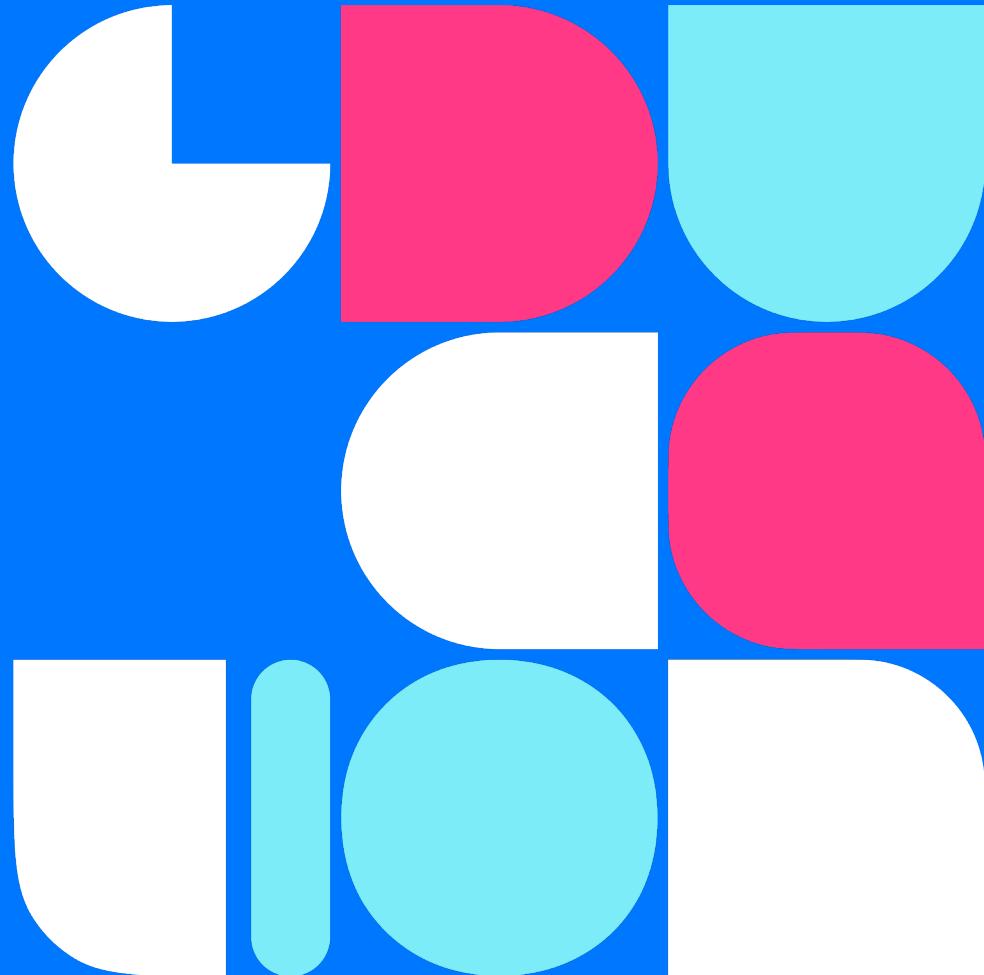


# Лекция 3. BERT

Дмитрий Калашников, 27.02.2024



# План занятия

Небольшое повторение

BERT – как обучаются умные эмбеддинги

Как сделать BERT лучше

BERT на практике

Как привлекать внешние знания для генерации текста, и причем тут BERT

# Восстановим контекст



# Восстановим контекст

1

Как мы получали  
представления для  
слов?

2

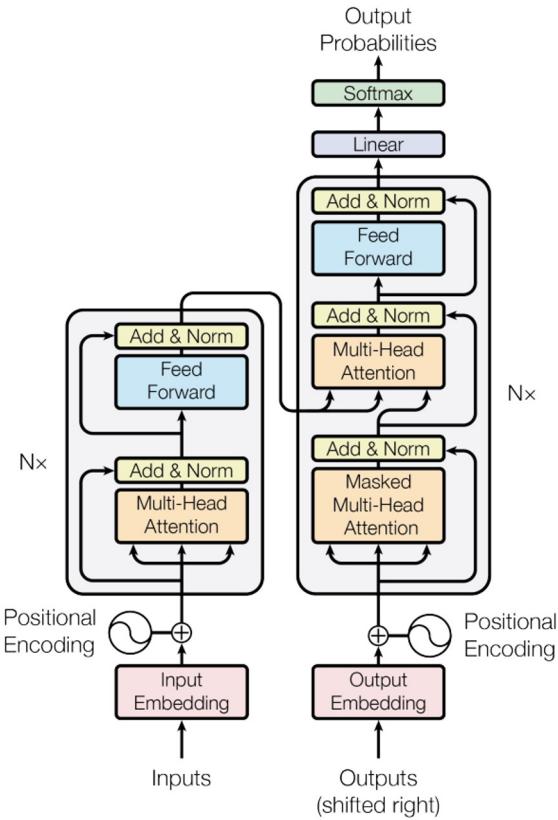
Word2Vec, Fasttext  
Проблема: не  
используем контекст при  
инфереенсе;  
При обучении смотрим  
лишь на малую часть  
последовательности

3

RNN (и дополнения)  
Смотрим на контекст  
последовательно  
Bi-RNN: в две стороны, но  
всё ещё последовательно  
Информация  
последовательно  
забывается

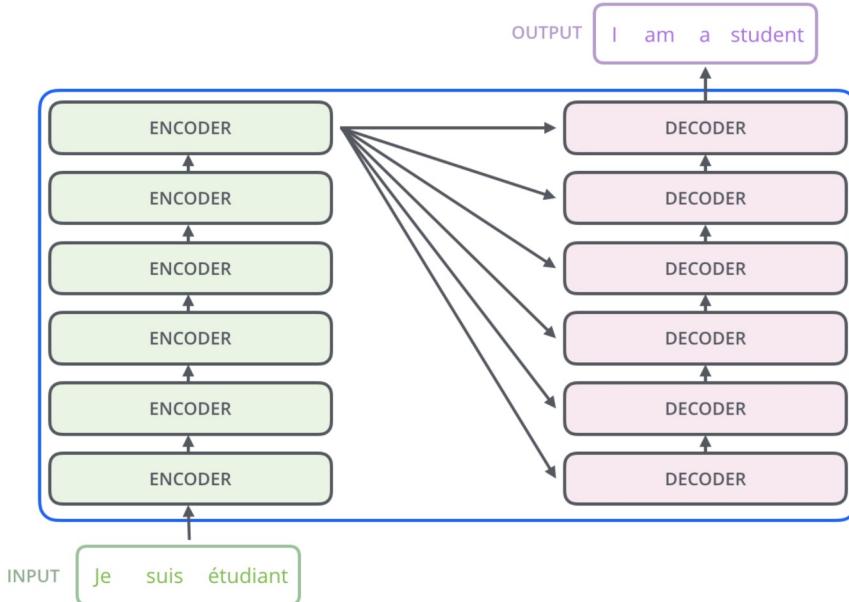
# Восстановим контекст

- Трансформер – стек энкодеров + стек декодеров
- Что подается трансформеру в энкодер?
- Что подается трансформеру в декодер?
- Где в трансформере маскируется последовательность?



# Encoder и decoder

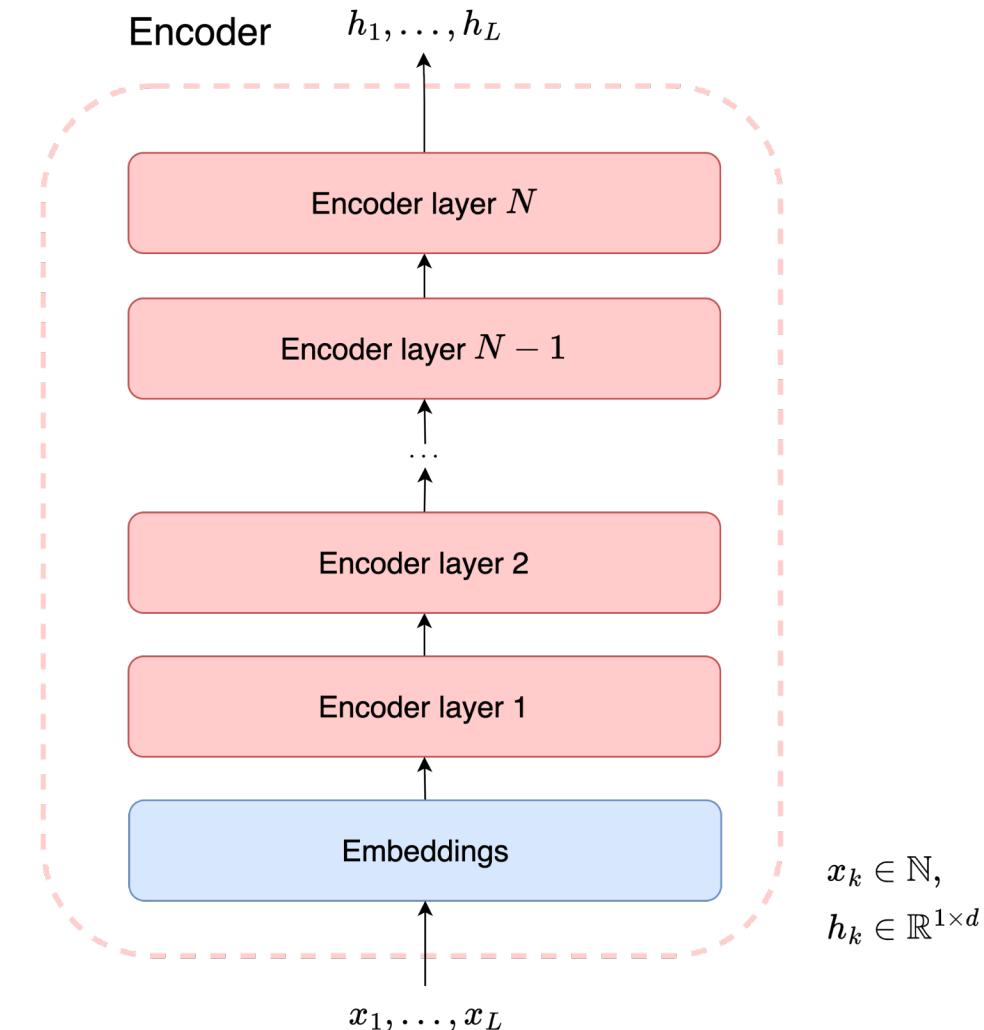
- Хотим качественные эмбеддинги токенов для разных задач
  - классификация текста
  - NER на токенах
- Токены должны узнать всю информацию о контексте
- Какая часть трансформера может приготовить эмбеддинги токенов?



