

Глубокое обучение и вообще

Ульянкин Филипп

3 апреля 2021 г.

Посиделка 16: Улица Сезам

Agenda

- Развитие идеи эмбедингов
- Seq2seq модели
- История автоперевода
- RNN и механизм внимания
- Attention is all you need
- Модификации трансформера

Развитие идеи эмбедингов

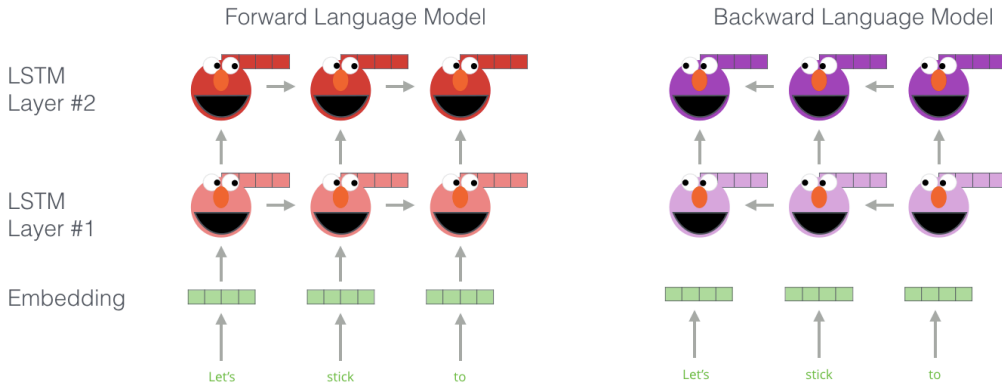
Серия вопросов в зал

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

Embeddings from language models (ELMo)



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



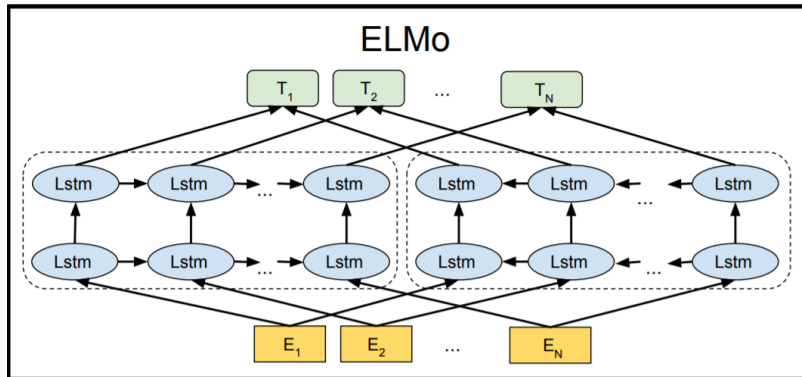
ELMO

- Захватываем контекст предложения через biderictional LSTM
- Модель пытается предсказать следующее слово в предложении

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

ELMo

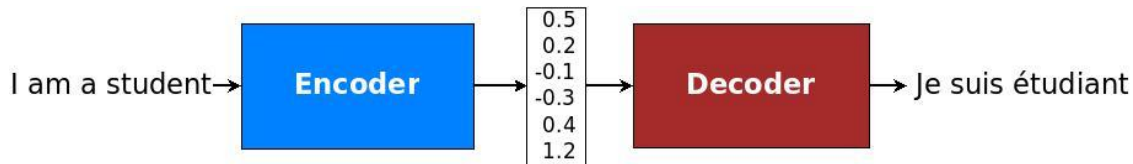
В качестве эмбединга используется вектор $[T, h^l, h^r]$, где T - токены, которые сетка выплёвывает наружу, h_l - итоговое скрытое состояние ячеек при проходе слева направо, h^r - справа налево.



