BytePairEncoding

April 20, 2024

```
<h1>Natural Language Processing</h1>
Assignment 01
<h3>General Information:</h3>
<p>Please do not add or delete any cells. Answers belong into the corresponding cells (below th
<p>Code cells where you are supposed to give your answer often include the line  ```raise NotIm
<h3>Submission:</h3>
<p>Please submit your notebook via the web interface (in the main view -> Assignments -> Submit
<h3>Group Work:</h3>
<p>You are allowed to work in groups of up to two people. Please enter the UID (your username f
<h3>Questions about the Assignment:</h3>
<p>If you have questions about the assignment please post them in the LEA forum before the deac
```

[1]:
```
'''
Group Work:
Enter the username of each team member into the variables.
If you work alone please leave the second variable empty.
'''
member1 = 'mfarra2s'
member2 = ''
member3 = ''
```

1 Byte Pair Encoding

We want to implement BPE.

1.1 Byte Pair Encoding A) [10 points]

First we want to do pre-tokenization using white spaces.

Please complete the function `pretokenize` below. This takes a list of sentences or documents and returns a list of tokenized sentences or documents. Look at the example in the docstring for more information.

[2]:
```
from typing import List

def pretokenize(sentences: List[str]) -> List[List[str]]:
    """
    Tokenizes a list of sentences into a list of lists of tokens.
```

```
    Args:
        sentences (List[str]): List of sentences to be tokenized.

    Returns:
        List[List[str]]: List of lists of tokens, where each inner list␣
 ↪represents
                         the tokens of a single sentence.
    Example:
        >>> sentences = ["Hello world", "This is a test"]
        >>> pretokenize(sentences)
        [['Hello', 'world'], ['This', 'is', 'a', 'test']]
    """
    # YOUR CODE HERE
    tokenized_sentences = []
    for sentence in sentences:
        tokens = sentence.split()
        tokenized_sentences.append(tokens)
    return tokenized_sentences

#     raise NotImplementedError()

example_sentences = [
    "This is an  example sentence",
    "Another sentence",
    "The final sentence"
]

tokenized = pretokenize(example_sentences)
tokenized
```

[2]:
```
[['This', 'is', 'an', 'example', 'sentence'],
 ['Another', 'sentence'],
 ['The', 'final', 'sentence']]
```

[ ]:
```

```

1.2 Byte Pair Encoding B) [10 points]

For BPE we first need an initial vocabulary. The input is a pretokenized list of sentences / documents.

The output should be a set of characters present in this list.

[3]:
```
from typing import List, Set

def build_initial_vocabulary(corpus: List[List[str]]) -> Set[str]:
    """
    Build the initial vocabulary from a corpus of tokenized sentences.
```

```
    Args:
        corpus (List[List[str]]): A list of tokenized sentences, where each␣
↪sentence
            is represented as a list of strings (tokens).

    Returns:
        Set[str]: A set containing all unique tokens in the corpus.

    Example:
        >>> corpus = [['hello', 'world'], ['This', 'is', 'a', 'test']]
        >>> build_initial_vocabulary(corpus)
        {'T', 'a', 'd', 'e', 'h', 'i', 'l', 'o', 'r', 's', 't', 'w'}
    """
    # YOUR CODE HERE
    vocabulary = set()
    for sentence in corpus:
        for word in sentence:
            for letter in word:
                vocabulary.add(letter)
    return vocabulary
    raise NotImplementedError()

build_initial_vocabulary(pretokenize(["hello world", "This is a test"]))
```

[3]:
```
{'T', 'a', 'd', 'e', 'h', 'i', 'l', 'o', 'r', 's', 't', 'w'}
```

[ ]:
```

```

1.3 Byte Pair Encoding C) [10 points]

Now we want to build our dictionary for the split tokens. Complete the function `get_splits` below. Look at the example in the docstring!

Make sure to add the end of word symbol (`</w>`) to each token.