# Устройство кодировщика

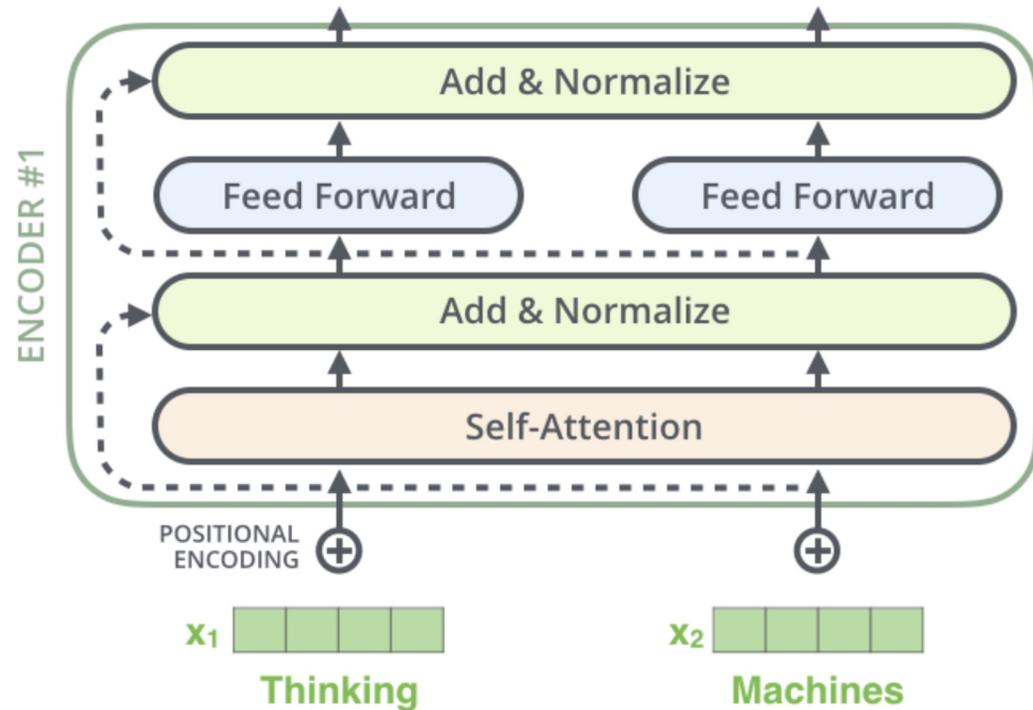
Кодировщик:

- Принимает на вход последовательность токенов
- Вычисляет эмбеддинги входных токенов (слой embeddings)
- Применяет последовательно  $N$  однотипных преобразований (encoder layer)



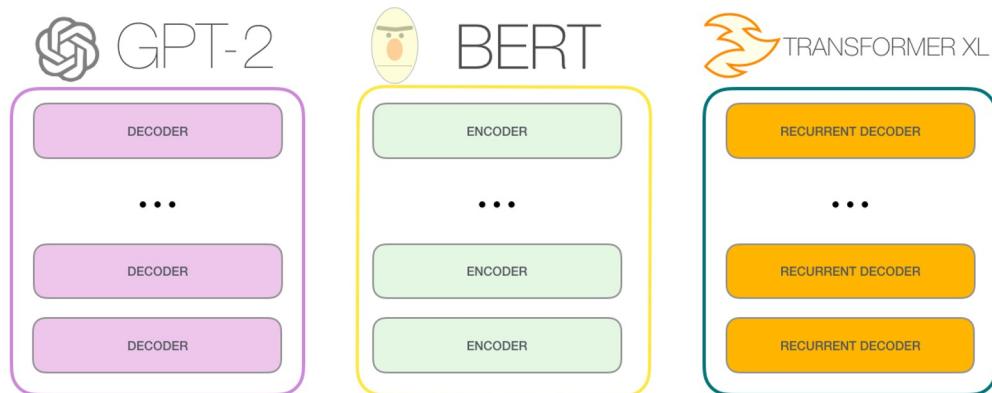
# Блок энкодера

- Каждый слой энкодера формирует более сложное представление исходного токена с учетом всего контекста
- Энкодер видит **всю** последовательность
- Больше энкодеров – выше уровень абстракции представлений (представления “умнее”)



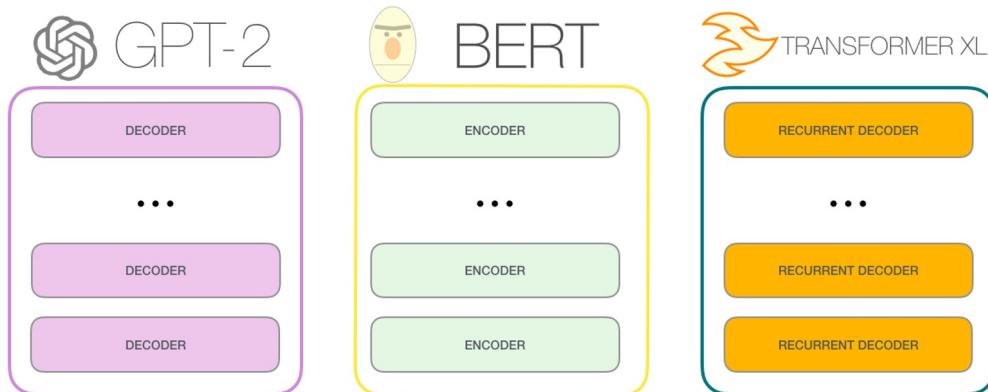
# Энкодеры – для эмбеддингов

- Можем не использовать стек декодеров вообще
- Выкидываем декодеры, обучаем только энкодеры
- BERT – текстовый энкодер



# Декодеры – для LM

- Если наоборот: не интересуют эмбеддинги, а хотим решать задачу LM?
- Аналогично можем выкинуть стек энкодеров
- Обучаем стек декодеров генерировать текст по известному контексту
- Реализовали в GPT
- Но нас пока интересуют только эмбеддинги



# BERT



# Ключевые идеи

Умеем решать supervised-задачи на текстах

- сформулирована задача, например: классификация спама
- есть обучающие данные и разметка под задачу, например: 1 / 0
- обучаем модель, внутри которой формируется некоторое скрытое представление, по которому мы умеем хорошо решать поставленную задачу

Проблемы:

- По полученному скрытому представлению вряд ли получится решить другую задачу
- На каждую задачу - своя модель
- Нужна разметка - это долго, дорого и часто разметка шумная

# Ключевые идеи

Идея 1:

- обучить модель, которая будет "понимать" язык

Идея 2:

- примеров правильного употребления языка очень много: весь Интернет
- можно использовать тексты из интернета как обучающие и как таргет
- тогда не нужна ручная разметка
- можно набрать любое количество подходящих данных, в том числе под конкретный домен



# Ключевые идеи

Главный вопрос: как поставить задачу, на которую модель будем обучать

Вариант 1: задача генерации токена

- по входным токенам предсказать следующий

Вариант 2: задача восстановления токенов

- В исходном тексте удаляем какой-то токен
- С помощью модели восстанавливаем его



# Кто такой этот ваш BERT

- Bidirectional Encoder Representations from Transformers
- Используется для получения представлений (эмбеддингов) токенов
- Из представлений токенов можно получить эмбеддинг текста
- Для чего используются эмбеддинги:
  - Классификация текста / токенов
  - Поиск похожих текстов
  - Экстрактивная суммаризация текста
  - NER



# На какие задачи обучать

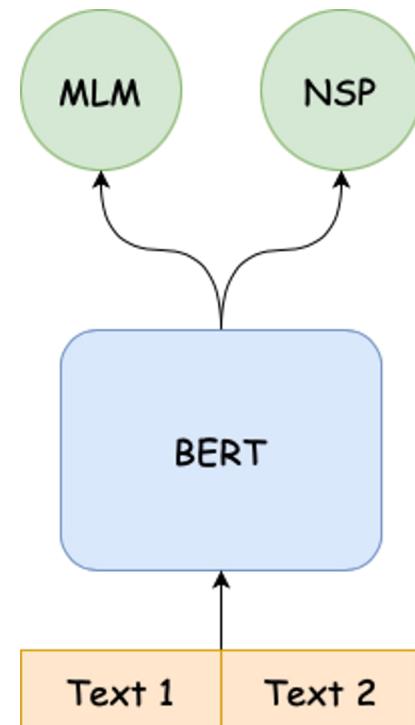
Гипотеза – модель “понимает” язык, если:

- по токенам слева и справа может восстановить пропущенный токен
- может определить, следует ли в тексте одно из двух предложений за другим

Две задачи для обучения:

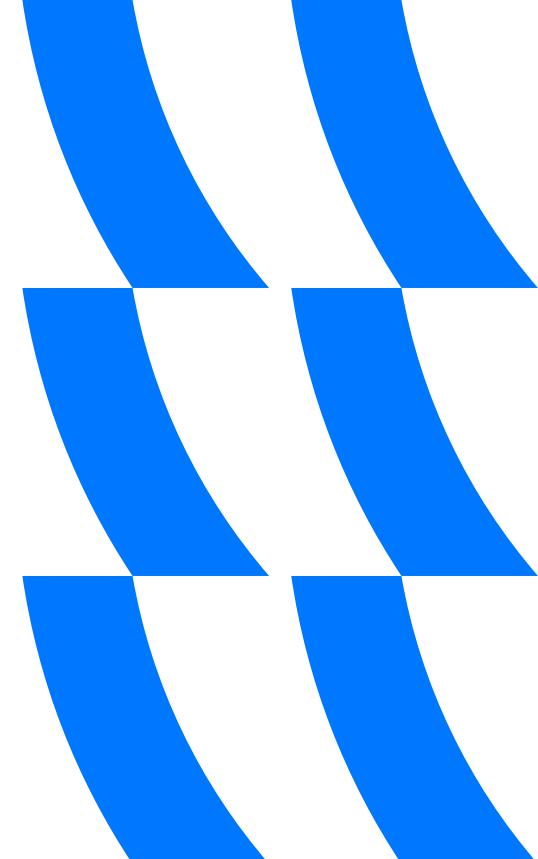
- Masked Language Modeling
- Next Sentence Prediction