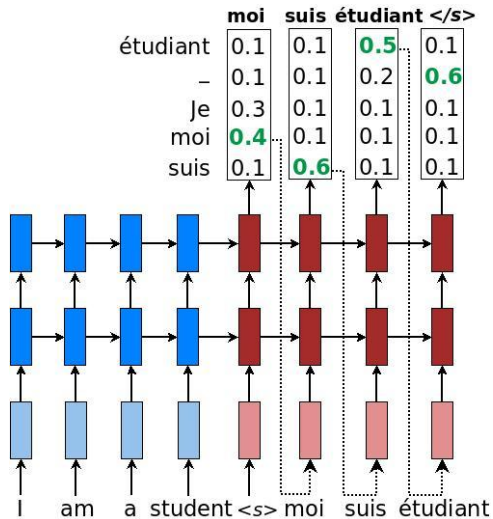
RNN и механизм внимания

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



Автопереводы

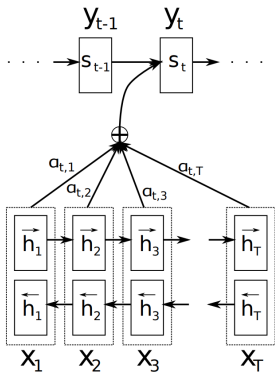


<https://github.com/tensorflow/nmt>

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

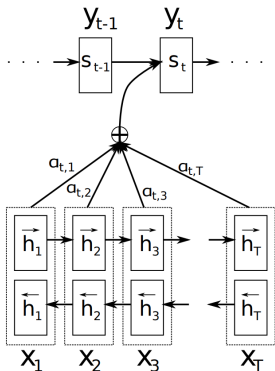
Механизм внимания



- 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 декодировщика

$$h_t^d = g(\hat{y}_{t-1}, h_{t-1}^d, c_t)$$

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

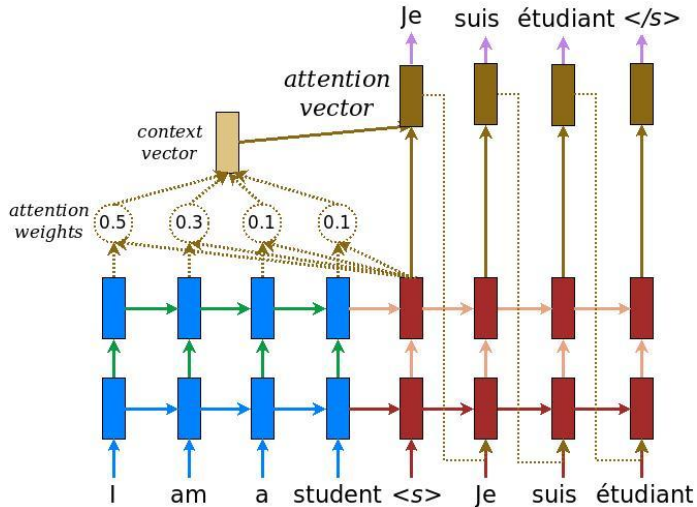
$$\text{sim}(h_j^e, h_{t-1}^d)$$

- Распределение на входных словах:

$$\alpha_{it} = \text{Softmax}(\text{sim}(h_j^e, h_{t-1}^d))$$

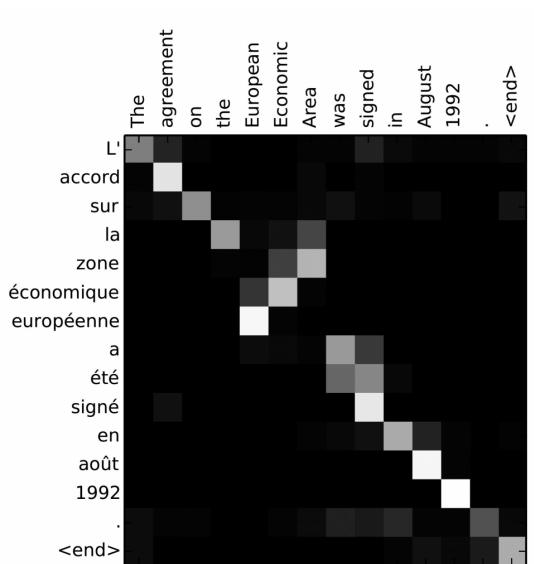
- Пытаюсь угадать какие слова входной последовательности важны: $c_t = \sum_i \alpha_{jt} \cdot h_j^e$

Механизм внимания



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

Механизм внимания



Как посчитать sim?

