Language Technology
http://cs.lth.se/edan20/
Chapter 7, Subword tokenization

Pierre Nugues

Pierre.Nugues@cs.lth.se
http://cs.lth.se/pierre_nugues/

September 11, 2023

Motivation: Language Differences (Source: Xerox)

Breaking a text into morphemes is more economical then breaking it into words.

Language	**# stems**	**# inflected forms**		**Lex. size (kb)**
English	55,000	240,000		200–300
French	50,000	5,700,000		200–300
German	50,000	350,000	or	450
		infinite	(compounding)	
Japanese	130,000	200	suffixes	500
		20,000,000	word forms	500
Spanish	40,000	3,000,000		200–300

Identifying all the morphemes of all the languages and writing parse rules would be difficult

Instead, we can derive them automatically from a corpus.

Déjean proposed to cluster prefixes and suffixes according to distributional properties of the letters.

English	-e -s -ed -ing -al -ation -ly -ic -ent
French	-s -e -es -ent -er -ds -re -ation -ique
German	-en -e -te -ten -er -es -lich -el
Turkish	-m -in -lar -ler -dan -den -inl -ml
Swahili	-wa -ia -u -eni -o -isha -ana -we
	wa- m- ku- ali- ni- aka- ki- vi-

Subword Tokenization Techniques

What if the documents are in two languages: Korean and Japanese?
In the context of a multilingual web, it is preferable to have minimal assumptions
We will examine how to

1. Create a dictionary of substrings from any raw corpus using statistics only;
2. Use them to tokenize a text into subwords.

Subword tokenization variants include:

- Byte-Pair Encoding (BPE)
- WordPiece
- Language model segmentation

Overview

These techniques have two steps:

1. A training step, where they build a vocabulary. This is done once;
2. A segmentation step, where they apply the model (vocabulary) to break a string into tokens.

They are entirely data-driven although they often rediscover morphemes.

Training

The model and the vocabulary are trained or extracted from a corpus
In the training step:

- We use the set of characters and merge them to form words until they have reached a predefined vocabulary size.
- Merging is done by frequency (BPE) or with a language model criterion (WordPiece)

The vocabulary consists then in the most frequent character sequences.

Segmentation

Using the model, the tokenizer splits the text into subwords using either:

- The merge sequence (BPE)
- The longest match (WordPiece)
- A language model criterion (unigrams)

SentencePiece implements BPE and unigram segmentation

Byte-Pair Encoding (BPE)

Originally, BPE is a compression algorithm

Sennrich adapted the original BPE algorithm to build automatically a lexicon of subwords from a corpus.

These subwords consist of

- a single character,
- A sequence of characters,
- Possibly a whole word

The size of the lexicon is fixed in advance, for instance 20,000 tokens

BPE Algorithm

The main steps of the algorithm are:

1. Split the corpus into individual characters. These characters will be the initial subwords and will make up the start vocabulary:
2. Then:
   1. Extract the most frequent adjacent pair from the corpus;
   2. Merge the pair and add the corresponding subword to the vocabulary;
   3. Replace all the occurrences of the pair in the corpus with the new subword;
   4. Repeat this process until we have reached the desired vocabulary size.

To tokenize a text, the merge rules are applied in the same order

BPE can be character or byte-based

With bytes, by construction, there is no unknown word. The algorithm always falls back to a byte

Demonstration

SentencePiece with BPE:
https://github.com/google/sentencepiece/blob/master/python/sentencepiece_python_module_example.ipynb
The vocabulary: vocab.m

Pretokenization

pretokenization: splitting a text document into smaller tokens before feeding it to the model. A token is asingle character or a long word, or a subword

BPE normally does not cross the whitespaces.

It can apply or not a whitespace pretokenization:

- BPE in GPT-2 uses pretokenization. It can speed up the learning process with a word count.

  The simplest pretokenization uses the whitespaces as delimiters:

  ```
  pattern = r'\p{L}+|\p{N}+|[^\s\p{L}\p{N}]+'
  ```

  See also: https://github.com/karpathy/minGPT/blob/master/mingpt/bpe.py
- BPE in SentencePiece uses raw text, where it replaces spaces with ▁ (U+2581). It does not cross this symbol.

Results

SentencePiece tokenizes the sentence

This is a text

as:

['▁This', '▁is', '▁a', '▁t', 'est']

The ▁ (U+2581) prefix makes the output more legible.
When using a pretokenization (GPT-2), we segment the pretokenized words

To speed up the merges, we can use a cache.

As the results can be difficult to read, BPE in GPT-2 adds a Ġ prefix

WordPiece

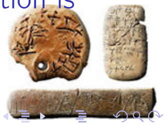
WordPiece's principles are similar to BPE.
The main differences are

1. The criterion to merge a pair is the quality of the resulting language model;
2. The tokenization uses a greedy longest match algorithm.

Google never released WordPiece's construction algorithm.
If they used a unigram model, the merge decision would be the pair that maximizes the difference:

$$\prod_{i=1}^{N} P(x_i)$$

before and after the merge
As there is no public implementation, the vocabulary construction is often replaced by BPE.



WordPiece Tokenization

Given a vocabulary, WordPiece uses a greedy longest match:

- Word: *there*
- Vocabulary: [th, the, he, re, t, h, r, e]

Competing segmentations: [th, e, r, e], [t, he, re], [th, e, re], [the, re], etc.

Solution:

1. Sort the subwords in the vocabulary by length
2. Create a disjunction
3. Apply `finditer()`
4. Check you have not lost characters (See demo)

WordPiece marks the subwords with ## prefix from the second one

there –> `[the, ##re]`

BERT implementation: `https://github.com/google-research/bert/blob/master/tokenization.py#L300C7-L300C25`

Unigram

The unigram tokenizer is only about tokenization
Given a vocabulary, the longest match or sequential merges may not optimize a language model.
The unigram tokenization selects the maximal product:

$$
\begin{aligned}
&P(th) \cdot P(e) \cdot P(r) \cdot P(e),\\
&P(t) \cdot P(he) \cdot P(re),\\
&P(th) \cdot P(e) \cdot P(re),\\
&P(the) \cdot P(re),
\end{aligned}
$$

A brute-force search has an exponential complexity
In the lab you will use Norvig's implementation of Viterbi's algorithm
`https://nbviewer.org/url/norvig.com/ipython/How%20to%20Do%20Things%20with%20Words.ipynb`, Sec. 5
Just adapt his code

Unigram

The vocabulary construction uses BPE generally

We estimate the probabilities with the expectation-maximization algorithm:

1. We first estimate the distribution with a BPE tokenization,
2. We segment with the unigram language model and the current distribution
3. We re-estimate the distribution from the segmentation
4. We repeat 2 and 3 until convergence

The unigram original paper also includes a step to discard less valuable subtokens. See the lab.

SentencePiece

SentencePiece is a subword tokenizer with two segmentation algorithms: BPE or a unigram language model.

It treats the space as any other character which makes it well suited for CJK languages

As a preprocessing step, it replaces the spaces with a ▁ (U+2581) character. It is used in many large language models

Code

BPE: https://github.com/rsennrich/subword-nmt/tree/master/subword_nmt

SentencePiece: https://github.com/google/sentencepiece/tree/master

Implementation in Rust: https://github.com/huggingface/tokenizers