Находим огромный корпус текстов, нарезаем на части, подаем в модель, обучаем



# MLM

- На входе: текст
- Заменяем 15% токенов в тексте на спецтокен [MASK]
- Подаем последовательность в кодировщик, получаем представления каждого токена
- Решаем задачу LM
- Лосс – кросс-энтропия



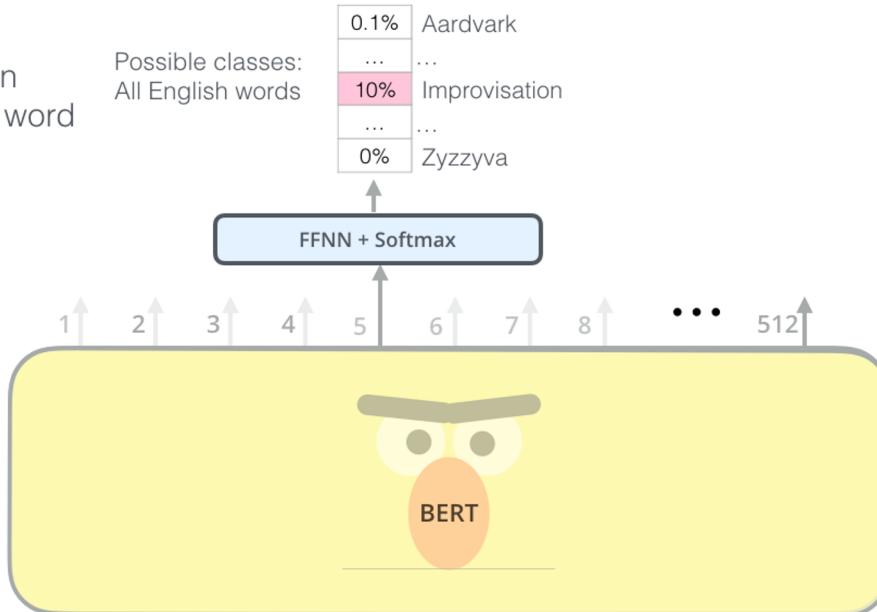
Use the output of the masked word's position to predict the masked word

Possible classes:  
All English words



FFNN + Softmax

Randomly mask  
15% of tokens



Input

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

# MLM

15% максимизируем

- $p=0.8$  на [MASK]
- $p=0.1$  на случайный токен – аугментация
- $p=0.1$  оставляем как есть

Зачем?

- Маска статическая - для каждой последовательности сгенерировали 10 масок перед обучением, во время обучения используем только их
- 15% – оптимальное значение
- Если умеем решать, то понимаем “смысл токенов” – контекст и семантику отдельных токенов
- Ручная разметка не нужна

# NSP

- Классификация, следует ли В за А
- Если умеем решать, то понимаем "смысл текста": логику, смысл повествования
- Ручная разметка не нужна
- Соотношение классов: 50/50
- Авторы заметили, что помогает в задачах QA и NLI

LLM - смысл моей жизни. Не знаю, как можно жить без LLM

Какой anchor\_id в итоге использовать? Тут написано 120, там написано 50

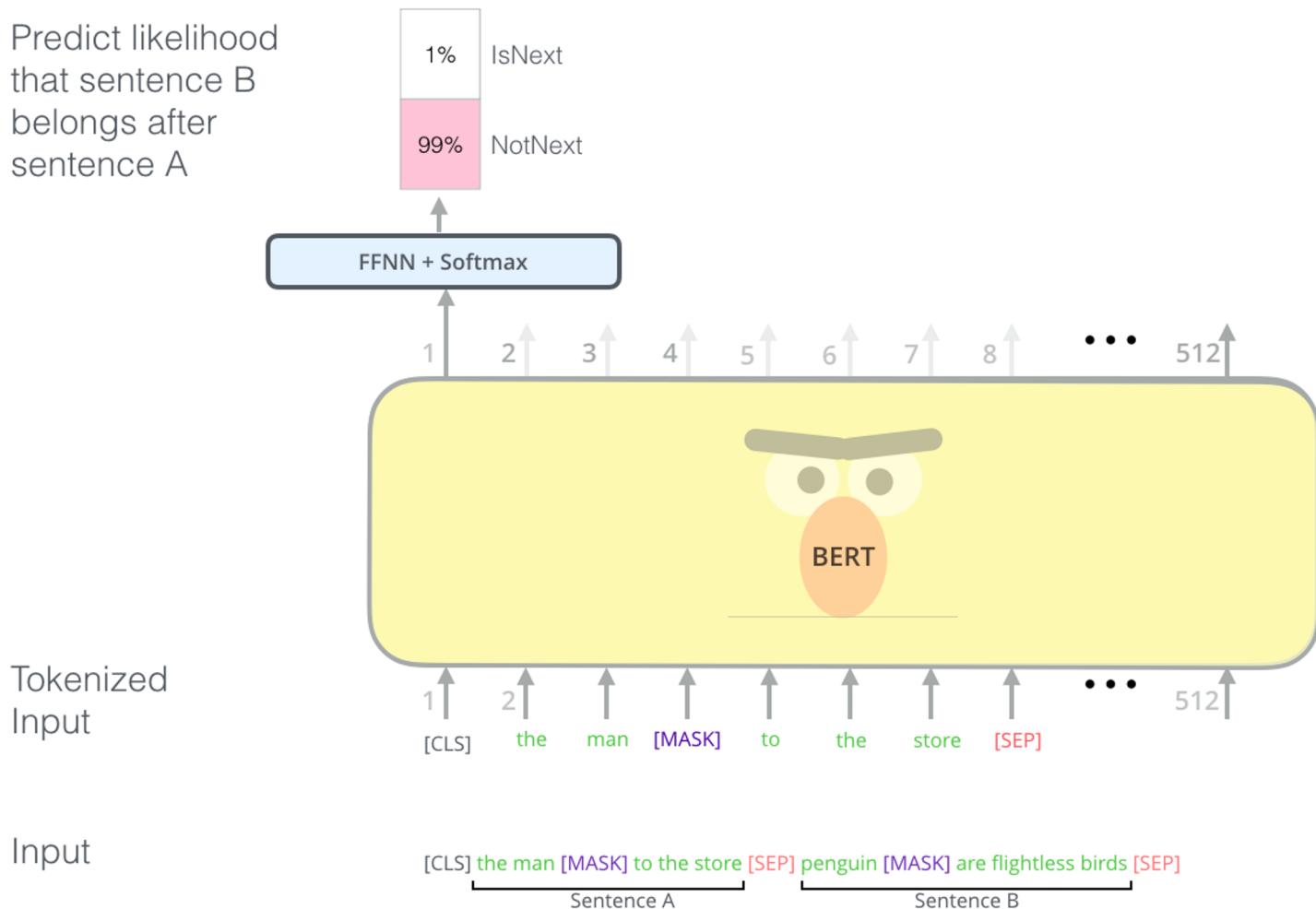


LLM - смысл моей жизни. [SEP] Не знаю, как можно жить без LLM

LLM - смысл моей жизни. [SEP] Тут написано 120, там написано 50



Predict likelihood  
that sentence B  
belongs after  
sentence A



# Спецтокены

[CLS]

- Где: в начале
- Зачем: для задачи NSP; часто - для SFT и как эмбеддинг всего текста

[MASK]

- Где: в соответствии со статической маской
- Для задачи MLM

[SEP]

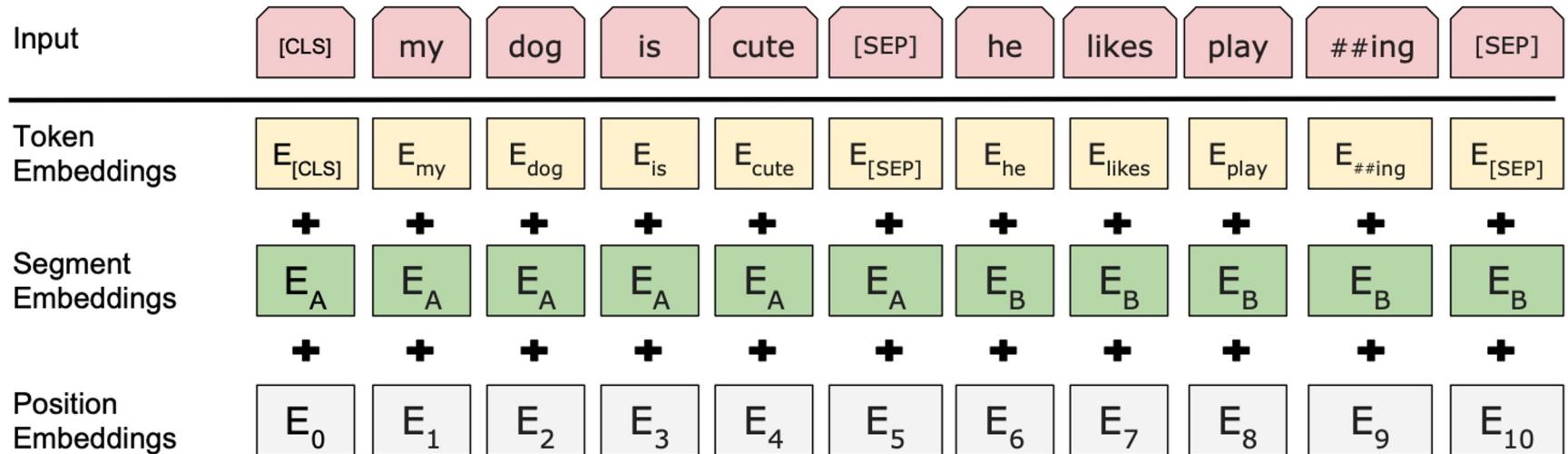
- Где: в середине и конце значимой последовательности
- Зачем: для задачи NSP

[PAD]

- Где: в конце
- Дополнить до фиксированной длины предложения, чтобы выравнить длины всех последовательностей на входе и обрабатывать батчами
- В attention не учитываем, не маскируем

