The 🤗 Tokenizers library

How to train a brand new tokenizer on a corpus of texts, so it can then be used to pretrain a language model.

What is happening in this chapter?

- How to train a new tokenizer similar to the one used by a given checkpoint on a new corpus of texts
- The special features of fast tokenizers
- The differences between the three main subword tokenization algorithms used in NLP today
- How to build a tokenizer from scratch with the 🤗 Tokenizers library and train it on some data

Training a new tokenizer from an old one:

Training a tokenizer is not the same as training a model! Model training uses stochastic gradient descent to make the loss a little bit smaller for each batch.

It's randomized by nature (meaning you have to set some seeds to get the same results when doing the same training twice).

Training a tokenizer is a statistical process that tries to identify which subwords are the best to pick for a given corpus, and the exact rules used to pick them depend on the tokenization algorithm.

It's deterministic, meaning you always get the same results when training with the same algorithm on the same corpus.

Skipping how to train new tokenizer for now.

Fast tokenizers' special powers

Note: In the following discussion, we will often make the distinction between “slow” and “fast” tokenizers. Slow tokenizers are those written in Python inside the 🤗 Transformers library, while the fast versions are the ones provided by 🤗 Tokenizers, which are written in Rust.

```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")
example = "My name is Sylvain and I work at Hugging Face in Brooklyn."
encoding = tokenizer(example)
print(type(encoding))

## <class 'transformers.tokenization_utils_base.BatchEncoding'>

tokenizer.is_fast
# True

encoding.is_fast
# True
```

