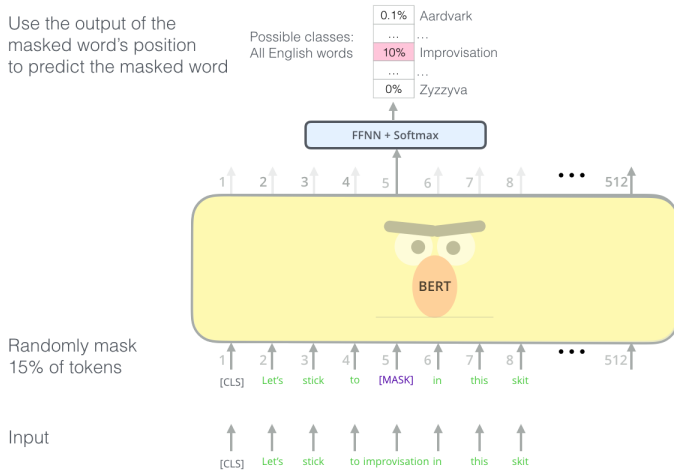
2 - **Supervised** training on a specific task with a labeled dataset.



<http://jalamar.github.io/illustrated-bert/>

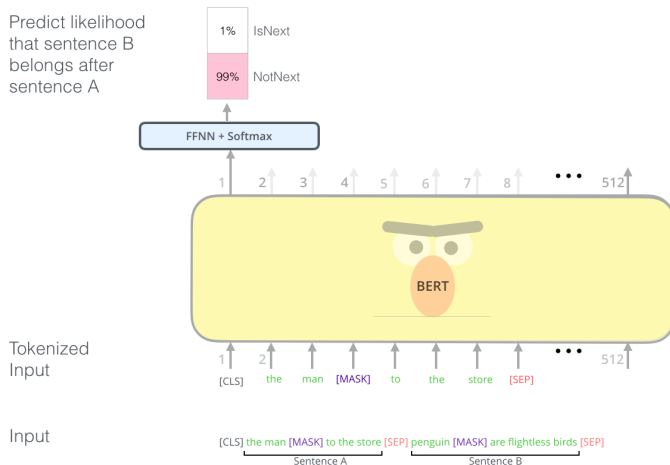
BERT (2018)

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



<http://jalamar.github.io/illustrated-bert/>

BERT (2018)



<http://jalamar.github.io/illustrated-bert/>

Как используем?

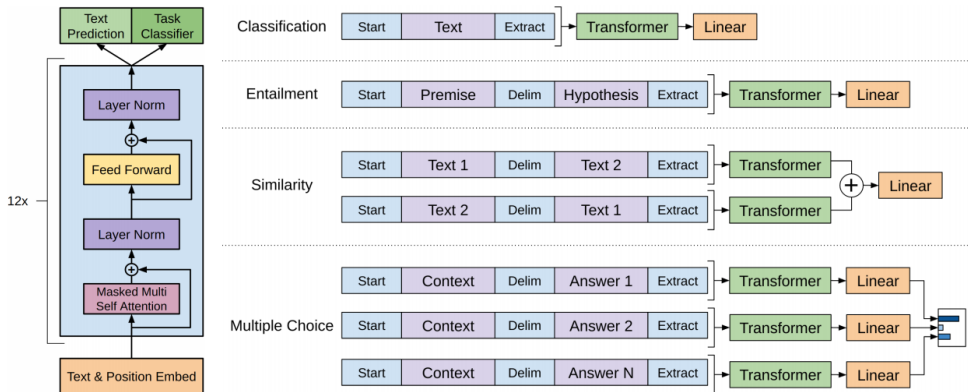


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.