# Input

- Input = pos\_emb + segment\_emb + token\_emb
- Все эмбеддинги обучаемые



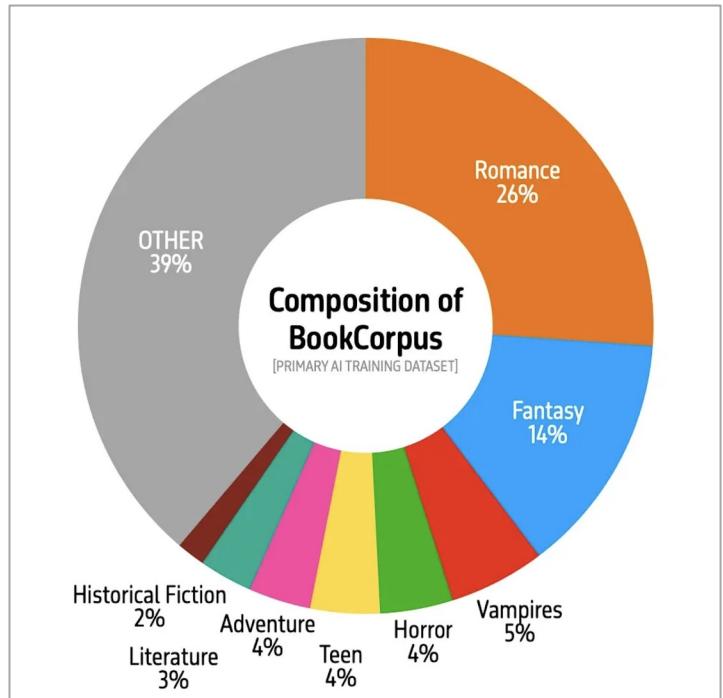
# Датасет

Данные: Википедия + Bookcorpus

- Корпус: 2.5B + 0.8B words
- Эквивалент: 5500 книг "Война и мир"

Batch size: 131\_072 слов

- 1024 текста в 128 слов
- 256 текстов в 512 слов



# Архитектура

## BERT-BASE

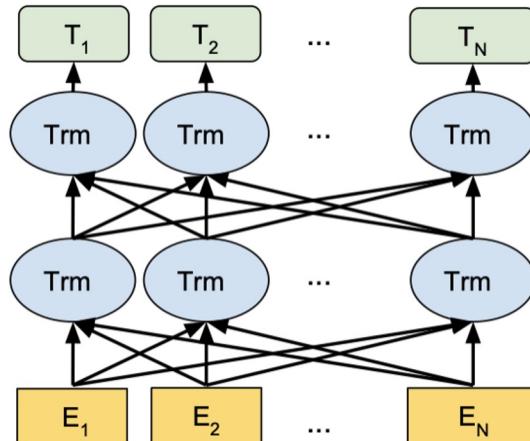
- 12 слоев, 768 скрытое состояние, 12 голов
- Какой размерности каждое представление токена на выходе?

## BERT-Large

- 24 слоя, 1024 скрытое состояние, 16 голов

## Токенизация: WordPiece

- Размер словаря: 30\_000 токенов



# Обучение

## AdamW

- lr=1e-4 – сравни с трансформером
- Warmup (через 10\_000 шагов к макс точке) + linear decay

## Время обучения

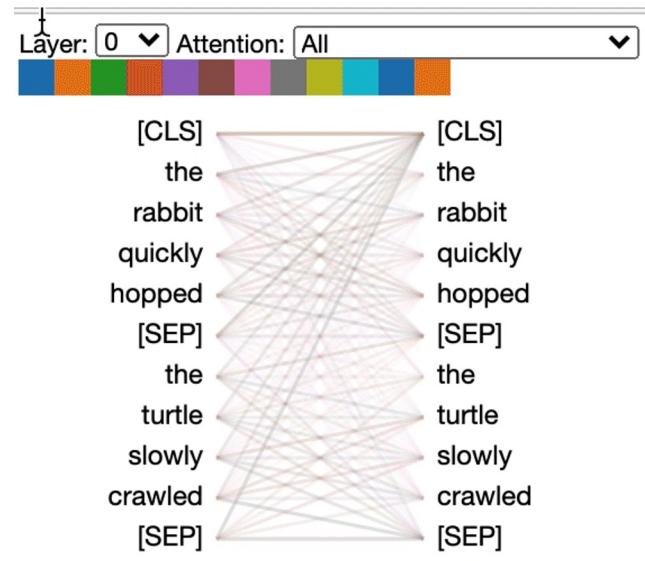
- 1M шагов
- 40 эпох
- B = 256
- T = 128 токенов первые 90% обучения
- T = 512 токенов последние 10% обучения

Обучение: 4 дня, 4x4 / 8x8 ТРУ

# BERT

## Bidirectional Encoder Representations from Transformers

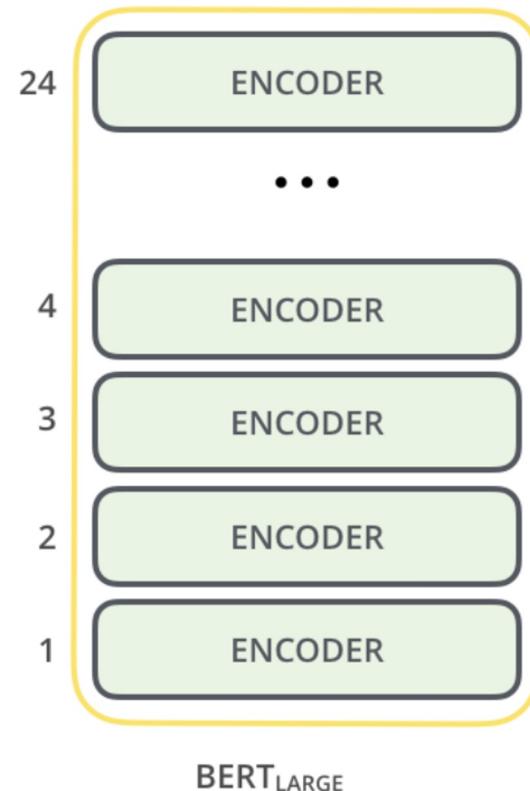
- Двусторонний контекст – для каждого токена механизм внимания смотрит на все токены
  - Равноправно на все токены
  - Bi-RNN – не равноправно за счет забывания RNN
- Какая цена такой двунаправленности?
  - Сложность относительно длины последовательности



# BERT

## Bidirectional Encoder Representations from Transformers

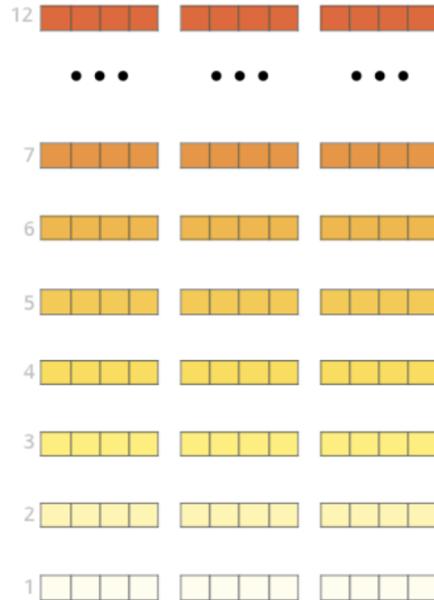
- Обучаем и используем только стек энкодеров
- В трансформере веса энкодера обновлялись за счет задачи LM через декодер
- В BERT веса обновляются за счет задач MLM и NSP



# BERT

## Bidirectional Encoder Representations from Transformers

- Используем как автокодировщик – получаем эмбеддинги
- На вход словарные представления токенов, на выходе – “умные” представления с учетом контекста для тех же токенов