[4]:
```
from collections import Counter
from typing import Dict, Tuple

def get_splits(corpus: List[List[str]]) -> Dict[Tuple[str], int]:
    """
    Get subword splits of tokens in a corpus.

    Args:
        corpus (List[List[str]]): A list of sentences where each sentence is␣
↪represented
            as a list of tokens.
```

```
    Returns:
        Dict[Tuple[str], int]: A dictionary where keys are tuples representing␣
↪subword splits
            and values are the counts of occurrences of those splits in the␣
↪corpus.

    Example:
        >>> corpus = [['apple', 'banana', 'apple'], ['apple']]
        >>> get_splits(corpus)
        {('a', 'p', 'p', 'l', 'e', '</w>'): 3, ('b', 'a', 'n', 'a', 'n', 'a',␣
↪'</w>'): 1}
    """
    # YOUR CODE HERE
    elements = []

    for sentence in corpus:
        for word in sentence:
            elements.append(word)

    elements_counts = dict(Counter(elements))

    words_count = dict()
    word_letters = []
    for key in elements_counts.keys():
        word_letters = [l for l in key]
        word_letters.append('</w>')
        word_letters = tuple(word_letters)
        words_count[word_letters] = elements_counts.get(key)

    return words_count
    raise NotImplementedError()

get_splits(pretokenize(["apple banana apple", "apple"]))
```

[4]:
```
{('a', 'p', 'p', 'l', 'e', '</w>'): 3,
 ('b', 'a', 'n', 'a', 'n', 'a', '</w>'): 1}
```

[ ]:
```

```

1.4 Byte Pair Encoding D) [10 points]

In the next step we want to find the most common pair from a splits dictionary.

Complete the function `find_most_frequent_pair` which returns the most frequent pair alongside its count (e.g. `(('a', 'n'), 2)`)

[5]:
```python
def find_most_frequent_pair(splits: Dict[Tuple[str], int]) -> Tuple[Tuple[str, str], int]:
    """
    Find the most frequent pair of characters from a dictionary of split words along with its count.

    Args:
        splits (Dict[Tuple[str], int]): A dictionary where keys are tuples of split words and values are their counts.

    Returns:
        Tuple[Tuple[str, str], int]: A tuple containing the most frequent pair of characters and its count.

    Example:
        >>> splits = {('a', 'p', 'p', 'l', 'e', '</w>'): 3,
                      ('b', 'a', 'n', 'a', 'n', 'a', '</w>'): 1}
        >>> find_most_frequent_pair(splits)
        (('a', 'n'), 2)
    """
    # YOUR CODE HERE
#     voc = splits.keys()
#     letters= []
#     for tup in voc:
#         lis = list(tup)
#         for chara in lis:
#             letters.append(chara)

#     pair = []
#     check = []
#     count = 0
#     pair_count = {}
#     for ind, l in enumerate(letters):
#         if ind < len(letters)-1:
#             pair = [letters[ind],letters[ind +1]]
#             for i, j in enumerate(letters):
#                 if i < len(letters)-1:
#                     check = [letters[i],letters[i +1]]
#                     if check == pair:
#                         count = count + 1
#             t_pair = tuple(pair)
#             pair_count[t_pair]= count
#             pair.clear
#             check.clear
#             count = 0

#     most_frequent_pair = max(pair_count, key=pair_count.get)
```

```
#     frequency = pair_count[most_frequent_pair]

#     return (most_frequent_pair,frequency)
    voc = splits.keys()
    words = []
    letters= []

    for tup in voc:
        lis = list(tup)
        for _ in range(splits[tup]):
            words.append(lis)


    for word in words:
        for letter in word:
            letters.append(letter)

    pair = []
    check = []
    count = 0
    pair_count = {}
    for ind, l in enumerate(letters):
        if ind < len(letters)-1:
            pair = [letters[ind],letters[ind +1]]
            for i, j in enumerate(letters):
                if i < len(letters)-1:
                    check = [letters[i],letters[i +1]]
                    if check == pair:
                        count = count + 1
            t_pair = tuple(pair)
            pair_count[t_pair]= count
            pair.clear
            check.clear
            count = 0

    most_frequent_pair = max(pair_count, key=pair_count.get)
    frequency = pair_count[most_frequent_pair]

    return (most_frequent_pair,frequency)
    raise NotImplementedError()

find_most_frequent_pair(get_splits(pretokenize(["apple banana apple",␣
 ↪"apple"])))
```

[5]:
```
(('a', 'p'), 3)
```

[ ]:
```

```

1.5 Byte Pair Encoding E) [15 points]

Now write a function that takes a pair and the splits and merges all occurences of the pair in the splits.

