

# Нейронные сети в задачах NLP

04/05.10.2024

# Что такое NLP?

**NLP** – широкий круг **задач** по обработке текстов на **естественном языке** (т. е. на языке, на котором говорят и пишут люди).

Существует набор классических задач NLP, решение которых несет практическую пользу.

# Классические задачи NLP

Первая и самая исторически важная задача – это **машинный перевод**. Ей занимаются очень давно, и есть огромный прогресс.

# Классические задачи NLP

Вторая задача — **классификация текстов**. Дан набор текстов, и задача — классифицировать эти тексты по категориям. Каким? Это вопрос к корпусу.

- Классификация писем на спам.
- Многоклассовая классификация новостей по категориям (рубрикация) — внешняя политика, спорт, шапито и т. п.
- Анализ тональности текста (классификация отзывов на положительные, отрицательные и нейтральные).

# Классические задачи NLP

Третья задача – **извлечение именованных сущностей** (NER).

Мы выделяем в тексте **участки**, которые соответствуют заранее выбранному **набору сущностей**, например, надо найти в тексте все локации, персоны и организации.

В тексте «*Остап Бендер — директор конторы “Рога и Копыта”*» вы должны понять, что Остап Бендер – это **персона**, а “Рога и Копыта” – это **организация**.

# Классические задачи NLP

Четвертая задача — **извлечения фактов и отношений**.

Например, есть отношение работы (Occupation). Из текста «Остап Бендер — директор конторы “Рога и Копыта”» ясно, что наш герой связан профессиональными отношениями с “Рогами и Копытами”.

То же самое можно сказать множеством других способов: «Контору “Рога и Копыта” возглавляет Остап Бендер», или «Остап Бендер прошел путь от простого сына лейтенанта Шмидта до главы конторы “Рога и Копыта”». Эти предложения отличаются не только предикатом, но и структурой.

# Классические задачи NLP

Примерами других часто выделяемых отношений являются отношения купли/продажи (Purchase and Sale), владения (Ownership), факт рождения с атрибутами — датой, местом и т. д. (Birth) и некоторые другие.

Задача кажется не имеющей очевидного практического применения, но, тем не менее, она используется при структуризации неструктурированной информации. Кроме того, это важно в **вопросно-ответных** и **диалоговых системах**, в поисковиках — всегда, когда вам нужно анализировать вопрос и понимать, к какому типу он относится, а также, какие ограничения есть на ответ.

# Классические задачи NLP

Формулировка задачи **суммаризации** — на вход система принимает **текст большого размера**, а выходом служит **текст меньшего размера**, каким-то образом отражающий содержание большого.

Например, от машины требуется сгенерировать **пересказ** текста, его название или **аннотацию**.



# Какие сложности?

Явления **полисемии** (многозначные слова имеют общий исходный смысл) и **омонимии** (разные по смыслу слова произносятся и пишутся одинаково) характерны для любого естественного языка.

Полисемия: остановка (процесс или здание), стол (организация или объект), дятел (птица или человек).

Омонимия: ключ, лук, замок, печь.

# Какие сложности?

Другим классическим примером сложности языка является местоименная **анафора**. Например, пусть нам дан текст «Дворник два часа мел снег, он был недоволен».

Местоимение «он» может относиться как к **дворнику**, так и к **снегу**. По контексту мы легко понимаем, что он – это дворник, а не снег. Но добиться, чтобы компьютер это тоже легко понимал, непросто.

# Какие сложности?

Еще одна дополнительная сложность – это **эллипсис**.

Например, «Петя съел зеленое яблоко, а Маша – красное». Мы понимаем, что Маша съела красное яблоко. Тем не менее, добиться, чтобы машина тоже поняла это, непросто.

# Токены

Когда речь идет о языке, основная единица, с которой мы работаем, это слово (подслово), или более формально **«токен»**.

Мы используем этот термин, потому что не очень понятно, что такое 2128506 — это слово или нет? Токен обычно отделен от других токенов пробелами или знаками препинания. И как можно понять из сложностей, которые мы описали выше, очень важен **контекст** каждого **токена**.

