

Solutions for Assignment 2

Exercise 1 - Implementing a tokenizer

Implement a basic whitespace tokenizer in Python from scratch without the use of any NLP libraries. This tokenizer should drop whitespaces and create tokens for the following cases:

- (a) End-of-sentence (EOS) symbols, brackets and separators
- (b) Abbreviations - Assume those are only one of the following: Ph.D., Dr., M.Sc.
- (c) Special characters as in prices separated (i.e. \$45.55)
- (d) Dates - Assume that they follow the format dd/mm/yy (i.e. 01/02/06)
- (e) URLs - Assume that they follow the format:
`http[s]://[...]`, (i.e. <https://www.stanford.edu>)
- (f) Hashtags separated (i.e. #nlproc)
- (g) Email addresses - Assume that they follow the format:
`name@domain.xyz` (i.e. `someOne@brown.edu`)

Apply your code on the test example below, which should yield the specified tokens:

Exercise 2 - Implementing a BPE tokenizer

Implement a Byte Pair Encoder (BPE) tokenizer as shown in the lecture and apply it to a sample text. You are free with your choice of libraries. You can assume that the corpus only consists of a list of words.

Exercise 2 – BPE Learner

BPE token learner algorithm (Lecture 2 – Slide 48)


```
function BYTE-PAIR ENCODING(strings  $C$ , number of merges  $k$ ) returns vocab  $V$ 

 $V \leftarrow$  all unique characters in  $C$            # initial set of tokens is characters
for  $i = 1$  to  $k$  do                             # merge tokens til  $k$  times
     $t_L, t_R \leftarrow$  Most frequent pair of adjacent tokens in  $C$ 
     $t_{NEW} \leftarrow t_L + t_R$                  # make new token by concatenating
     $V \leftarrow V + t_{NEW}$                      # update the vocabulary
    Replace each occurrence of  $t_L, t_R$  in  $C$  with  $t_{NEW}$  # and update the corpus
return  $V$ 
```

Exercise 2 – BPE Learner

BPE token learner algorithm (Lecture 2 – Slide 48)

```
3 def BPE_learner(corpus, num_merges): returns vocab  $V$ 
```

 $V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
 $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
 $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
 # update the vocabulary with t_{NEW} # and update the corpus

Comes with a few notes!

return V

Exercise 2 – BPE Learner

Notes (Lecture 2 - Slide 49):

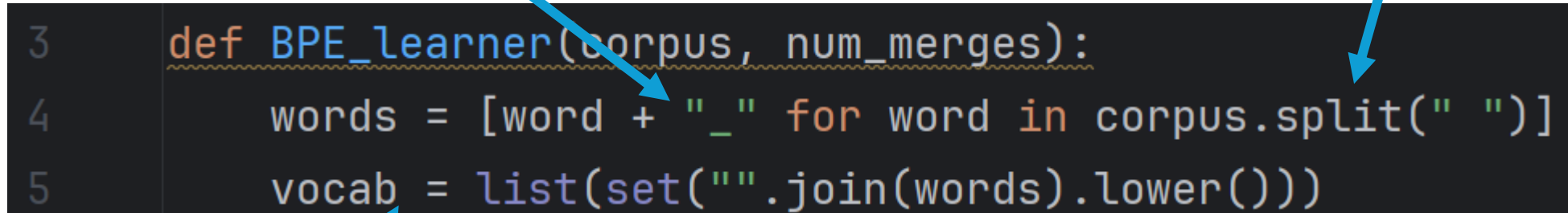
1. Most subword algorithms are run inside space separated tokens.
2. So we commonly first add a special end of word symbol '_' before space in training corpus

Exercise 2 – BPE Learner

Step 1 - Prepare the vocab:

2. Add EOW-symbol for each word

1. space separation



```
3  def BPE_learner(corpus, num_merges):  
4      words = [word + "_" for word in corpus.split(" ")]  
5      vocab = list(set("".join(words).lower()))
```

The code snippet is displayed in a dark-themed editor. Three blue arrows point from external text to the code: one from '1. space separation' to the `split(" ")` call on line 4; one from '2. Add EOW-symbol for each word' to the `word + "_"` expression on line 4; and one from '3. Initial vocab:' to the `list(set(...))` expression on line 5.