- Скалярное произведение:

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

- Additive attention:

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

- Multiplicative attention:

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

- W, W_s, W_h - обучаемые параметры
- Многие-многое другие функции из разных модных работ :)

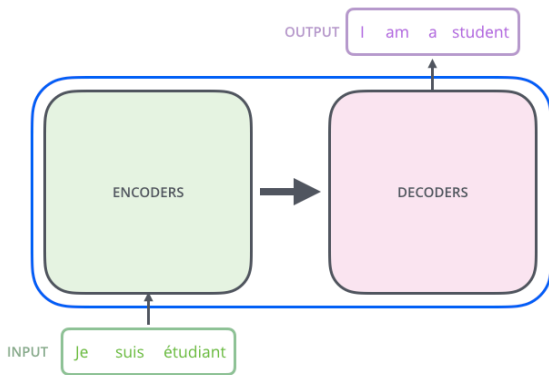
Как сделать seq2seq быстрее без
потери качества?

Attention is All You Need! (2017)

- RNN это очень долго! Всегда, чтобы найти следующий токен, надо знать предыдущий
- Backward pass идёт ещё и через время :(
- Transformer — нейросетевая архитектура для задач seq2seq, основанная исключительно на полносвязных слоях
- Превзошла существовавшие seq2seq архитектуры как по качеству, так и по скорости работы
- Основной элемент — multi-head self-attention

Transformer

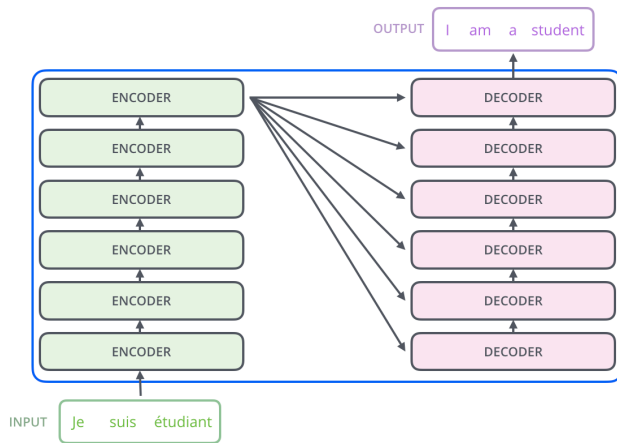
Верхнеуровнего - это просто энкодер и декодер