# NLP пайплайн

1. Сегментация (деление текста на предложения)
2. Токенизация (деление предложений на токены, то есть отдельные (под)слова).

# NLP пайплайн

3. Вычисление признака (эмбединга) каждого токена. Как правило, это происходит в два этапа:

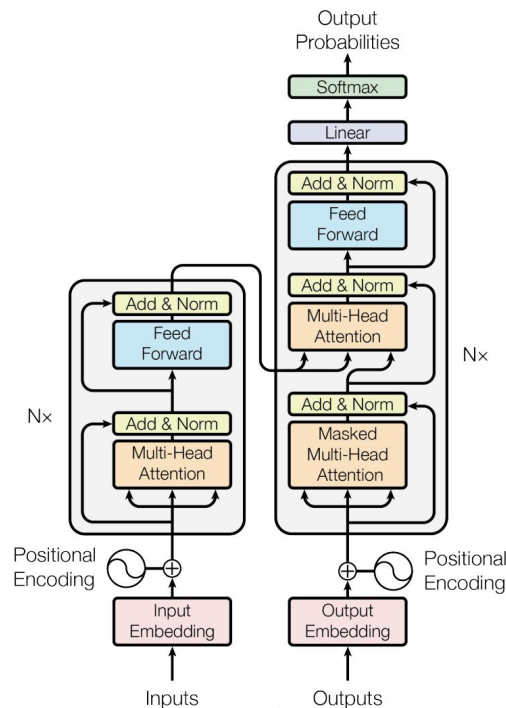
- Первый – вычислить **контекстно-независимые** признаки токена. Это набор признаков, которые никак не зависят от окружающих наш токен других токенов.
- Второй – вычислить **контекстно-зависимые** признаки токена.

# Архитектуры NN для NLP

- RNN
- LSTM
- **Transformer\***

BERT

Encoder



GPT

Decoder

# Модель BERT

1. **Классификация текста**
2. **Ответы на вопросы**
3. **Извлечение информации:** BERT может быть использован для извлечения конкретной информации из текста, например, для нахождения имен, дат, мест и других сущностей.
4. **Семантическое сходство:** Модель позволяет оценивать сходство между двумя текстами, что может быть полезно для задач, связанных с поиском дубликатов или для систем рекомендаций.



# BERT. Токенизация

**Токенизация** - процесс **разбиения** фрагмента текста на более **мелкие единицы**, называемые токенами, и присвоение каждому токену числового значения.

# BERT. Токенизация

Словарь (Vocabulary) токенизатора – это **список всех токенов**, которые он способен обработать. Пока что можно считать, что токен – это аналог слова.

# BERT. Токенизация

В реальном мире текст редко бывает таким простым, как "hello world". Часто он содержит знаки препинания (например, !, ?, ; и т.д.), символы интервала, такие как табуляция (\t) и новая строка (\n), или не ASCII символы, такие как эмодзи 🤔

Как токенизировать?

# BERT. Токенизация

Чтобы справиться с этими случаями, **BERT Tokenizer** выполняет несколько операций предварительной обработки:

1. Все **пробельные символы**, такие как табуляция и новая строка, **преобразуются** в одинарные пробелы.
2. Затем перед и после каждого **знака** препинания **добавляются пробелы**. Это позволяет рассматривать знаки препинания как **отдельные входные токены**, помимо слов, с которыми они связаны во входной строке.

# BERT. Токенизация

Например, строка "hello world!" разбивается на следующие 6 токенов:

[CLS] hello , world ! [SEP]

Которые соответствуют следующим индексам в словаре токенизатора:

101 7592 1010 2088 999 102

# Токенизация подслова

Один из способов, с помощью которого токенизатор BERT может **эффективно обрабатывать** широкий спектр входных строк с ограниченным словарным запасом, заключается в использовании техники **токенизации подслова** под названием *WordPiece*.