[6]:
```
def merge_split(split: Tuple[str], pair: Tuple[str, str]):
    """
    Merge a split tuple if it contains the given pair.

    Args:
        split (Tuple[str]): The split tuple to merge.
        pair (Tuple[str, str]): The pair to merge.

    Returns:
        Tuple[str]: The merged split tuple.

    Example:
        >>> merge_split(split=('a', 'b', 'c', 'b', 'c'), pair=('b', 'c'))
        ('a', 'bc', 'bc')
    """
    # YOUR CODE HERE
    list_split = list(split)
    list_pair = list(pair)
    new_split = []

    for ind, letter in enumerate(list_split):
        if ind < len(list_split)-1:
            check = [list_split[ind],list_split[ind+1]]
            if check == list_pair:
                new_split.append(list_split[ind] + list_split[ind+1])
            else:
                if check[0] == list_pair[-1] and [list_split[ind-1],␣
↪list_split[ind]] == list_pair:
                    continue
                else:
                    new_split.append(list_split[ind])
        else:
            check = [list_split[ind-1],list_split[ind]]
            if check == list_pair:
                continue
            else:
                if check[0] == list_pair[-1] and [list_split[ind-1],␣
↪list_split[ind]] == list_pair:
                    continue
                else:
                    new_split.append(list_split[ind])

    return tuple(new_split)
```

```
    raise NotImplementedError()

def merge_splits(splits: Dict[Tuple[str], int], pair: Tuple[str, str]):
    """
    Merge all split tuples in a dictionary that contain the given pair.

    Args:
        splits (Dict[Tuple[str], int]): A dictionary of split tuples and their counts.
        pair (Tuple[str, str]): The pair to merge.

    Returns:
        Dict[Tuple[str], int]: A dictionary with merged split tuples and their counts.

    Example:
        >>> merge_splits({('a', 'p', 'p', 'l', 'e', '</w>'): 3,
                          ('b', 'a', 'n', 'a', 'n', 'a', '</w>'): 1},
                          ('a', 'n'))
        {('a', 'p', 'p', 'l', 'e', '</w>'): 3,
         ('b', 'an', 'an', 'a', '</w>'): 1}
    """
    # YOUR CODE HERE
    merged_split = {}
    for key in splits.keys():
        new_key = merge_split(key, pair)
        merged_split[new_key] = splits[key]
    return merged_split
    raise NotImplementedError()

splits = get_splits(pretokenize(["apple banana apple", "apple"]))
print(splits)
most_frequent_pair, count = find_most_frequent_pair(splits)
print(most_frequent_pair, count)
print(merge_splits(splits, most_frequent_pair))
```

```
{('a', 'p', 'p', 'l', 'e', '</w>'): 3, ('b', 'a', 'n', 'a', 'n', 'a', '</w>'):
1}
('a', 'p') 3
{('ap', 'p', 'l', 'e', '</w>'): 3, ('b', 'a', 'n', 'a', 'n', 'a', '</w>'): 1}
```

[ ]:

1.6 Byte Pair Encoding E) [40 points]

Now let us put this all together into a single class. Complete the methods `train`, `encode` and `decode`.

- `train` will learn the vocabulary and a list of merged pairs to use for encoding / tokenizing.
- `encode` will tokenize a list of strings using the merge rules by applying them in order
- `decode` will take a BPE encoded list of lists and merge subwords

Look at the examples in the docstrings for more information.