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

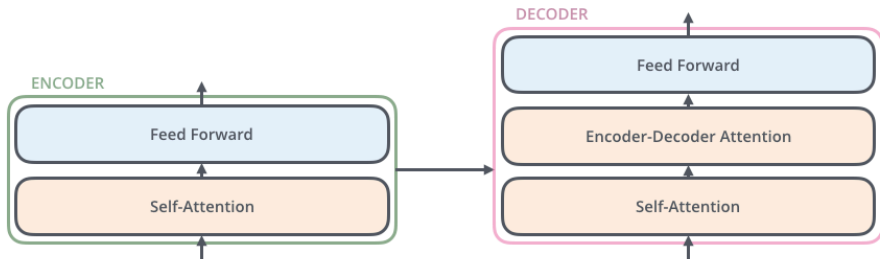
Transformer

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



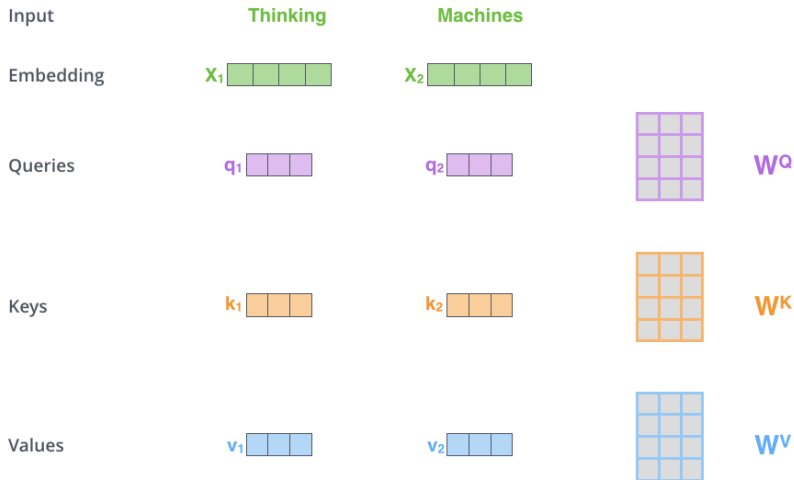
Transformer

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



Абстракции

- Для каждого входного слова считаются три вектора: Query, Key и Value
- Матрицы W^Q , W^K , W^V обучаются вместе с моделью
- Value - то, что мы знаем об этом слове
- Query, Key помогают искать связи между словами, мы ходим по всем словам и пытаемся понять насколько они связаны между собой
- Query - мое текущее слово, Key - мое слово с которым я сравниваю себя

Self-attention

Цель этого слоя — сложить Value с некоторыми весами

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

=

Z

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

Более детально

Input

Embedding

Queries

Keys

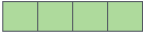
Values

Score

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

Softmax

Thinking

x_1 

q_1 

k_1 

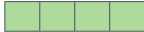
v_1 

$q_1 \cdot k_1 = 112$

14

0.88

Machines

x_2 

q_2 

k_2 

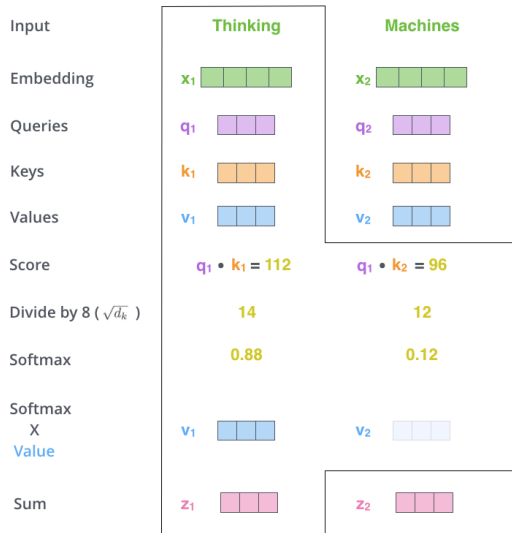
v_2 

$q_1 \cdot k_2 = 96$

12

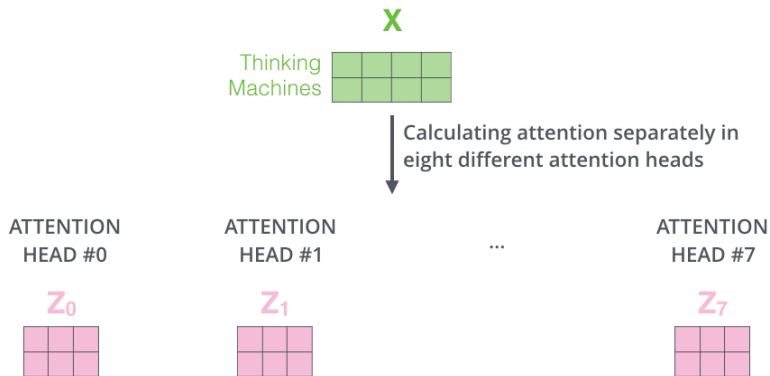
0.12

Более детально



Зверь с кучей голов

Несколько голов обеспечивают разное внимание



Слой целиком

1) This is our input sentence*

Thinking
Machines

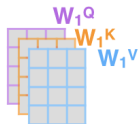
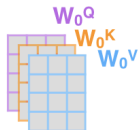
2) We embed each word*



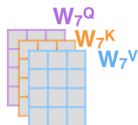
* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



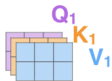
3) Split into 8 heads. We multiply X or R with weight matrices



...



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



...



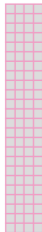
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer



...