Help

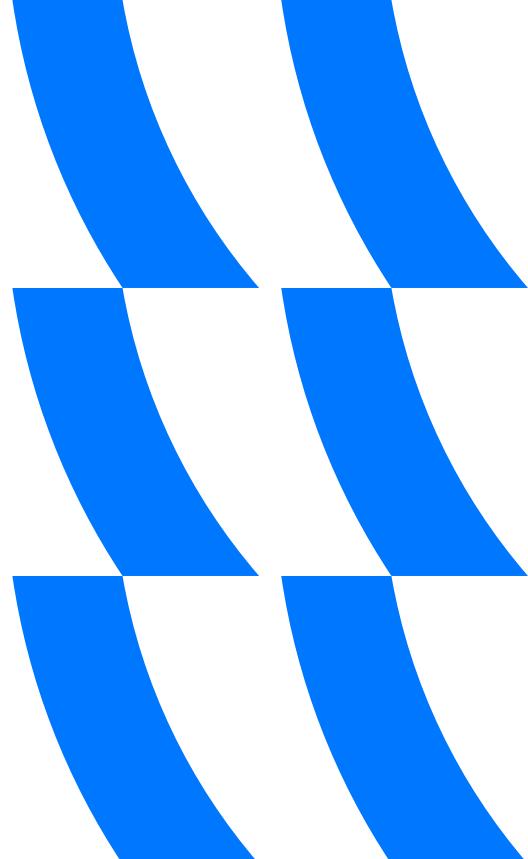
Prince

Mayuko

## В чём вклад статьи

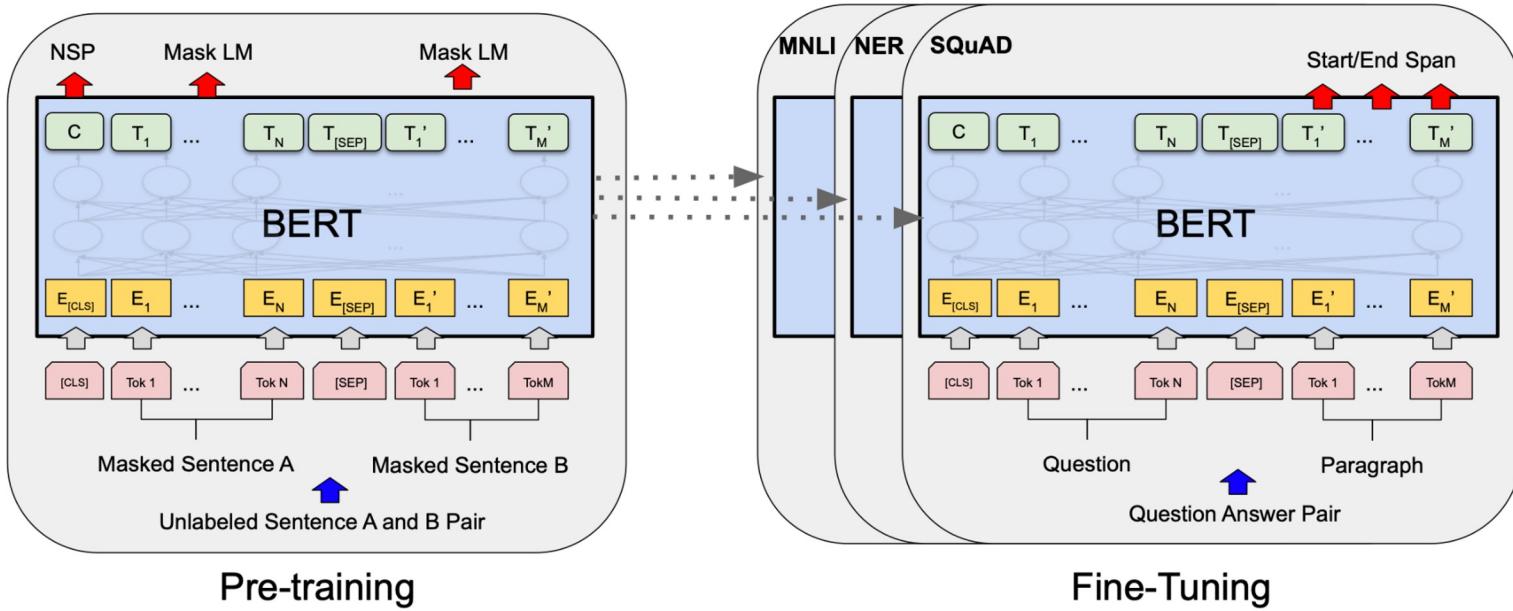
Придумали как обучать качественные представления:

- С учетом полноценного двустороннего контекста
  - спасибо трансформеру
- Без даталиков
  - спасибо Masked LM
- Без размеченных данных
  - спасибо MLM, NSP и Интернету



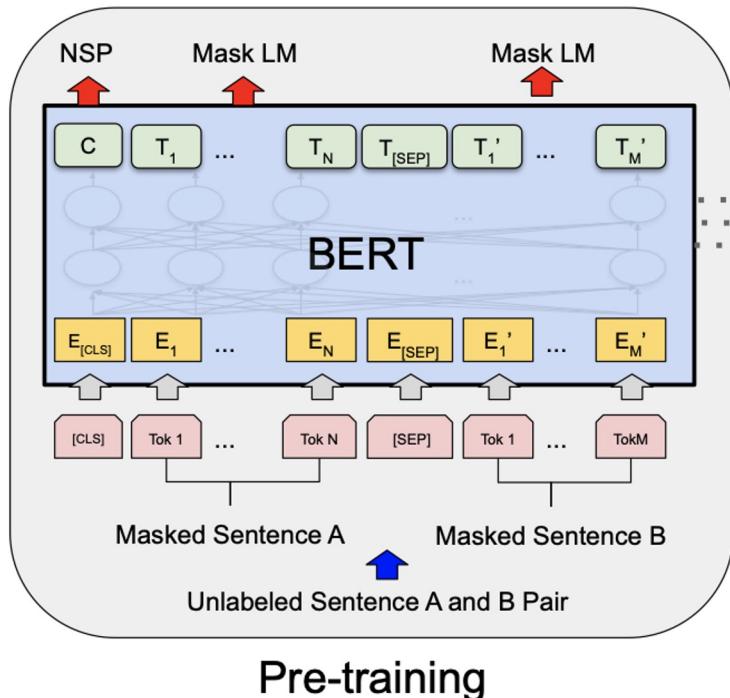
# Pre-training + Fine-tuning

- Идея Pre-training: заложить в модель максимум знаний о мире
- Идея Fine-tuning: научить модель извлекать эти знания для конечной задачи



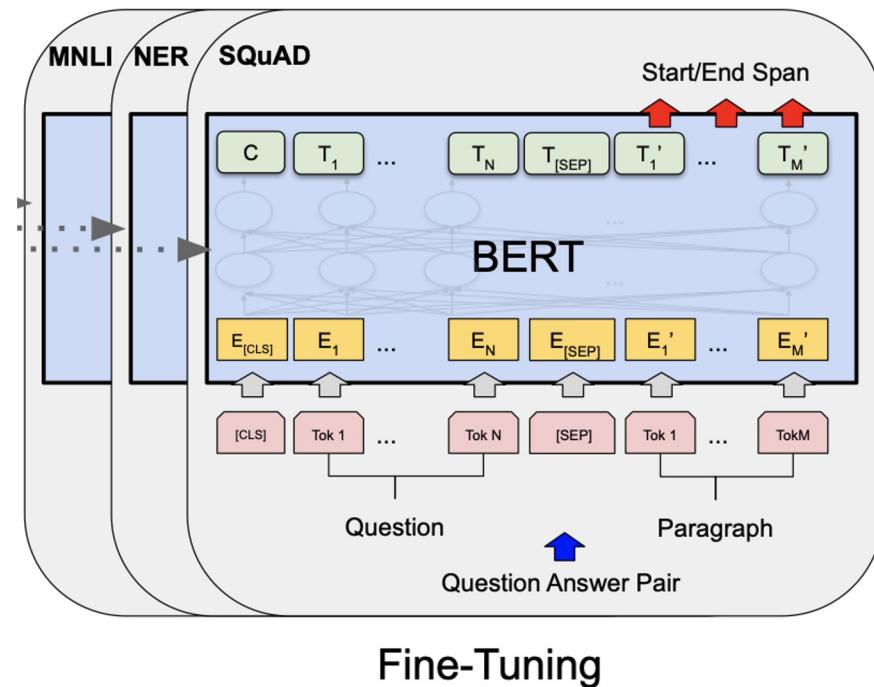
# Детали Pretrain

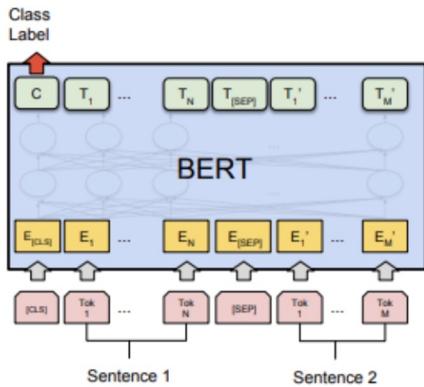
- Одновременно ли обучаем MLM и NSP?
- Чем плох NSP?
  - В реальности соседние предложения редко являются однологическим следствием другого – связь проявляется только на некотором расстоянии
- Чем плох MLM?
  - сдвиг входного распределения относительно инференса, во время которого мы ничего не маскируем



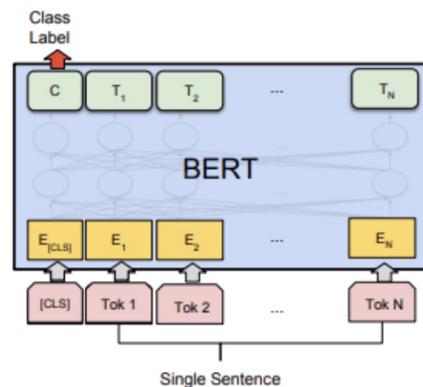
# Fine-tuning

- Pretrain – долго и дорого обучали на огромном корпусе текстов
- После претрэйна BERT умеет извлекать из текста качественные признаки
- Такую модель можно недорого дообучить на конечную задачу (например, классификация спама)

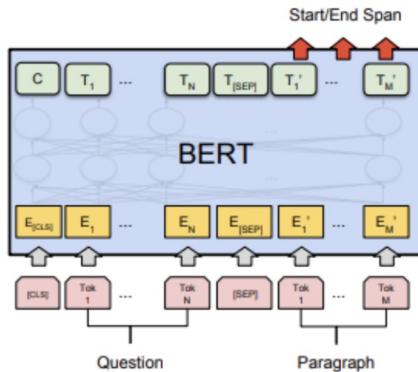




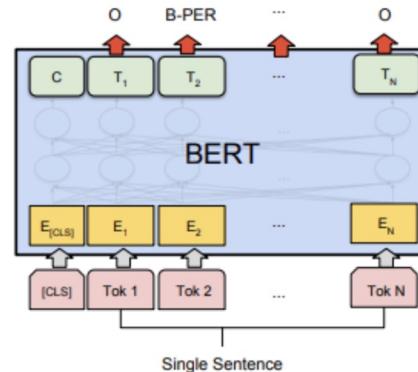
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(b) Single Sentence Classification Tasks:  
SST-2, CoLA