Эта техника позволяет представлять некоторые слова, не входящие в словарный запас, в виде нескольких "подслов", а не в виде токена **[UNK]**.

Например, рассмотрим слова "clock" and "clockwork". В словаре токенизатора, "clock" находится по индексу 5 119, а слова "clockwork" вообще не присутствует в словаре.

# Токенизация подслова

При токенизации строки "clockwork" вместо того, чтобы преобразовать ее в токен [UNK], токенизатор преобразует это **одно** слово в **два** отдельных токена: **clock** и **##work**.

Символы **##** используются токенизатором в качестве суффиксального индикатора, чтобы отличить "work" как **суффикс** (как в "clockwork", "classwork" и т. д.) от "work" как **самостоятельного слова**.

# Токенизация подслова

Иногда существует несколько способов разбить слово на подслова. Например, слово "metalworking" может быть разбито одним из следующих способов:



metal ##work ##ing



metal ##working



# Токенизация подслова

Аналогично, слово "artwork" может быть представлено токенами **art** и **##work**. Однако, поскольку словарь содержит токен **artwork**, вместо нее используется этот единственный токен.

Токенизатор BERT всегда будет пытаться разбить слово на наименьшее количество подслов, то есть строка "metalworking" будет разбита на токены **metal** и **##working**.

# BERT. Embeddings layer

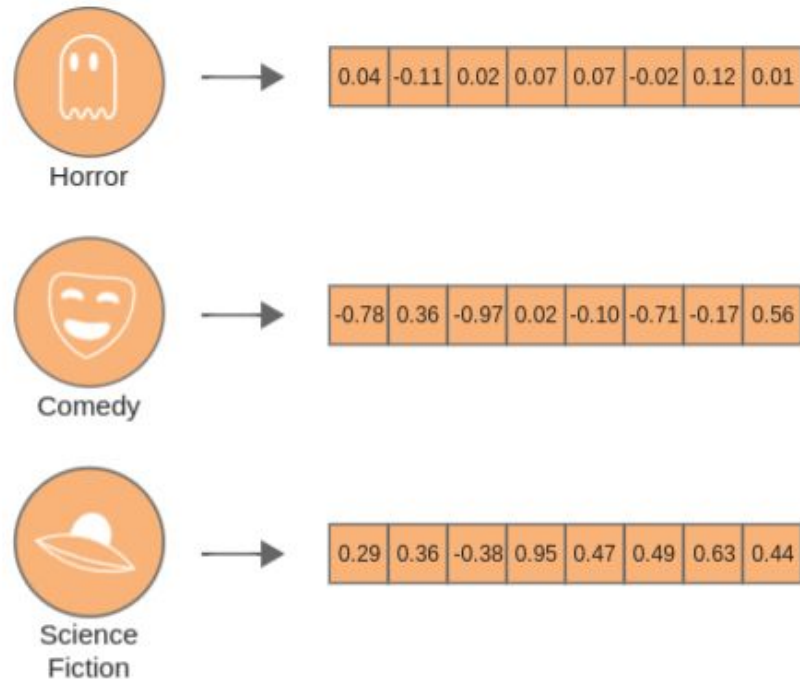
Признаки, или входные данные, любой модели машинного обучения, как правило, можно считать **непрерывными** или **категориальными**. Непрерывные признаки представляют собой числовые значения.

**Категориальные признаки** представляют собой экземпляры определенной категории. Категориальные признаки имеют **конечное** множество возможных значений.

**Эмбединг** – это обученное **числовое представление** категориального признака. На практике это означает список значений с плавающей точкой, которые обучаются в процессе обучения модели. Количество этих значений, также называемое размером эмбединга, может варьироваться от модели к модели.

# Пример эмбединга

Например, если для представления жанров фильмов используется **эмбединг** размера 8, то каждый жанр будет отображаться на определенный список из 8 значений с плавающей точкой, и эти значения будут настраиваться в процессе обучения, как и любой другой вес в нейронной сети.