3. Initial vocab:
all unique characters in corpus
(incl. EOW-symbol)

Exercise 2 – BPE Learner

BPE token learner algorithm

```
3 def BPE_learner(corpus, num_merges): returns vocab  $V$ 
```

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
 $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
 $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
 $V \leftarrow V + t_{NEW}$ # update the vocabulary
 Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus

re

The actual longest part of the code!

Exercise 2 – BPE Learner

Step 2 – Initialize the tokens:

Initial tokens should look like this (L2 Slide 48):

representation

1. Represent corpus as set of words

2. Compute word counts

corpus

5	l	o	w	_			
2	l	o	w	e	s	t	_
6	n	e	w	e	r	_	
3	w	i	d	e	r	_	
2	n	e	w	_			

3. Split each word into letters

Exercise 2 – BPE Learner

Step 2 – Initialize the tokens:

1. Represent corpus as set of words

```
6 tokens = [list(token) for token in set(words)]  
7 vocab_count = collections.Counter(words)
```

2. Compute word counts

3. Split each word into letters

Exercise 2 – BPE Learner

Step 3 – Merge/Iterate k times:

```
8         for iteration in range(num_merges):
```

Exercise 2 – BPE Learner

Step 3.1 - Compute pair frequencies:

	corpus	Pair-counts
→	5 l o w _	7 lo
→	2 l o w e s t _	5 ow
	6 n e w e r _	5 w_
	3 w i d e r _	
	2 n e w _	

Exercise 2 – BPE Learner

Step 3.1 - Compute pair frequencies:

1. Create a counter for the pairs

Pair-counts

7	lo
5	ow
5	w_

corpus

5	l	o	w	_			
2	l	o	w	e	s	t	_
6	n	e	w	e	r	_	
3	w	i	d	e	r	_	
2	n	e	w	_			

2. Iterate through each word in set

3. Iterate through each pair in word

4. Update counter for every pair

Exercise 2 – BPE Learner

Step 3.1 - Compute pair frequencies:

```
9 counts = {}  
10 for token in tokens:  
11     word = "".join(token)  
12     for t_l, t_r in zip(token[:-1], token[1:]):  
13         pair = t_l + t_r  
14         if pair in counts.keys():  
15             counts[pair] += vocab_count[word]  
16         else:  
17             counts[pair] = vocab_count[word]
```

1. Create a counter for the pairs

2. Iterate through each word

3. Iterate through each pair

4. Update counter

Exercise 2 – BPE Learner

Intermission – exit condition

All words are represented by a single token
-> Nothing can be merged anymore!



```
18 if len(counts) == 0:  
19     print("No more pairings possible")  
20     break
```

Exercise 2 – BPE Learner

BPE token learner algorithm

```
3  def BPE_learner(corpus, num_merges):  returns vocab  $V$ 
```

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
 $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
 $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
 $V \leftarrow V + t_{NEW}$ # update the vocabulary
 Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Exercise 2 – BPE Learner

Step 3.2 - Find most frequent pair:

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

Pair-counts

9 er
9 r_
8 ne
:
2 es

Step 3.3: Append to vocab:

