Natural Language Processing Course

CA#1 (BPE & WordPiece Tokenizers) Report

Parnian Fazel – 810198516

Step #1

Some of the popular subword-based tokenization algorithms are WordPiece, Byte-Pair Encoding (BPE), Unigram, and SentencePiece. We will go through BPE and WordPiece algorithm in this assignment.

Explanation of BPE & WordPiece Algorithms:

- BPE (Byte Per Encoding):

Basically, BPE is a sub-word and a morphological tokenizer, and the main idea of it is that it merges adjacent byte pairs based on their frequency in a training corpus. BPE takes a pair of tokens (bytes), looks at the frequency of each pair, and merges the pair which has the highest combined frequency. The process is greedy for the highest combined frequency at each step.

The overview of how it works: first we should detect all words in the corpus and add a symbol to show the end of word like "</w>" or "_". Then we make a vocabulary of all bytes (symbols and characters) in these words, finally we count the frequency of the consecutive byte pairs and merge the most frequently occurring one. We should do this procedure until we reach a token limit.

For the given corpus the steps of BPE is like this:

Corpus

low	lower	newest
low	lower	newest
low	widest	newest
low	widest	newest
low	widest	newest

1. The “</w>” symbol is concatenated to all words.
2. All words will be splitted into symbols(characters):
   l o w </w> l o w </w> l o w </w> l o w </w> l o w </w>
   l o w e r </w> l o w e r </w>
   w i d e s t <w/> w i d e s t <w/> w i d e s t <w/>
   n e w e s t </w> n e w e s t </w> n e w e s t </w> n e w e s t </w> n e w e s t </w>
3. The vocabulary is created by unique symbols of all words in the corpus:
   Vocabulary = {l, o, w, e, r, i, d, s, t, n, </w>}
4. Now we compute the frequency of every consecutive pair and merge the most frequent one.
   “l o w </w>”: 5
   “l o w e r </w>”: 2
   “w i d e s t <w/>”: 3
   “n e w e s t </w>”: 5
   So, the most frequent pair is “es” with frequency of 8. Then is “est” with frequency of 8, and so on… . Finally, the vocabulary for some more iterations would be:
   Vocabulary = {l, o, w, e, r, i, d, s, t, n, </w>, es, est, est</w>, lo, low, ne, new, newest</w>, low</w>}

Here is the algorithm given in the *Neural Machine Translation of Rare Words with Subword Units* paper:

Algorithm 1 Learn BPE operations

```
import re, collections

def get_stats(vocab):
  pairs = collections.defaultdict(int)
  for word, freq in vocab.items():
    symbols = word.split()
    for i in range(len(symbols)-1):
      pairs[symbols[i],symbols[i+1]] += freq
  return pairs

def merge_vocab(pair, v_in):
  v_out = {}
  bigram = re.escape(' '.join(pair))
  p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
  for word in v_in:
    w_out = p.sub(''.join(pair), word)
    v_out[w_out] = v_in[word]
  return v_out

vocab = {'l o w </w>' : 5, 'l o w e r </w>' : 2,
         'n e w e s t </w>':6, 'w i d e s t </w>':3}
num_merges = 10
for i in range(num_merges):
  pairs = get_stats(vocab)
  best = max(pairs, key=pairs.get)
  vocab = merge_vocab(best, vocab)
  print(best)
```

What happen to OOV words? BPE ensures that the most common words are represented in the vocabulary as a single token while the rare words and OOV words are broken down into two or more sub-word tokens. In the given example, for encoding word “lowest”, it will be broken down to “low” and “est</w>”.

- WordPiece:

This algorithm is pretty much like BPE except that at each iterative step, it chooses a pair which will result in the largest increase in likelihood upon merging. This probability score is:

$$Score = \frac{frequency\ of\ pair}{frequency\ of\ first\ elemnt \times frequency\ of\ second\ element}$$

Overview of how it works: first we should detect all words in the corpus and add a symbol to show the end of word like “</w>” or “_”. Then we make a vocabulary of all bytes (symbols and characters) in these words (for start symbols in word, the exact symbol but for symbols which do not start the word, two “#” symbols should be added before the symbols). Finally, we calculate the score of the consecutive symbols and merge the pairs with highest score. We should do this procedure until we reach a token limit.

Similarities: Both are sub-word tokenizer, and they are both greedy. The procedure of both algorithms is pretty much the same: first they make vocabulary of symbols then they merge consecutive symbols (pairs) based on a specific score.

Differences: The main difference is at each step of merging. BPE algorithm merges the pairs with the highest frequency, but WordPiece algorithm merges pairs which will result in the largest increase in likelihood once merged.