W^O

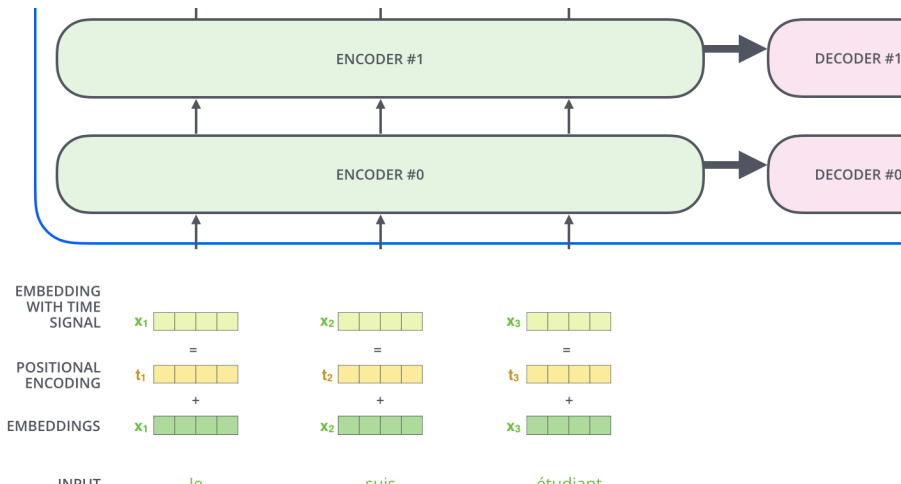


Z

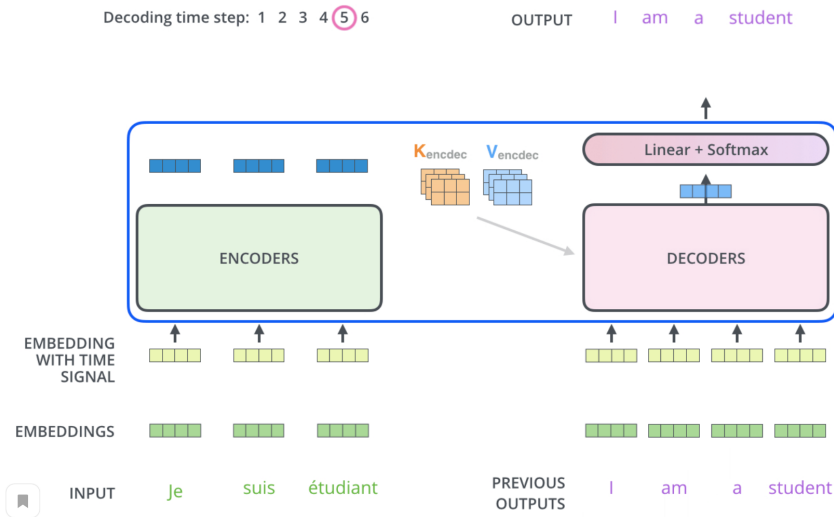


Positional encoding

Для учёта позиции слова в предложении входные эмбединги можно преобразовывать случайным шумом t_i , который зависит от позиции (зашумлённый косинус и тп)



Что происходит в декодере?



Модификации трансформера

Модификации

- 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?

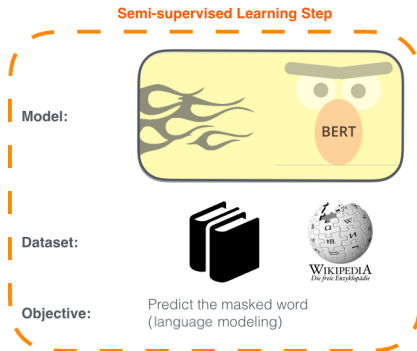
Модификации

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

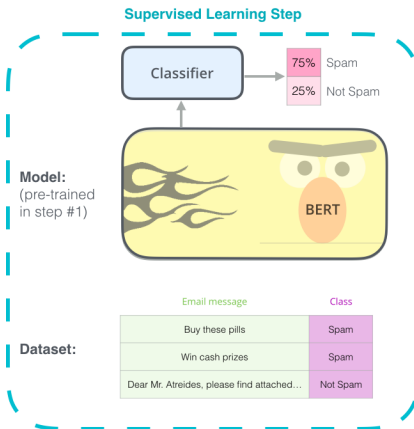
BERT (2018)

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.