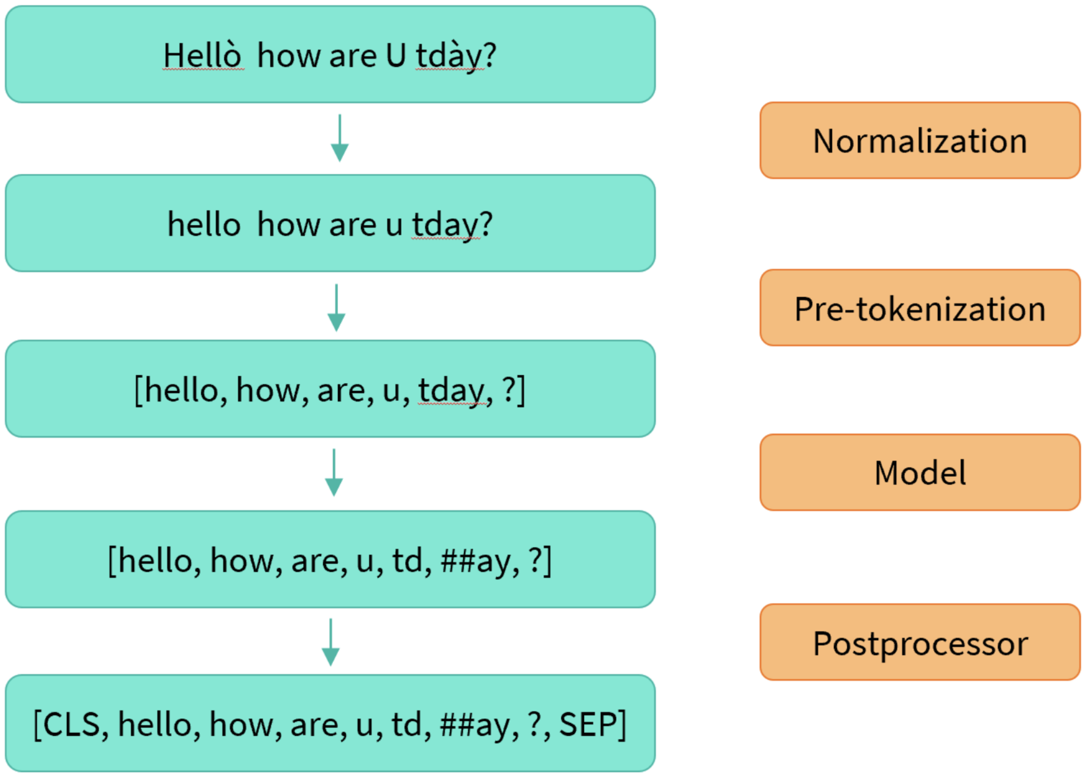
Fast tokenizers in the QA pipeline

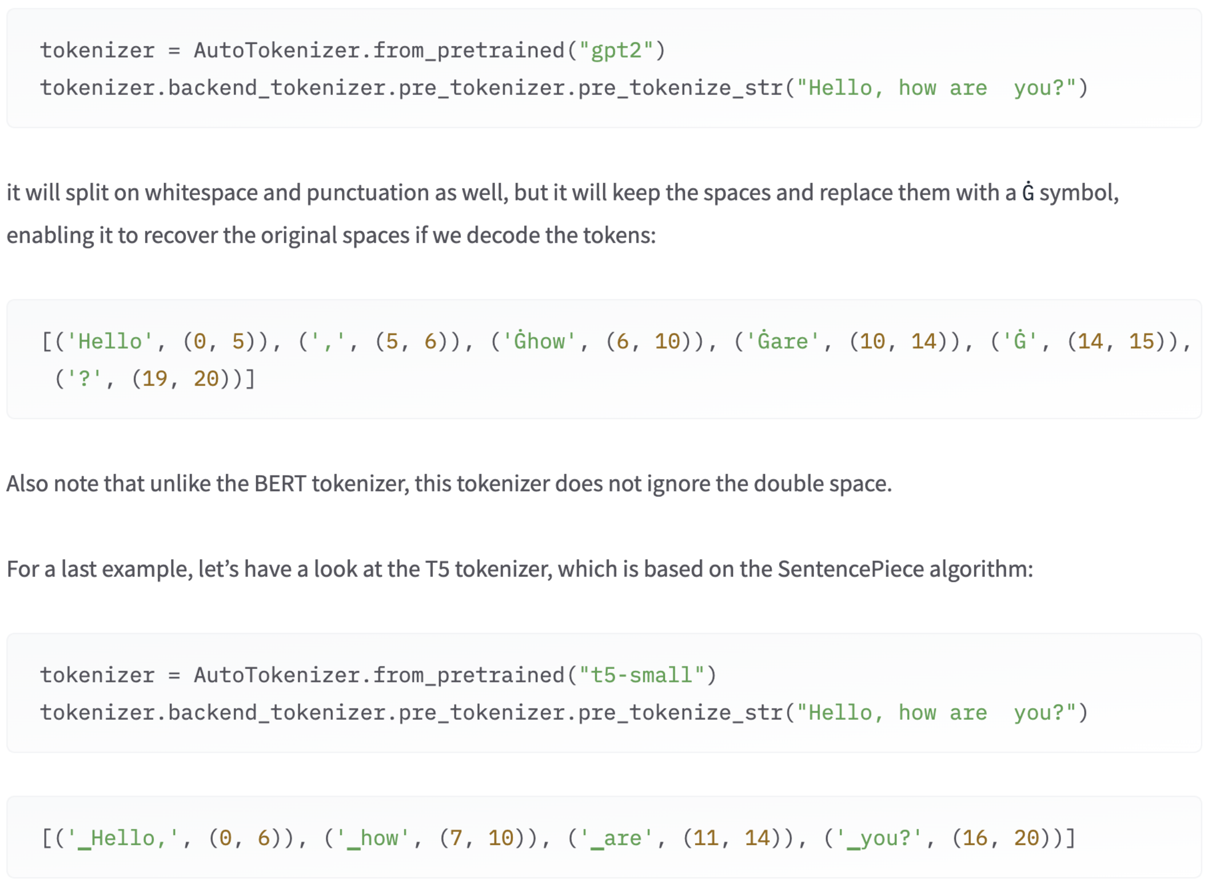
Skipped

Normalization and pre-tokenization

1. The normalization step involves some general cleanup, such as removing needless whitespace, lowercasing, and/or removing accents.

2. The 🤗 Transformers `tokenizer` has an attribute called `backend_tokenizer` that provides access to the underlying tokenizer from the 🤗 Tokenizers library:


Hellò how are U tdày?
hello how are u tday?
[hello, how, are, u, tday, ?]
[hello, how, are, u, td, ##ay, ?]
[CLS, hello, how, are, u, td, ##ay, ?, SEP]
Normalization
Pre-tokenization
Model
Postprocessor



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

it will split on whitespace and punctuation as well, but it will keep the spaces and replace them with a Ġ symbol, enabling it to recover the original spaces if we decode the tokens:

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

Also note that unlike the BERT tokenizer, this tokenizer does not ignore the double space.

For a last example, let's have a look at the T5 tokenizer, which is based on the SentencePiece algorithm:

```
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))]
```

SentencePiece:

SentencePiece is a tokenization algorithm for the preprocessing of text. It considers the text as a sequence of Unicode characters, and replaces spaces with a special character, `▁`. Used in conjunction with the Unigram algorithm it doesn't even require a pre-tokenization step, which is very useful for languages where the space character is not used (like Chinese or Japanese).

Byte-Pair Encoding tokenization:

Byte-Pair Encoding (BPE) was initially developed as an algorithm to compress texts, and then used by OpenAI for tokenization when pretraining the GPT model. It's used by a lot of Transformer models, including GPT, GPT-2, RoBERTa, BART, and DeBERTa.

WordPiece tokenization

WordPiece is the tokenization algorithm Google developed to pretrain BERT. It has since been reused in quite a few Transformer models based on BERT, such as DistilBERT, MobileBERT, Funnel Transformers, and MPNET. It’s very similar to BPE in terms of the training, but the actual tokenization is done differently.

Unigram tokenization

The Unigram algorithm is often used in SentencePiece, which is the tokenization algorithm used by models like AlBERT, T5, mBART, Big Bird, and XLNet.

Building a tokenizer, block by block

Summary:

Tokenizers are simply awesome, most of the tokenizer I have dealt with have been a dictinoary which mapped a vocabulary (string: number) and did some encoding on that. But the modern tokenizers take a lot of other things into account and work well. I have not gone in depth in any of these above mentioned sections as I don’t feel an urgent need to digest these.

But now I am aware of the various modern tokenization algorithm and how post-processing is not a simple look up into a table.