vocabulary

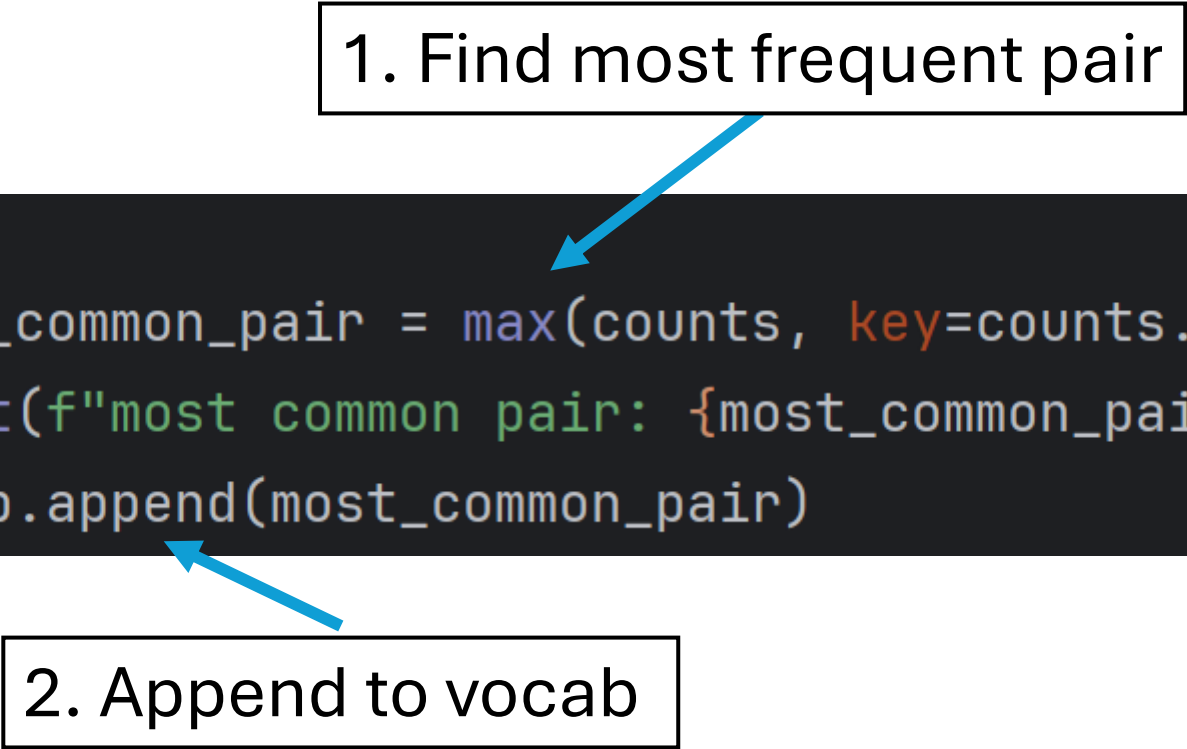
_, d, e, i, l, n, o, r, s, t, w, er

Exercise 2 – BPE Learner

Step 3.2 + 3.3:

1. Find most frequent pair

```
21 else:
22     most_common_pair = max(counts, key=counts.get)
23     print(f"most common pair: {most_common_pair}")
24     vocab.append(most_common_pair)
```



The diagram illustrates the process flow. A box labeled '1. Find most frequent pair' has a blue arrow pointing to the line `most_common_pair = max(counts, key=counts.get)` in the code block. Another blue arrow points from the line `vocab.append(most_common_pair)` in the code block to a box labeled '2. Append to vocab'.

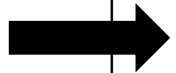
2. Append to vocab

Exercise 2 – BPE Learner

BPE token learner algorithm

```
3  def BPE_learner(corpus, num_merges):  returns vocab  $V$ 
```

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
 $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
 $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
 $V \leftarrow V + t_{NEW}$ # update the vocabulary
 Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V



Exercise 2 – BPE Learner

Step 3.4 - Merge all occurrences of most frequent pair:

most frequent pair: er

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _



corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

Exercise 2 – BPE Learner

Step 3.4 - Merge all occurrences of most frequent pair:

most frequent pair: er

corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _



corpus

5 l o w _
2 l o w e s t _
6 n e w e r _
3 w i d e r _
2 n e w _

1. Iteratively find occurrences...

2. ... and merge them

Exercise 2 – BPE Learner

Step 3.4 - Merge all occurrences of most frequent pair:

```
25 for token in tokens:
26     ind = 1
27     while ind < len(token):
28         if token[ind - 1] + token[ind] == most_common_pair:
29             token[ind - 1] = most_common_pair
30             token.pop(ind)
31         else:
32             ind += 1
```

Iterate over each word

1. find occurrences

2. Merge them

Exercise 2 – BPE Learner

BPE token learner algorithm

```
3 def BPE_learner(corpus, num_merges): returns vocab  $V$ 
```

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
 $t_L, t_R \leftarrow$ Most frequent pair in C
 $t_{NEW} \leftarrow t_L + t_R$ # new token created by concatenating
 $V \leftarrow V + t_{NEW}$ # add new token to vocabulary
 Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

That's it!