# Типы эмбеддингов в BERT


Для представления текстовых входных данных BERT использует 3 различных типа эмбеддингов:

1. Token Embeddings
2. Position Embeddings
3. Token Type Embeddings.

# 1. Token Embeddings

Прежде чем строка текста будет передана в модель BERT, **BERT Tokenizer** используется для преобразования входа из строки в список целочисленных идентификаторов токенов, где каждый идентификатор непосредственно соответствует слову или части слова в исходной строке.

Например, строка "Hello, world!" преобразуется токенизатором в следующие идентификаторы токенов:



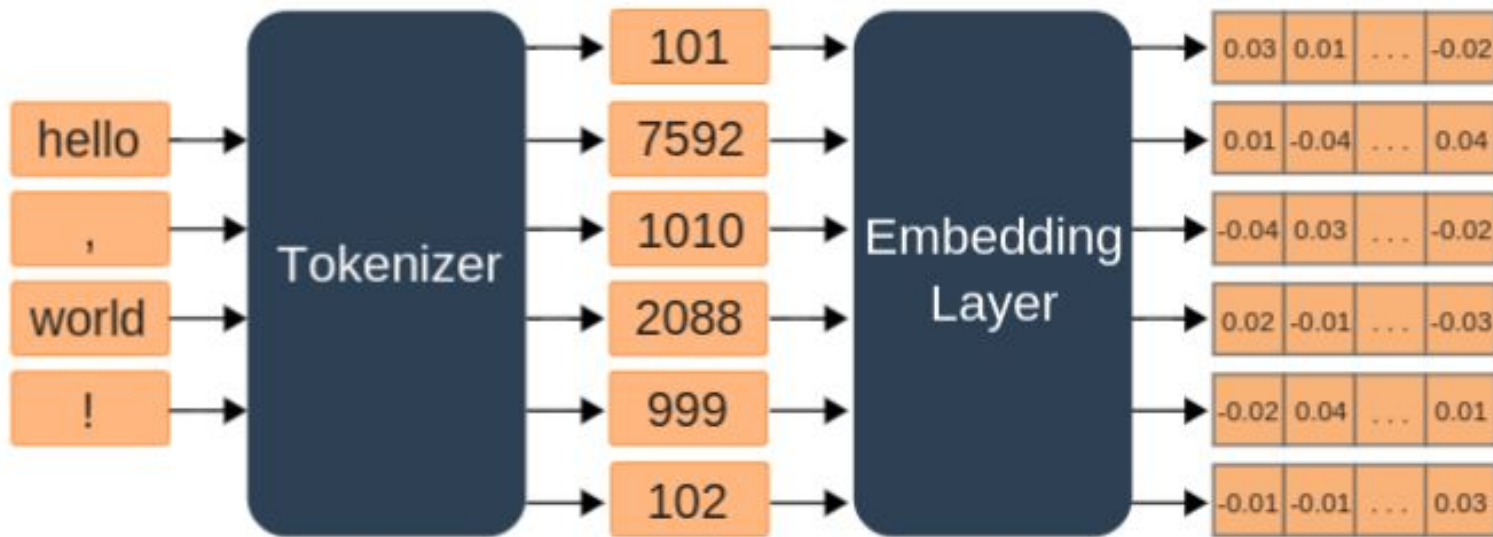
101	7592	1010	2088	999	102
-----	------	------	------	-----	-----

# Размер словаря BERT

Для каждого уникального идентификатора токена (для каждого из 30 522 слов и подслов в словаре BERT Tokenizer) модель BERT содержит эмбединг, который обучен представлять этот конкретный токен.