[7]:
```
class BPETokenizer:
    """
    Byte-Pair Encoding (BPE) Tokenizer.

    This tokenizer learns a vocabulary and encodes/decodes text using the␣
 ↪Byte-Pair Encoding algorithm.
    """

    def __init__(self):
        """
        Initialize the BPETokenizer.
        """
        self.vocab: set = set()
        self.end_of_word: str = "</w>"
        self.merge_pairs: List[Tuple[str, str]] = []

    def train(self, corpus: List[str], max_vocab_size: int) -> None:
        """
        Train the tokenizer on a given corpus.
        First pretokenizes the corpus using whitespace
        Then uses BPE to update the vocabulary and learn the merge pairs

        Args:
            corpus (List[str]): The corpus of text for training.
            max_vocab_size (int): The maximum size of the vocabulary.

        Returns:
            None

        Example:
        >>> corpus = [
            "lowest lower newer newest",
            "low lower new"
        ]
        >>> tokenizer.train(corpus, max_vocab_size=20)
        """
        # YOUR CODE HERE
        self.tokenized = pretokenize(corpus)
        self.vocab = build_initial_vocabulary(self.tokenized)
        splits = get_splits(self.tokenized)
#         print('splits =', splits)
```

```
        for i in range(max_vocab_size):

            most_frequent_pair, count = find_most_frequent_pair(splits)
            self.merge_pairs.append(most_frequent_pair)
#             print('merge_pairs =',self.merge_pairs)

            new_pair = ''.join(most_frequent_pair)
            self.vocab.add(new_pair)
#             print('vocab =', self.vocab)

            splits = merge_splits(splits, most_frequent_pair) #get_splits(self.
 ↪tokenized)
#             print('splits =', splits)
            if len(self.vocab) == max_vocab_size:
                break

    def encode(self, corpus: List[str]) -> List[List[str]]:
        """
        Encode / Tokenize a corpus of text using the learned vocabulary and␣
 ↪merge pairs.

        Args:
            corpus (List[str]): The corpus of text to be encoded.

        Returns:
            List[List[str]]: The encoded corpus.

        Example:
        >>> corpus = [
            "lowest lower newer newest",
            "low lower new"
        ]
        >>> tokenizer.train(corpus, max_vocab_size=20)
        >>> tokenizer.encode(corpus)
        [['lowest</w>', 'lower</w>', 'newer</w>', 'newe', 'st</w>'],
         ['lo', 'w</w>', 'lower</w>', 'ne', 'w</w>']]

        """
        # YOUR CODE HERE
        self.tokenized = pretokenize(corpus)
        self.splits = get_splits(self.tokenized)
        self.encodes = []
        for pair in self.merge_pairs:
            self.splits = merge_splits(self.splits, pair)
        print(self.splits)

        for enc in self.splits.keys():
```

```
            l_enc = list(enc)
            self.encodes.append(l_enc)
        return self.encodes
        raise NotImplementedError()

    def decode(self, tokenized: List[List[str]]) -> List[List[str]]:
        """
        Decode a corpus of tokenized text.

        Args:
            tokenized (List[List[str]]): The tokenized text to be decoded.

        Returns:
            List[List[str]]: The decoded text.

        Example:
        >>> corpus = [
            "lowest lower newer newest",
            "low lower new"
        ]
        >>> tokenizer.train(corpus, max_vocab_size=20)
        >>> tokenizer.decode([['lowest</w>', 'lower</w>', 'newer</w>', 'newe',␣
↪'st</w>'],
                              ['lo', 'w</w>', 'lower</w>', 'ne', 'w</w>']])
        [['lowest', 'lower', 'newer', 'newest'], ['low', 'lower', 'new']]      ␣
↪
        """
        # YOUR CODE HERE
        letters = []
        decode = []
        self.decodes = []
        for enc in tokenized:
            for word in enc:
                for letter in word:
                    letters.append(letter)
        print(letters)
        # letters = ''.join(letters)

        for letter in letters:
            if letter == '<':
                ind = letters.index('<')
                decode.append(letters[0:ind])
        #         print(decode)
                letters = letters[ind+4:]

        for dec in decode:
            w = ''.join(dec)
```

```
            self.decodes.append(w)
        return self.decodes
        raise NotImplementedError()

corpus = [
    "lowest lower newer newest",
    "low lower new"
]
tokenizer = BPETokenizer()
tokenizer.train(corpus, 20)
# tokenizer.encode(corpus)
tokenizer.decode(tokenizer.encode(corpus))
```

```
{('lowest</w>',): 1, ('lower</w>',): 2, ('newe', 'r</w>'): 1, ('newe',
'st</w>'): 1, ('lo', 'w</w>'): 1, ('ne', 'w</w>'): 1}
['l', 'o', 'w', 'e', 's', 't', '<', '/', 'w', '>', 'l', 'o', 'w', 'e', 'r', '<',
'/', 'w', '>', 'n', 'e', 'w', 'e', 'r', '<', '/', 'w', '>', 'n', 'e', 'w', 'e',
's', 't', '<', '/', 'w', '>', 'l', 'o', 'w', '<', '/', 'w', '>', 'n', 'e', 'w',
'<', '/', 'w', '>']
```

[7]:
```
['lowest', 'lower', 'newer', 'newest', 'low', 'new']
```

[ ]:
```

```

1.7 Byte Pair Encoding F) [5 points]

Use your BPE tokenizer on the movie script of spider. Then encode a random sentence using the tokenizer. Finally decode the sentence again.

Training might take ~3 minutes.

[ ]:
```
with open("/srv/shares/NLP/datasets/yelp/reviews_sents.txt", "r") as f:
    sentences = f.read().split("\n")

# YOUR CODE HERE
tokenizer = BPETokenizer()
tokenizer.train(sentences, 20)
tokenizer.decode(tokenizer.encode(sentences))
raise NotImplementedError()
```

[ ]:
```

```