Exercise 2 – BPE segmenter

Lecture 2 – Slide 55:

On the test data, run each merge learned from the training data:

- Greedily
- In the order we learned them
- (test frequencies don't play a role)

Exercise 2 – BPE segmenter

Algorithm:

1. Replace spaces with EOW-symbol
2. Split input into letters
3. Merge letters in learned order

Exercise 2 – BPE segmenter

Input : low low

Vocab : {..., lo, low, newer_, low_, ...}

1. Replace spaces : low_low_

2. Split : l, o, w, _, l, o, w, _

3. Merge : lo, w, _, lo, w, _

: low, _, low, _

: low_, low_

Exercise 2 – BPE segmenter

```
39  ✓ def BPE_tokenizer(text, vocab): 2 usages
40      tokens = list(text.lower().replace(" ", "_"))
41      i = 0
42      ✓ while i < len(tokens):
43      ✓         if not tokens[i] in vocab:
44                  tokens[i] = "UNK"
45                  i += 1
46      ✓ for entry in vocab:
47      ✓         if len(entry) == 1:
48                  continue
49      ✓         else:
50                  i = 1
51      ✓         while i < len(tokens):
52      ✓                 if tokens[i - 1] + tokens[i] == entry:
53                          tokens[i - 1] = entry
54                          tokens.pop(i)
55                          i += 1
56      return tokens
```

Insert EOW and split


Deal with unknown letters
(in general unnecessary)

Iteratively merge

Exercise 3 - Using pre-implemented tokenizers

Use an existing tokenizer from the T5 Transformer or any other tokenizer of choice from the HuggingFace library. Apply the tokenizer to a text sample of choice. Compare the output of this tokenizer with the two tokenizers you implemented in the previous questions and explain the similarities and differences.


Transformers Library

 **Hugging Face**

[Models](#) [Datasets](#) [Spaces](#) [Posts](#) [Docs](#) [Enterprise](#) [Pricing](#) [Log In](#) [Sign Up](#)

Transformers ▾

Ctrl+K

V4.51.3 ▾ EN ▾  144,050

GET STARTED

Transformers

Installation

Quickstart

BASE CLASSES

INFERENCE

TRAINING

QUANTIZATION

EXPORT TO PRODUCTION


RESOURCES


CONTRIBUTE


API

Join the Hugging Face community

and get access to the augmented documentation experience

 Collaborate on models, datasets and Spaces

 Faster examples with accelerated inference

 Switch between documentation themes

Sign Up

to get started

Transformers

Transformers is a library of pretrained natural language processing, computer vision, audio, and multimodal models for inference and training. Use Transformers to train models on your data, build inference applications, and generate text with large language models.

Explore the [Hugging Face Hub](#) today to find a model and use Transformers to help you get started right away.

Features

Transformers

Features

Design

29

Transformers Library

- Installation:

<https://huggingface.co/docs/transformers/en/installation>

- Quickstart

<https://huggingface.co/docs/transformers/en/quicktour#pretrained-models>

```
from transformers import AutoModelForCausalLM, AutoTokenizer
```

```
model = AutoModelForCausalLM.from_pretrained("meta-llama/Llama-2-7b-hf",
```

```
tokenizer = AutoTokenizer.from_pretrained("meta-llama/Llama-2-7b-hf")
```

Transformers Library

- We want the „T5“ transformer

 `from_pretrained`

[<source>](#)

```
( pretrained_model_name_or_path: typing.Union[str, os.PathLike, NoneType], *model_args, config:
typing.Union[transformers.configuration_utils.PretrainedConfig, str, os.PathLike, NoneType] =
None, cache_dir: typing.Union[str, os.PathLike, NoneType] = None, ignore_mismatched_sizes: bool
= False, force_download: bool = False, local_files_only: bool = False, token: typing.Union[str,
bool, NoneType] = None, revision: str = 'main', use_safetensors: typing.Optional[bool] = None,
weights_only: bool = True, **kwargs )
```

Parameters

- **`pretrained_model_name_or_path`** (`str` or `os.PathLike`, *optional*) — Can be either:
 - A string, the *model id* of a pretrained model hosted inside a model repo on huggingface.co.
 - A path to a *directory* containing model weights saved using `save_pretrained()`, e.g., `./my_model_directory/`.

Transformers Library



- Need T5 transformer

`pretrained_model_name_or_path` (`str` or `os.PathLike`, *optional*) — Can be either:


- A string, the *model id* of a pretrained model hosted inside a model repo on `huggingface.co`.