**Embedding layer** в модели отвечает за сопоставление токенов с соответствующими эмбедингами.

# Пример токенизации



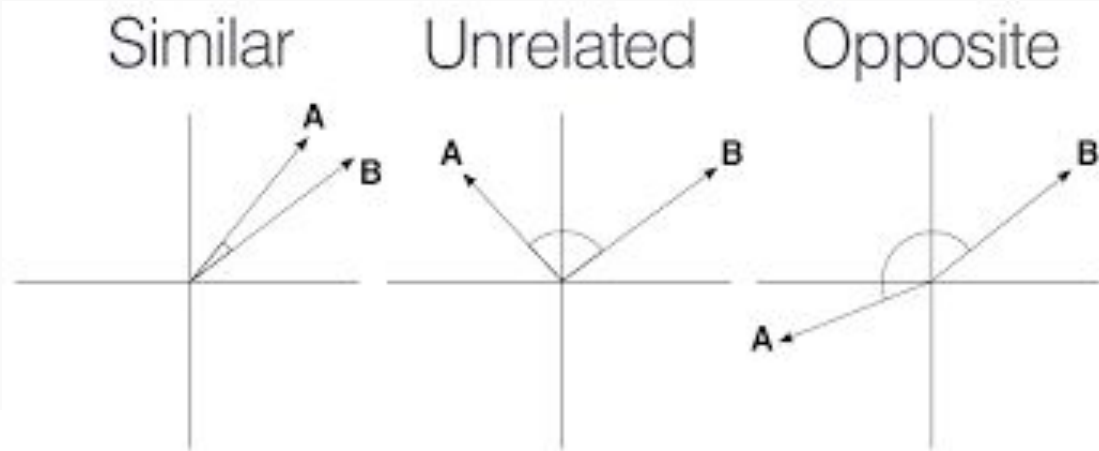
# Сравнение эмбеддингов

Хотя при взгляде на них это не очевидно, **эмбеддинги** для каждого идентификатора токена – это **не** просто **случайные числа**. Эти значения были получены при обучении модели BERT, что означает, что **каждый эмбеддинг** кодирует **понимание** моделью данного **конкретного токена**.

Можно увидеть, какие слова модель BERT считает наиболее похожими, сравнив их соответствующие эмбеддинги токенов. Поскольку каждый эмбеддинг представляет собой вектор или одномерный тензор, для измерения степени сходства двух эмбеддингов можно использовать **косинусное сходство**.



# Косинусное сходство



$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}},$$

## 2. Position Embeddings

В дополнение к описанным до сих пор Token Embeddings, BERT также полагается на **Position Embeddings**. В то время как Token Embeddings используются для **представления** каждого возможного **слова** или подслова, которые могут быть предоставлены модели, **Position Embeddings** представляют **позицию** каждого токена во входной последовательности.

## 2. Position Embeddings



# Ограничение на вход BERT

В то время как существует 30 522 различных Token Embeddings, существует только **512** различных Position Embeddings.

Это связано с тем, что **наибольшая входная последовательность**, принимаемая моделью BERT, имеет длину 512 токенов.