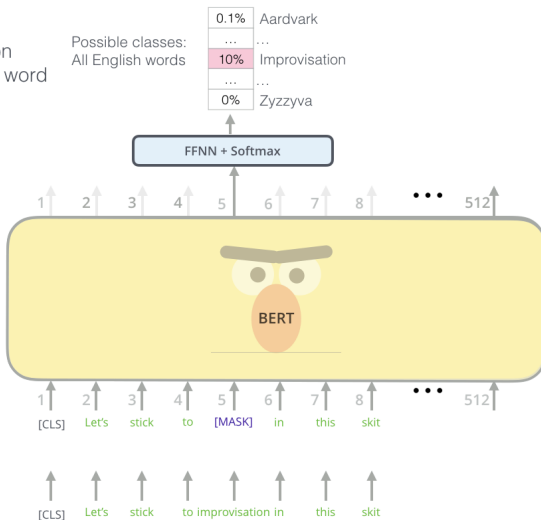
2 - **Supervised** training on a specific task with a labeled dataset.



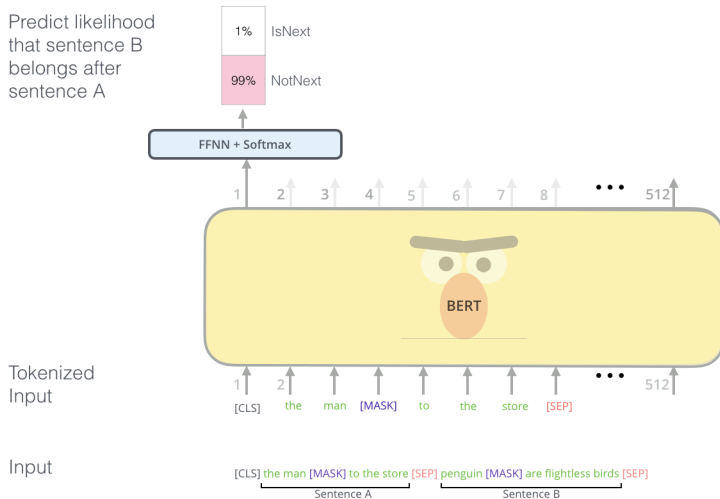
<http://jalamar.github.io/illustrated-bert/>

BERT (2018)

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



BERT (2018)



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

BERT (2018)

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

Как используем?

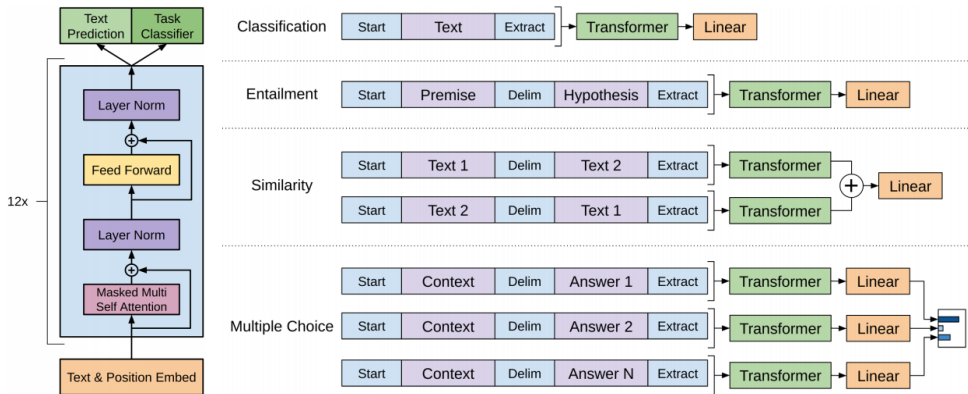


Figure 1: **(left)** Transformer architecture and training objectives used in this work. **(right)** Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.