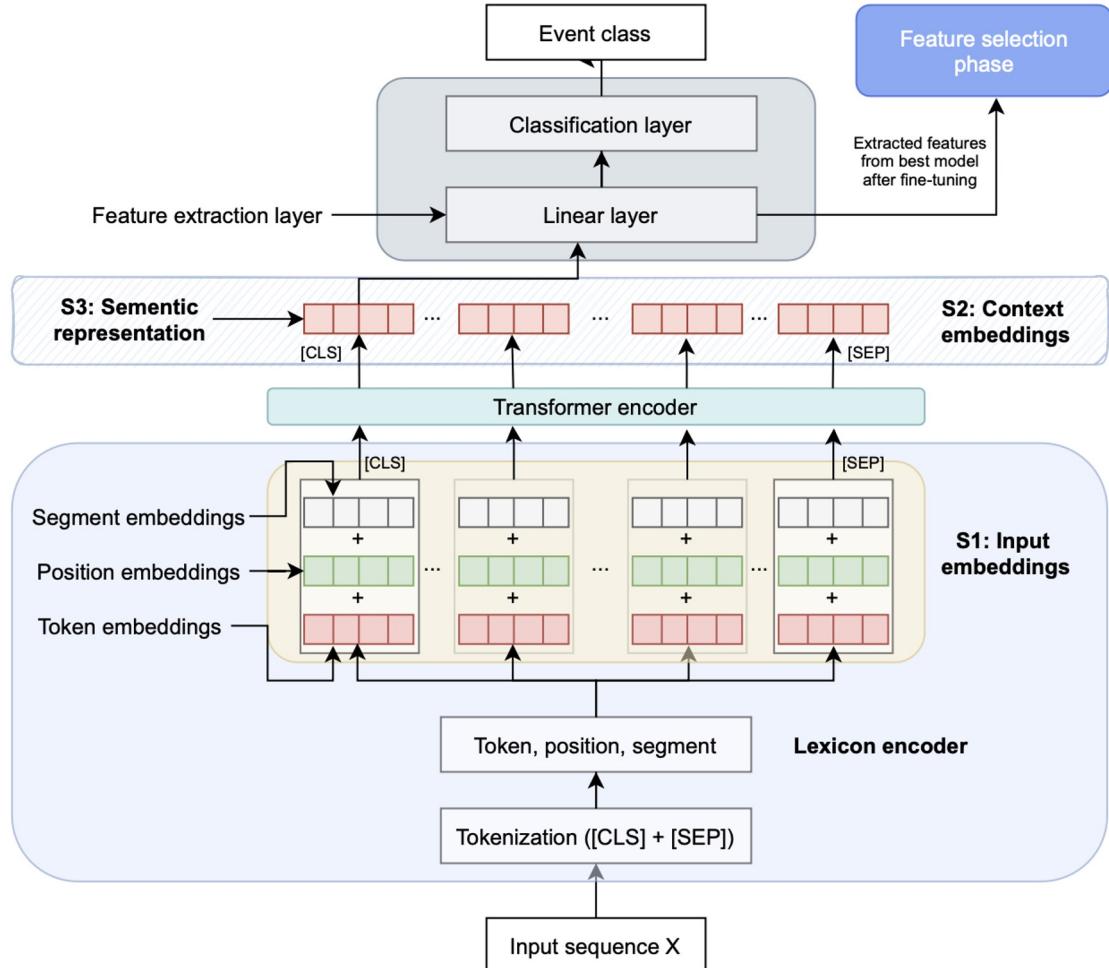


(c) Question Answering Tasks:  
SQuAD v1.1



(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# Fine-tuning



Источник:

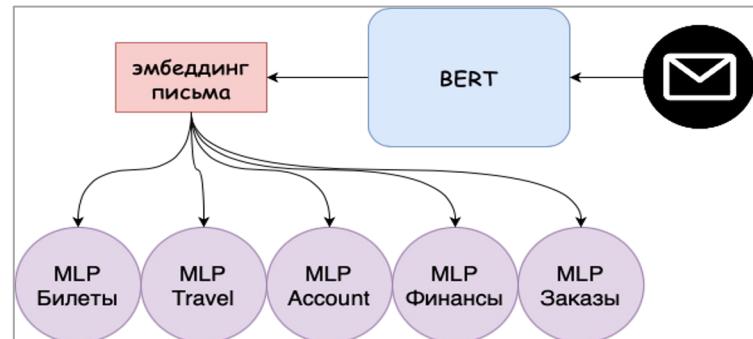
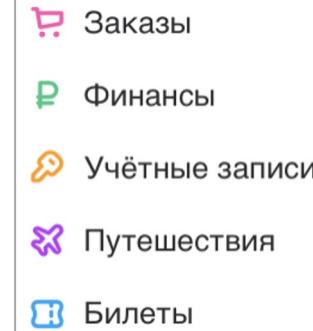
[https://www.researchgate.net/publication/358239462\\_Improving\\_Crisis\\_Events\\_Detection\\_Using\\_DistilBERT\\_with\\_Hunger\\_Games\\_Search\\_Algorithm](https://www.researchgate.net/publication/358239462_Improving_Crisis_Events_Detection_Using_DistilBERT_with_Hunger_Games_Search_Algorithm)

# Fine-tuning

Можно заморозить веса и обучать только доп голову

- Для прода удобно – один экстрактор признаков, много голов под задачи
- Быстрее и удобнее дообучать модель под независимые задачи: data shift, target drift
- Пример: категории писем

Можно делать полный FT



# GLUE

- GLUE - general language understanding
- Large лучше Base – неочевидно, тк в теории могли переобучиться
- Base vs Large: 110M vs 300M параметров
- Для сравнения: в одной из версий генеративной модели Llama – 7B параметров

| System                | MNLI-(m/mm)<br>392k | QQP<br>363k | QNLI<br>108k | SST-2<br>67k | CoLA<br>8.5k | STS-B<br>5.7k | MRPC<br>3.5k | RTE<br>2.5k | Average<br>- |
|-----------------------|---------------------|-------------|--------------|--------------|--------------|---------------|--------------|-------------|--------------|
| Pre-OpenAI SOTA       | 80.6/80.1           | 66.1        | 82.3         | 93.2         | 35.0         | 81.0          | 86.0         | 61.7        | 74.0         |
| BiLSTM+ELMo+Attn      | 76.4/76.1           | 64.8        | 79.8         | 90.4         | 36.0         | 73.3          | 84.9         | 56.8        | 71.0         |
| OpenAI GPT            | 82.1/81.4           | 70.3        | 87.4         | 91.3         | 45.4         | 80.0          | 82.3         | 56.0        | 75.1         |
| BERT <sub>BASE</sub>  | 84.6/83.4           | 71.2        | 90.5         | 93.5         | 52.1         | 85.8          | 88.9         | 66.4        | 79.6         |
| BERT <sub>LARGE</sub> | <b>86.7/85.9</b>    | <b>72.1</b> | <b>92.7</b>  | <b>94.9</b>  | <b>60.5</b>  | <b>86.5</b>   | <b>89.3</b>  | <b>70.1</b> | <b>82.1</b>  |

# Извлечение признаков

Как получать эмбеддинг предложения

- Эмбеддинг CLS-токена
  - из последнего блока
  - агрегация выходов нескольких блоков
- Усреднение/сумма
- Случайная проекция и усреднение/сумма
- SentenceBERT

Полезная для задачи информация размазана по разным слоям

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

For named-entity recognition task CoNLL-2003 NER

|                                    |  | Dev F1 Score |
|------------------------------------|--|--------------|
| 12                                 |  |              |
| • • •                              |  |              |
| 7                                  |  |              |
| 6                                  |  |              |
| 5                                  |  |              |
| 4                                  |  |              |
| 3                                  |  |              |
| 2                                  |  |              |
| 1                                  |  |              |
| <b>Help</b>                        |  |              |
| <b>First Layer</b>                 |  | <b>91.0</b>  |
| <b>Last Hidden Layer</b>           |  | <b>94.9</b>  |
| <b>Sum All 12 Layers</b>           |  | <b>95.5</b>  |
| <b>Second-to-Last Hidden Layer</b> |  | <b>95.6</b>  |
| <b>Sum Last Four Hidden</b>        |  | <b>95.9</b>  |
| <b>Concat Last Four Hidden</b>     |  | <b>96.1</b>  |

# SentenceBERT

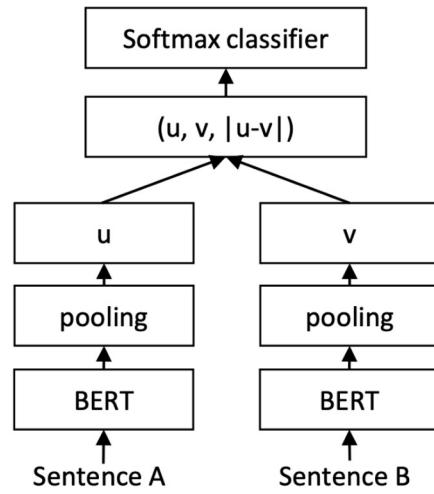


Figure 1: SBERT architecture with classification objective function, e.g., for fine-tuning on SNLI dataset. The two BERT networks have tied weights (siamese network structure).

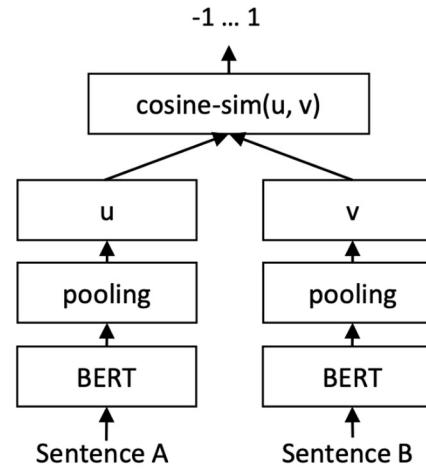


Figure 2: SBERT architecture at inference, for example, to compute similarity scores. This architecture is also used with the regression objective function.

# Как сделать BERT лучше



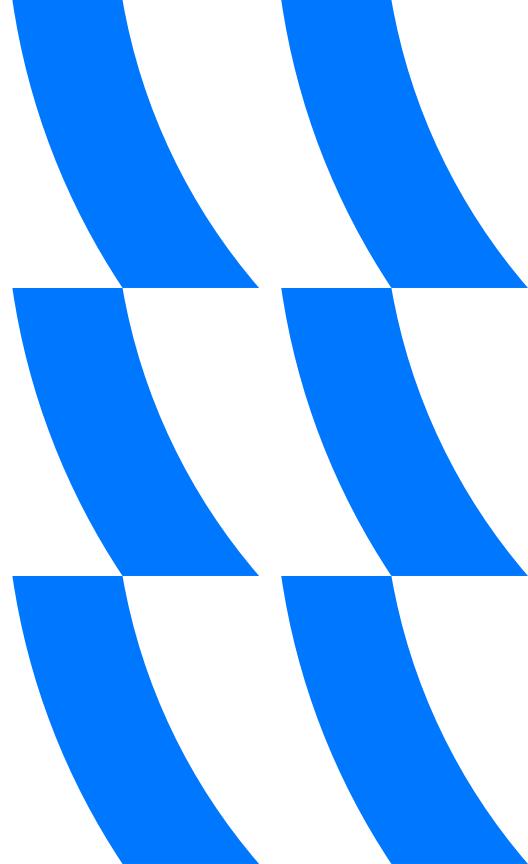
# Помимо BERT

- RoBERTa
- ALBERT
- ELECTRA
- DeBERTa
- DistillBERT
- SpanBERT
- XLNet
- ERNIE
- Transformer-XL
- T5
- BART



# RoBERTa

- Аккуратно обучили архитектуру BERT и сильно улучшили качество
- Данные. Обучать:
  - дольше
  - с большими батчами: 256 -> 8к
  - на большем объеме: 16ГБ -> 160ГБ
  - более длинные предложения: 512 токенов во время всего трейна
- Убрали задачу NSP – не влияет на качество
- Динамическое маскирование вместо статического
- BPE-токенизация, юникод -> байты