# 3. Token Type Embeddings

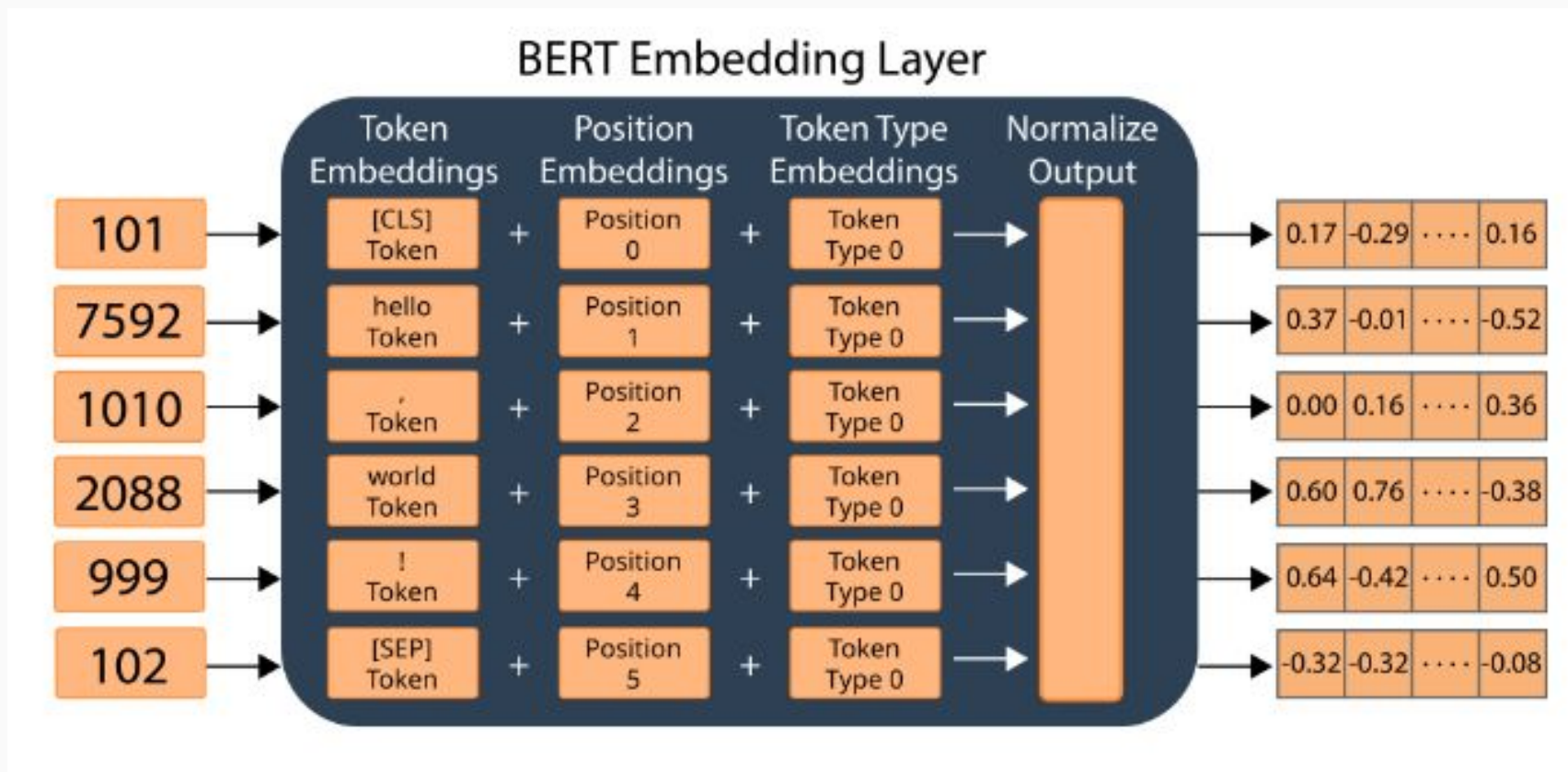
Одной из задач, для решения которой изначально обучался BERT, было **предсказание следующего предложения**. То есть, если даны два предложения A и B, BERT обучался определять, логически ли B следует за A.

Существует **только два** различных Token Type Embeddings: один используется для представления токенов в предложении A, а другой – для представления токенов в предложении B.

