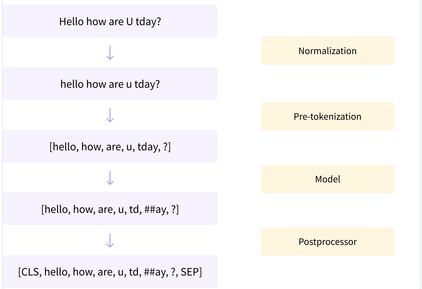
Нормализация и предварительная токенизация

Прежде чем мы более подробно рассмотрим три наиболее распространенных алгоритма токенизации подслов, используемых в моделях Transformer (Byte-Pair Encoding [BPE], WordPiece и Unigram), мы сначала рассмотрим предварительную обработку, которую каждый токенизатор применяет к тексту. Вот высокоуровневый обзор этапов конвейера токенизации:



Перед тем как разбить текст на подтокены (в соответствии со выбранной моделью), токенизатор выполняет два шага: *нормализацию* и *претокенизацию*.

Нормализация

Шаг нормализации включает в себя некоторую общую очистку, например, удаление ненужных пробельных символов, понижение регистра и/или удаление ударений. Если вы знакомы с Unicode normalization (например, NFC или NFKC), это также может быть применено токенизатором.

У 🤗 Transformers `tokenizer` есть атрибут `backend_tokenizer`, который предоставляет доступ к базовому токенизатору из библиотеки 🤗 Tokenizers:

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
print(type(tokenizer.backend_tokenizer))
```

```
<class 'tokenizers.Tokenizer'>
```

Атрибут `normalizer` объекта `tokenizer` имеет метод `normalize_str()`, который мы можем использовать, чтобы увидеть, как выполняется нормализация:

```
print(tokenizer.backend_tokenizer.normalizer.normalize_str("Héllò hôw are ü?"))
```

```
'hello how are u?'
```

В этом примере, поскольку мы выбрали контрольную точку `bert-base-uncased`, нормализация применила нижний регистр и удалила ударения.

> ✏️ **Попробуйте!** Загрузите токенизатор из контрольной точки `bert-base-cased` и передайте ему тот же пример. Какие основные различия вы можете увидеть между версией токенизатора cased и uncased?

Предварительная токенизация

Как мы увидим в следующих разделах, токенизатор не может быть обучен только на сыром тексте. Сначала необходимо разбить текст на небольшие части, например, на слова. Именно в этом и заключается этап предварительной токенизации. Как мы видели в Главе 2, токенизатор на основе слов (word-based tokenizer) может просто разбить необработанный текст на слова по пробелам и знакам пунктуации. Эти слова станут границами подтокенов, которые токенизатор сможет выучить в процессе обучения.

Чтобы увидеть, как быстрый токенизатор выполняет предварительную токенизацию, мы можем воспользоваться методом `pre_tokenize_str()` атрибута `pre_tokenizer` объекта `tokenizer`:

```
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are  you?")
```

```
[('Hello', (0, 5)), (',', (5, 6)), ('how', (7, 10)), ('are', (11, 14)), ('you', (16, 19)), ('?', (19, 20
```

Обратите внимание, что токенизатор уже следит за смещениями, и именно поэтому он может дать нам сопоставление смещений, которое мы использовали в предыдущем разделе. Здесь токенизатор игнорирует два пробела и заменяет их одним, но смещение перескакивает между `are` и `you`, чтобы учесть это.

Поскольку мы используем токенизатор BERT, предварительная токенизация включает часть пробельных символов и пунктуацию. Другие токенизаторы могут иметь другие правила для этого шага. Например, если мы используем токенизатор GPT-2:

```
tokenizer = AutoTokenizer.from_pretrained("gpt2")
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are  you?")
```

он также выполнит разбиение по пробельным символам и пунктуации, но сохранит пробелы и заменит их символом `Ġ`, что позволит ему восстановить исходные пробелы, если мы декодируем токены:

```
[('Hello', (0, 5)), (',', (5, 6)), ('Ġhow', (6, 10)), ('Ġare', (10, 14)), ('Ġ', (14, 15)), ('Ġyou', (15,
 ('?', (19, 20))]
```

Также обратите внимание, что в отличие от токенизатора BERT, этот токенизатор не игнорирует двойной пробел.

В качестве последнего примера рассмотрим токенизатор T5, основанный на алгоритме SentencePiece:

```
tokenizer = AutoTokenizer.from_pretrained("t5-small")
tokenizer.backend_tokenizer.pre_tokenizer.pre_tokenize_str("Hello, how are  you?")
```

```
[('▁Hello,', (0, 6)), ('▁how', (7, 10)), ('▁are', (11, 14)), ('▁you?', (16, 20))]
```

Как и токенизатор GPT-2, этот сохраняет пробелы и заменяет их специальным токеном (`_`), но токенизатор T5 делает разбиение только по пробелам, а не по знакам препинания. Также обратите внимание, что он по умолчанию добавляет пробел в начале предложения (перед `Hello`) и игнорирует двойной пробел между `are` и `you`.

Теперь, когда мы немного познакомились с тем, как обрабатывают текст различные токенизаторы, можно приступить к изучению самих алгоритмов, лежащих в их основе. Мы начнем с краткого обзора широко применяемого SentencePiece; затем, в следующих трех разделах, мы рассмотрим, как работают три основных алгоритма, используемых для токенизации по подсловам.

SentencePiece

SentencePiece - это алгоритм токенизации для предварительной обработки текста, который можно использовать с любой из моделей, которые мы рассмотрим в следующих трех разделах. Он рассматривает текст как последовательность символов Unicode и заменяет пробелы специальным символом `▁`. При использовании в сочетании с алгоритмом Unigram (см. раздел 7) он даже не требует шага предварительной токенизации, что очень полезно для языков, где символ пробела не используется (например, китайского или японского).

Другой главной особенностью SentencePiece является *обратимая токенизация*: поскольку в нем нет специальной обработки пробелов, декодирование токенов осуществляется просто путем их конкатенации и замены `▁` на пробелы - в результате получается нормализованный текст. Как мы видели ранее, токенизатор BERT удаляет повторяющиеся пробелы, поэтому его токенизация не является обратимой.

Обзор алгоритма

В следующих разделах мы рассмотрим три основных алгоритма токенизации по подсловам: BPE (используется в GPT-2 и других моделях), WordPiece (используется, например, в BERT) и Unigram (используется в T5 и других моделях). Прежде чем мы приступим, вот краткий обзор того, как работает каждый из них. Не стесняйтесь возвращаться к этой таблице после прочтения каждого из следующих разделов, если вам еще не все понятно.

Model	BPE	WordPiece	Unigram
Обучение	Начинается с маленького словаря и изучает правила слияния токенов	Начинается с маленького словаря и изучает правила слияния токенов	Начинается с большого словаря и изучает правила удаления токенов
Шаг обучения	Объединяет токены, соответствующие наиболее часто встречающейся паре	Объединяет токены, соответствующие паре с наилучшей оценкой, основанной на частоте пары, отдавая предпочтение парам, где каждый отдельный токен встречается реже	Удаляет все токены в словаре, что минимизирует потери, вычисленные для всего корпуса.
Обучение	Слияние правил и словаря	Только словарь	Словарь с оценкой каждого токена
Кодирование	Разбивает слово на символы и применяет слияния, полученные во время обучения	Находит самое длинное подслово, начиная с начала, которое есть в словаре, затем делает то же самое для остальной части слова	Находит наиболее вероятное разбиение на токены, используя оценки, полученные во время обучения

А теперь давайте погрузимся в BPE!