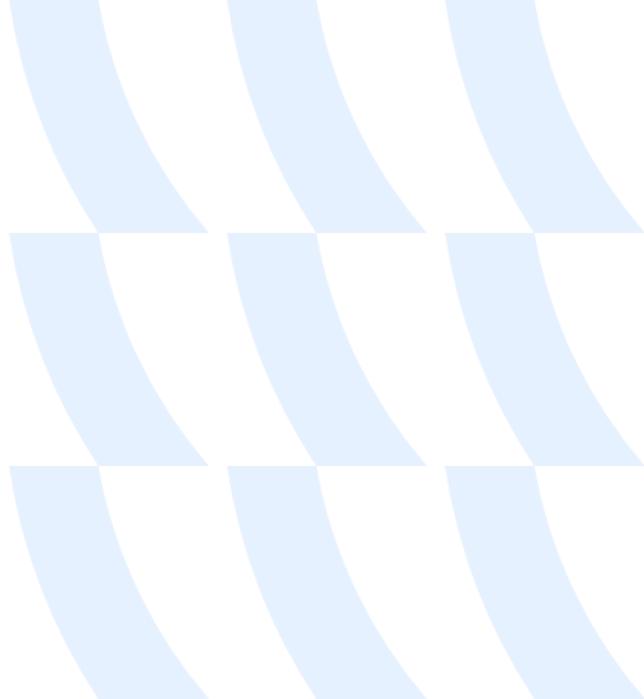
# Еще идеи для BERT-моделей

- Маскировать не токен, а последовательность токенов
  - SpanBERT
- Относительные позиционные эмбеддинги вместо абсолютных
  - DeBERTa
- Привлечение внешних знаний о сущностях в тексте
  - KnowBERT, ERNIE
- Не ломать входное распределение – вместо маскирования токенов генерировать их, и для каждого определять, исходный это токен или сгенерированный
  - ELECTRA
- SOP: правильный или неправильный порядок соседних предложений
  - ALBERT

# Как сделать BERT поменьше

## ALBERT — a Lite BERT

- Обучается с нуля
- в 18 раз меньше параметров, в 1.7 раз быстрее обучается
- Шаринг параметров во всех слоях энкодера
- Факторизуем массивную часть модели: матрицу эмбеддингов  $|V| \times h$
- Превосходит по качеству BERT, RoBERTa и др



# Как сделать BERT поменьше

## DistillBERT

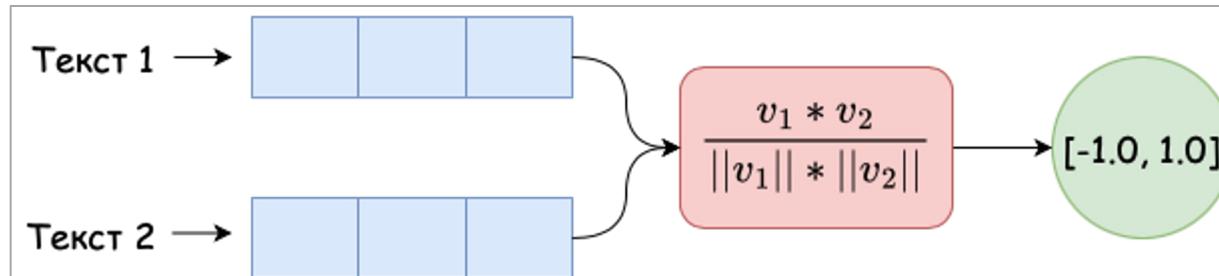
- Опирается на уже обученную модель
- На 40% меньше, 60% быстрее, 97% исходного качества
- Ученик – учитель
- Что берем от учителя:
  - каждый второй энкодер, 12 -> 6 слоев
  - учимся на его скоры
  - учим косинусную близость скрытых состояний на выходе соответствующих слоев
- $\text{Loss} = \text{MLM} + \text{DistillLoss} + \text{HiddenStateCosSim}$
- Часто используемый на практике подход

# BERT как инструмент для LLM



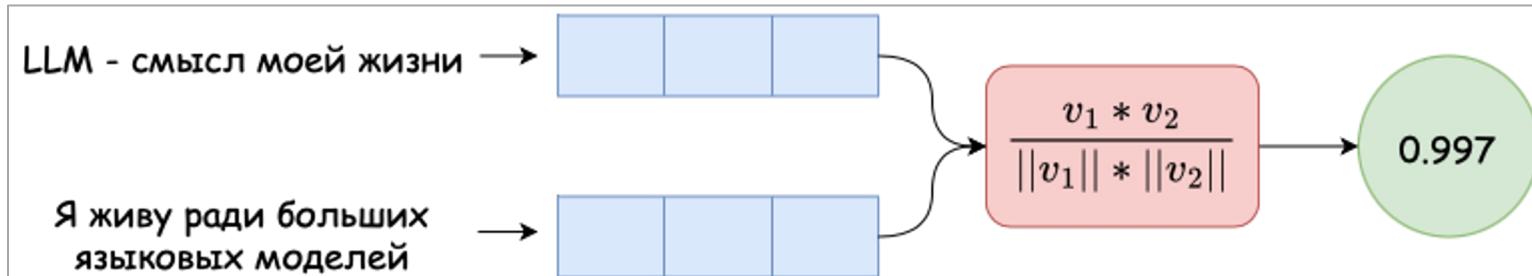
# Поиск семантически похожих писем

- Контекстуальные текстовые BERT-эмбеддинги позволяют осуществлять поиск семантически похожих текстов
- Поиск семантически похожих текстов сводится к поиску близких точек в векторном пространстве эмбеддингов
- Близость, как правило, рассматривается косинусная



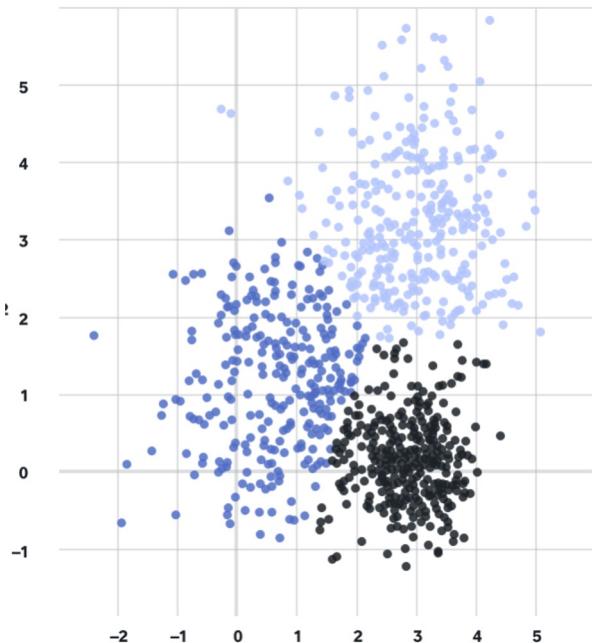
# Поиск семантически похожих писем

- Раз векторное расстояние отражает семантическую близость, можем строить семантические кластера, в которых всех тексты семантически похожи друг на друга
- Также можем выделять семантические дубликаты – тексты с одинаковой семантикой, но, возможно, по-разному сформулированные



# Поиск семантически похожих писем

- Качество языковых моделей сильно зависит от качества данных, на которых они обучаются
- Поэтому важная часть задачи построения языковой модели – подготовка данных:
  - кластеризация и анализ распределения данных
  - найти в трейне причину FP/FN на тесте
  - прочистка дубликатов
  - просеивание данных, увеличение разнообразности данных
  - классификация и удаление отдельных тем



# Retrieval-augmented generation



# Параметрическая и непараметрическая память

- Рассматриваем задачу текстовой генерации, ответы на вопросы (QA)
- Параметрическая память – знания внутри параметров языковой модели (GPT)
- Непараметрическая память – внешние знания, например, дамп Википедии
- В LM заложены только те знания, которые были в обучающей выборке => могут устаревать => модель может галлюцинировать
- Выход: условная генерация при условии релевантных знаний

❖ Кто президент Перу

Закрыть Esc

В настоящее время президентом Перу является Луис Арсе. Он был избран на должность президента в 2016 году и вступил в должность в 2017 году.

❖ Кто президент Перу?

Дина Эрсilia Болуárте Сегáрра (исп. Dina Ercilia Boluarte Zegarra; род. 31 мая 1962, Перу) — перуанская юристка и политик. С 7 декабря 2022 года — действующий президент Перу[6]. Она стала первой женщиной на этой должности[7]. С 2021 по 2022 год она занимала пост вице-президента, а также министра развития и социальной интеграции Перу.