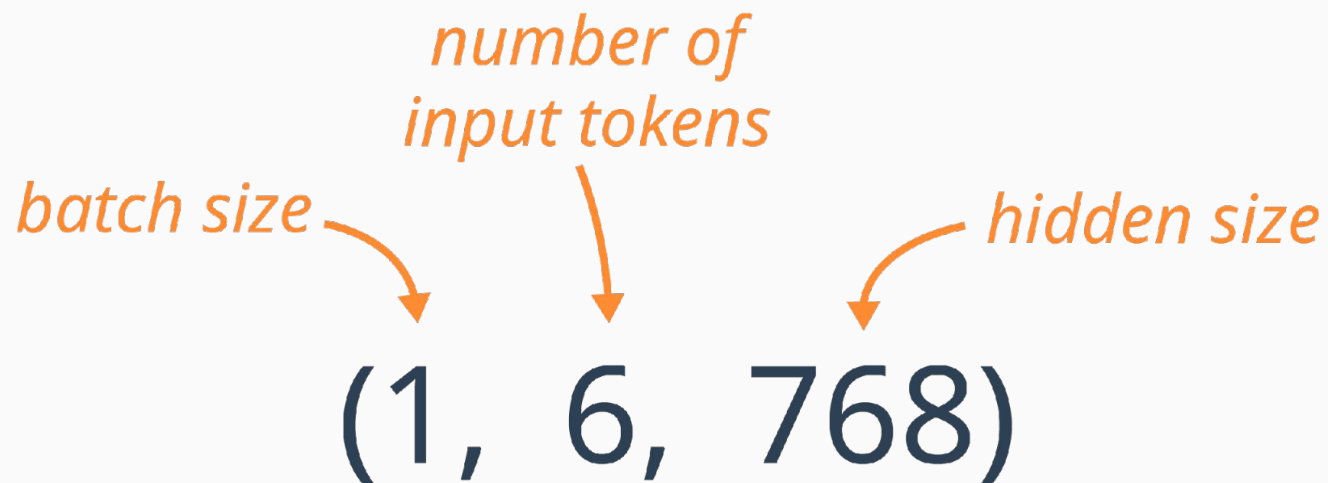
# BERT. Embedding Layer

Учитывая список идентификаторов токенов, BERT Embedding Layer отвечает за вычисление **итогового эмбединга** для каждого входного токена путем **суммирования** эмбедингов *Token Embeddings*, *Position Embeddings* и *Token Type Embeddings*, с последующей **нормализацией**.

# BERT. Embedding Layer



# Размерность выходного слоя





# BERT. Encoder Layer

**Encoder** – это основной строительный блок архитектуры модели BERT. На вход кодировщик принимает текстовые эмбединги, созданные **Embedding Layer**, а на выходе возвращает модифицированные эмбединги той же длины.

Поскольку вход и выход кодировщика имеют одинаковую длину, можно объединить в цепочку несколько кодировщиков, чтобы выход одного из них стал входом для следующего.

Оригинальная архитектура модели BERT включает в себя **12** кодировщиков, соединенных таким образом.

# Добавление контекста

Как уже говорилось, **BERT Encoder** принимает эмбеддинги на вход и производит эмбеддинги на выход. Чем отличаются эмбеддинги, созданные кодировщиком, от эмбеддинги, созданных Embedding Layer?

Для каждого входного токена **Embedding Layer** создает эмбеддинг, который представляет этот конкретный токен, а также его позицию во входной последовательности текста.

Однако **отдельного** слова и его **позиции** недостаточно, чтобы понять значение слова в строке текста.

# Добавление контекста

Например, представьте, что вам дана последовательность из 9 слов, где 5-е слово – "fire". Можете ли вы с уверенностью сказать, какое значение имеет это слово в этом фрагменте текста?

\_\_\_\_\_ *fire* \_\_\_\_\_  
\_\_\_\_\_

# Добавление контекста

*He is going to fire one of his employees*

*There was a huge fire raging through the forest*

*I learned how to fire a gun last year*

# Добавление контекста

**Очевидно, нет.** В зависимости от контекста (то есть слов перед и после) слово "fire" может иметь одно из множества различных значений

# Добавление контекста

В каждом из трех приведенных выше предложений Embedding Layer BERT создает **один и тот же** эмбеddинг для слова "fire".

**Encoder Layer**, с другой стороны, создает контекстуальный эмбеddинг для каждого токена, кодируя информацию не только о **самом** токене, но и о **других** токенах в тексте.

# Сравнение контекстуальных эмбеддингов

Сходство контекстуальных эмбеддингов, созданных кодировщиком BERT, может быть вычислено с помощью **косинусного сходства**. Это позволяет определить сходство значения двух слов в соответствующих контекстах.

Например, косинусное сходство между словом "fire" в первом и втором предложениях составляет 0.79.

# Вторая часть лекции

- Transfer learning / fine-tuning
- Transformer arch\*
- Разбор Jupyter-ноутбуков с некоторых актуальных вопросов