- Head to <https://huggingface.co/google-t5>


Transformers Library


 **Models** 5 


↑↓ Sort: Recently updated

 **google-t5/t5-base**
🌐 Translation • Updated Feb 14, 2024 • ⬇ 4.3M • ⚡ • ❤ 708

 **google-t5/t5-3b**
🌐 Translation • Updated Jan 29, 2024 • ⬇ 357k • ❤ 46

 **google-t5/t5-small**
🌐 Translation • Updated Jun 30, 2023 • ⬇ 3.02M • ⚡ • ❤ 455

 **google-t5/t5-large**
🌐 Translation • Updated Apr 6, 2023 • ⬇ 436k • ⚡ • ❤ 205

 **google-t5/t5-11b**

Choose any

Transformers Library

The image is a screenshot of the Hugging Face model card for 'google-t5/t5-small'. It features a header with the model name, a 'like' button, and a 'Follow' button. Below this is a row of tags for various frameworks and languages. A box labeled 'Repo string we need' points to the model name 'google-t5/t5-small'. Another box labeled 'Click for usage reference' points to the 'Use this model' button. The main content area includes a 'Model Card for T5 Small' section and a 'Downloads last month' chart showing 3,023,383 downloads.

Repo string we need

Click for usage reference

<https://huggingface.co/google-t5/t5-small>

Exercise 3

```
1 from transformers import T5Tokenizer
2
3 tokenizer = T5Tokenizer.from_pretrained("google-t5/t5-small")
4 test_text = "He has a M.Sc. in Math and a Ph.D. in NLP. A sess
5 #print(tokenizer.convert_ids_to_tokens(tokenizer(test_text).in
6 print(tokenizer.tokenize(test_text))
```

Exercise 3 - Similarities

...with WS-tokenizer:

- Mostly assigns (EOS-)symbols their own token
- Most splits happen on word-level

... with BPE:

- (T5 uses an extension of BPE called ,unigram‘ or ,SentencePiece‘)
- Learns (sub-)word splits
- Vocab size fixed before training
- Tries to minimize sequence length (token count)

Exercise 3 - Differences

...with WS-tokenizer:

- Splits are learned and not rule-based
- Does not care about abbreviations, links, emails, dates, ...
- Includes spaces in the tokens

... with BPE:

- Start-of-word instead of EOW-symbol
- Has some (model specific) special tokens (e.g. end-of-sequence `</s>`)
- (Trims a „seed“ vocabulary (i.e. built using BPE) instead of building one)

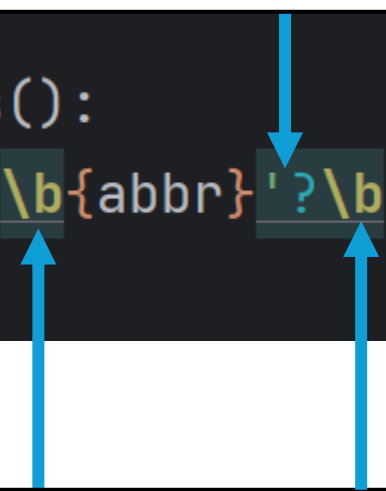
Exercise 4 - RegEx

Assume we have a lookup table named `lookup` storing abbreviation definitions. Using regular expressions in Python, write code that uses `lookup` to replace abbreviations in any given text with their full-text counterparts. Apply your code to the following snippet so that `example` is transformed to match `target_output`:

Exercise 4 - RegEx

Abbr. could be followed by an apostrophe

```
15 def convert_to_fulltext(text):  
16     for abbr, ft in lookup.items():  
17         text = re.sub(pattern: fr"\b{abbr}'?\b", ft, text)  
18     return text
```



Abbr. may not be preceded or followed by word characters
(match „bf“ but not „bff“)

„\b“: word boundary; matches anything but [A-Za-z0-9_]

-> excludes word letters

-> thus facilitates whole word search

Regex

- Powerful tool
- Can solve ex. 1 in 3 lines:

```
81 # Regex (optional)
82 import re
83 tokens=re.findall( pattern: r'https?://\S+|[\w.+-]+@[ \w.-]+\.\w{2,}|(?:Ph\.\D\.|M\.\Sc\.\Dr\.)|\d{2}/\d{2}/\d{2}|\$?\d+\.\d{2}\$?|#\w+|[\.,;:!?()[\]/]| \w+',test_text
84 print(all([token == ref_token for token, ref_token in zip(tokens, ref)]))
```