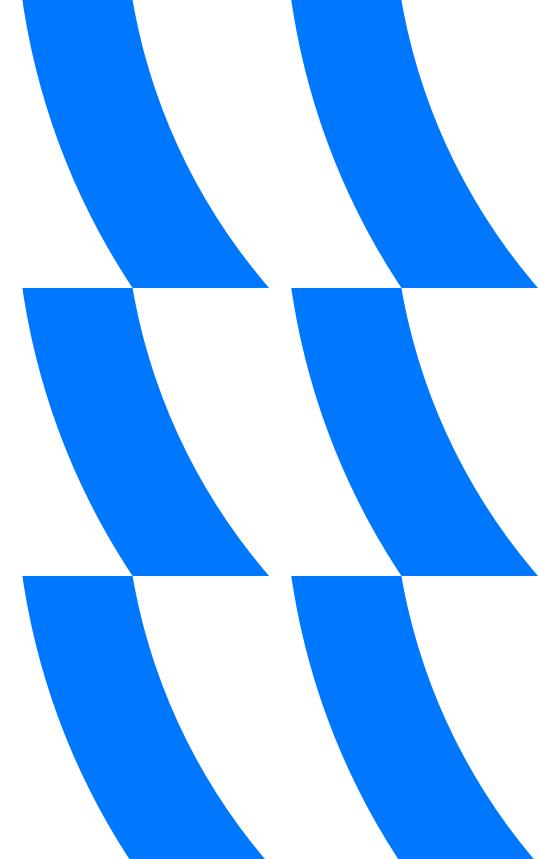
Отменить

Закрыть Esc

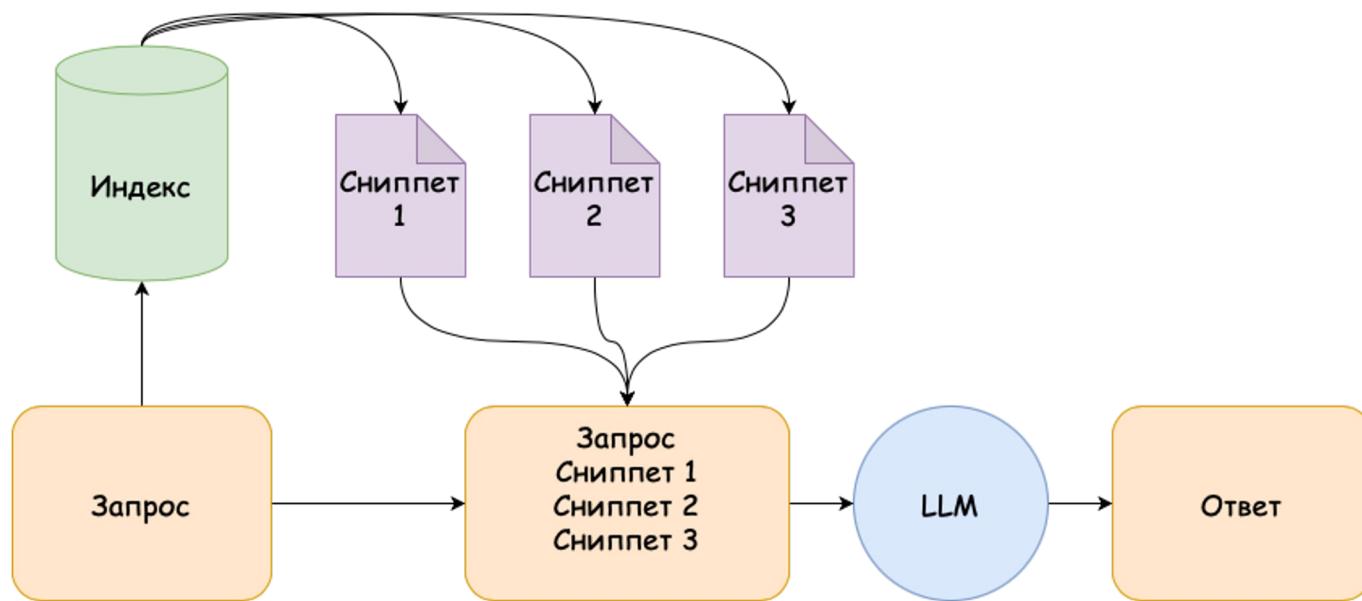
Дина Эрсilia Болуарте Сегарра — перуанская юристка и политик. С 7 декабря 2022 года — действующий президент Перу.

# Непараметрическая память

- Наиболее известные работы: REALM, ORQA, **RAG**
- RAG – retrieval-augmented generation
- Общая идея:
  - Получили запрос пользователя
  - Находим релевантные запросу сниппеты в заготовленном заранее индексе
  - Конкатенируем запрос и сниппеты
  - Подаем в генеративную модель
  - Ответ модели улучшается за счет внешних знаний



# Непараметрическая память



# RAG

- Как подготовить индекс?
- Как закодировать запрос и тексты из индекса?
- Как найти релевантные запросу тексты?
- Сколько релевантных запросов выбирать?
- Какие части этой архитектуры обучаемые?
- Как моделировать вероятности следующего токена и декодировать его?



# Подготовка индекса

В зависимости от задачи находятся потенциально полезные документы

- для QA: Википедия
- для саппорта: гайды по использованию системы
- для генеративного поиска: интернет

Документы – большие и шумные, часто содержат служебную разметку

- Удаляем весь мусор, оставляем только полезный текстовый контент
- Документы нарезаем на небольшие снippets: например, по 100 слов
- Желательно, чтобы снippetsы накладывались друг на друга – чтобы не потерять ответ, который был на стыке двух снippetsов

## А вот и BERT

- Закодируем все документы  $z$  в индексе с помощью BERT и зафиксируем
- Будем кодировать запросы  $x$  с помощью другого BERT – обучаемого
- MIPS: строим вероятностное распределение для всех документов в индексе (наиболее вероятный  $\Leftrightarrow$  наиболее релевантный)
- Берем топ- $k$  самых релевантных (вероятных) документов, на основе которых будем строить генерацию
- $k = 5, 10$

---

$$p_\eta(z|x) \propto \exp(\mathbf{d}(z)^\top \mathbf{q}(x)) \quad \mathbf{d}(z) = \text{BERT}_d(z), \quad \mathbf{q}(x) = \text{BERT}_q(x)$$

# Генератор

- С помощью генеративной модели (например, GPT) моделируем вероятностное распределение для следующего токена  $y_i$ :
  - По предыдущим токенам  $y_1, \dots, y_{i-1}$
  - По каждому релевантному текстовую сниппету  $z_j$
- Вероятности каждого из возможных следующих токенов  $y_i$  взвешиваем с помощью вероятностей (релевантностей) сниппетов

# Генератор

- RAG-sequence – каждый снippet используется для генерации всей последовательности
- RAG-token – выбираем все топ-k снippetов для генерации следующего токена

$$p_{\text{RAG-Sequence}}(y|x) \approx \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x, z) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) \prod_i^N p_\theta(y_i|x, z, y_{1:i-1})$$

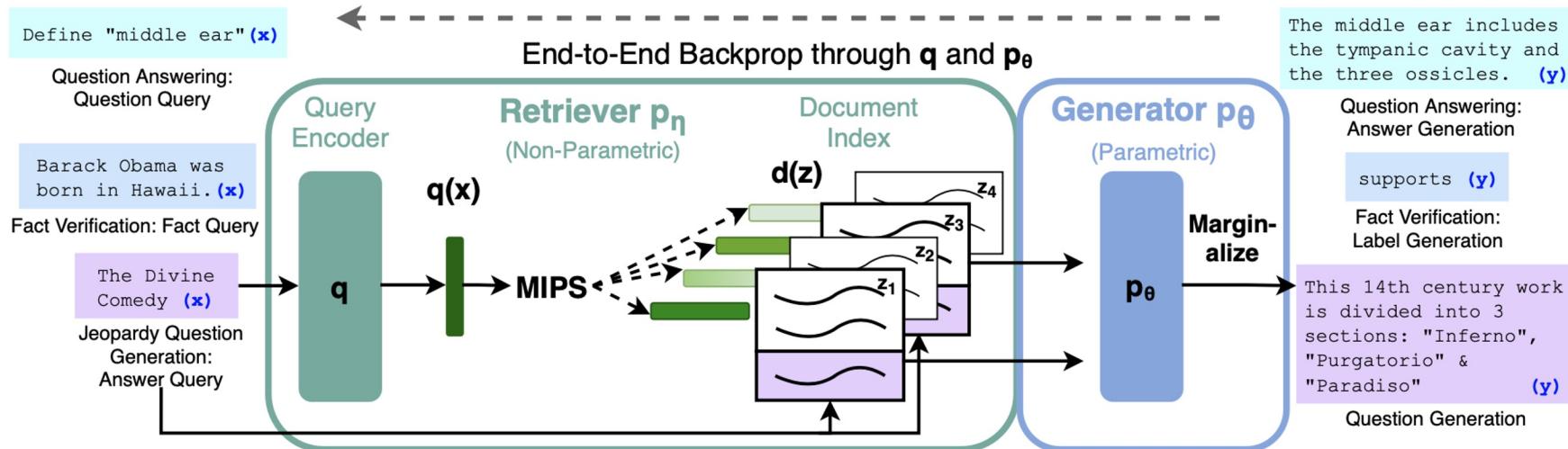
$$p_{\text{RAG-Token}}(y|x) \approx \prod_i^N \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z|x) p_\theta(y_i|x, z, y_{1:i-1})$$

# Что обучаем?

- Вход: текст (запрос + документы), таргет: текст (целевой ответ)
- Обучаем:
  - Веса генератора (например, GPT) – типичная задача LM; NLL-loss; Adam
  - BERT\_q – энкодер для запроса
  - \*BERT\_d – энкодер для документа – можем зафиксировать
- Почему фиксируем BERT\_d:
  - индекс в реальных задачах нужно время от времени менять / пополнять
  - его обучение на качество существенно не влияет

$$\sum_j -\log p(y_j | x_j)$$

# RAG



# Как декодировать

- В стандартной LM модель моделирует вероятностное распределение следующего токена, из которого мы семплируем токены и собираем всю последовательность токен за токеном
- В RAG наши вероятности взвешиваются с помощью вероятностей документа из индекса
- RAG-token: схема аналогичная, меняется только формула для вероятностей
- RAG-sequence: стандартным образом распределение не построить, поэтому считается чуть хитрее

$$p'_\theta(y_i|x, y_{1:i-1}) = \sum_{z \in \text{top-}k(p(\cdot|x))} p_\eta(z_i|x) p_\theta(y_i|x, z_i, y_{1:i-1})$$

# RAG – детали и выводы

- В оригинальной статье в качестве генератора использовалась модель BART
- Энкодеры – BERT Base
- Индекс – википедия с чанками по 100 слов (21M сниппетов)
- $k = 5, 10$  – больше не нужно
- За счет RAG ответы более:
  - точные и фактовые
  - разнообразные
- По метрикам лучше, чем обычный генератор
- Можно делать систему умнее, обновляя индекс и не меняя генератор

# Помните про domain-specific

- При использовании предобученных моделей важно учитывать, на каком домене они обучались
  - BERT, обученный на новостях, на электронных письмах может давать не очень полезные представления
  - GPT, обученная на английском языке, даже после FT может генерировать некачественный текст; начав отвечать по-русски, внезапно переходить на английский
- Выбирайте предобученную модель, наиболее близкую к вашему домену, либо пытайтесь эту модель к вашему домену приблизить

# Итого

- Оригинальный BERT
  - как устроен
  - на что и на чем обучается
  - насколько большой
  - для чего нужен
- Идеи, как сделать BERT лучше
- Как и где использовать BERT
- RAG – как BERT помогает генерировать тексты лучше

# Спасибо за внимание!

Дмитрий Калашников, разработчик-исследователь