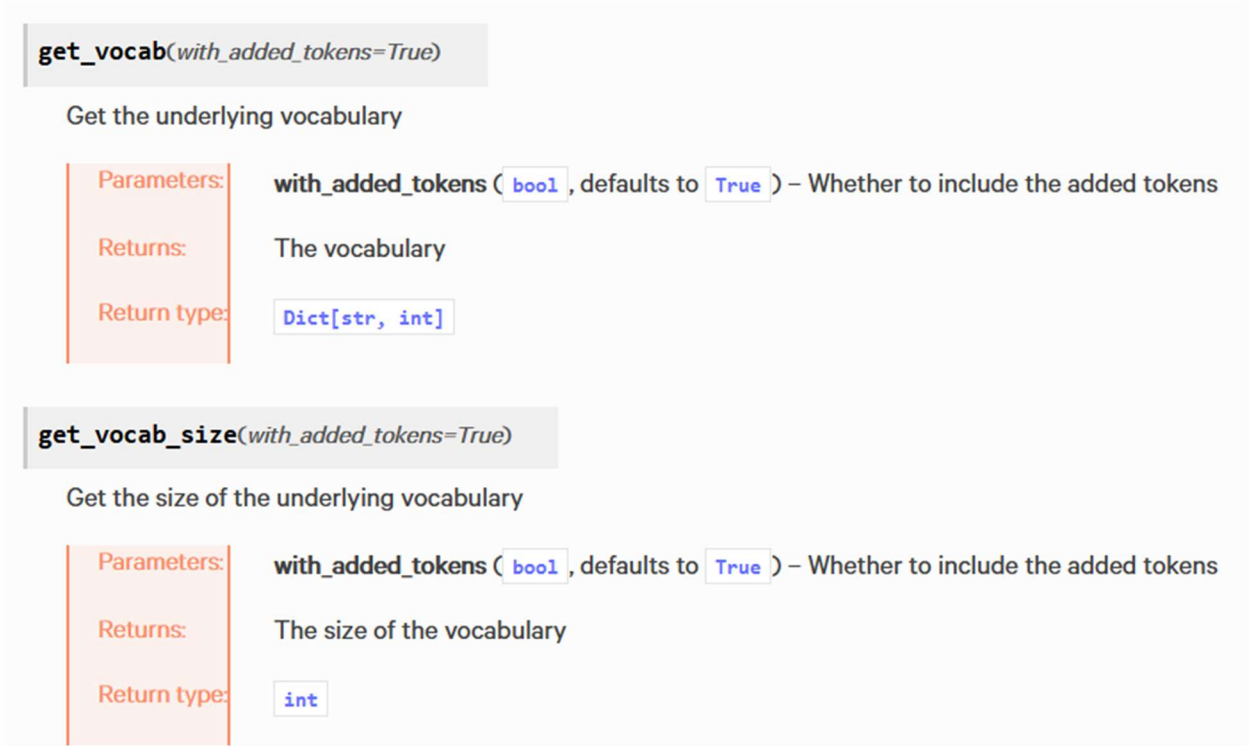
Step #2

In this step we are to use hugging face library. First for installing it I used the command:

!pip install tokenizers

Also, I installed hugging face hub.

There are some methods for trainers to study the results of model, for instance I used these two methods:



I saved more information about the model on a jason file for each model which can be seen in the files.

First, I should do the learner part of the two BPE & WordPiece models on the **text box**. The codes are in the notebook & the results are:

- BPE Model

  The number of extracted tokens: 113
  The extracted tokens: {'k': 18, 'he': 52, 'del': 94, 'Excited': 110, 'ial': 80, 'el': 73, 'pipeline': 107, 'il': 78, 'or': 84, 'io': 41, 'comparing': 111, 'pip': 85, 'tutor': 108, 'deep': 57, '😍': 32, 'ization': 48, 'by': 70, '?!': 60, 'T': 7, 'tokens': 95, '?':

2, 'ated': 92, 'ed': 50, 'each': 75, 'lear': 53, 'ning': 54, 'z': 31, 'be': 69, 'much': 106, 'fi': 76, 'ted': 88, 'f': 14, 'model': 105, 'i': 17, 'tor': 89, 'ear': 51, 'NLP': 63, 'oken': 37, 'mo': 81, 'W': 8, 'a': 9, 'wil': 90, 'mu': 83, 'd': 12, 'e': 13, 'learning': 59, 'This': 98, '!': 0, 'LP': 62, 'Th': 64, 'b': 10, 'o': 22, 'n': 21, 'ach': 68, 'P': 6, 'Token': 65, 'tokenization': 58, 'l': 19, 'c': 11, 'rst': 86, 'ken': 36, 'erated': 102, 'step': 96, 'ing': 45, 'ip': 79, 'gen': 77, 'Ex': 61, 'comp': 100, 'eline': 101, 'y': 30, 'tutorial': 112, 'E': 3, 'ine': 91, 'Exci': 97, 'izatio': 47, 'generated': 104, 'Tokenization': 99, 'ar': 38, 'p': 23, 'co': 72, 'L': 4, 'ch': 49, 'al': 67, 'atio': 46, '.': 1, 'is': 42, 't': 26, 'the': 56, 'ci': 71, 'x': 29, 'will': 109, 'm': 20, 'en': 33, 'u': 27, 'at': 35, 'N': 5, 'h': 16, 'first': 103, 'g': 15, 'token': 44, 'er': 74, 'iz': 43, 'de': 39, 'aring': 93, 'mp': 82, 'We': 66, 'in': 34, 'r': 24, 'st': 55, 's': 25, 'w': 28, 'ep': 40, 'tu': 87}

- WordPiece Model

The number of extracted tokens: 139
The extracted tokens: {'l': 19, 'is': 71, '##rs': 105, 'by': 90, 'each': 124, '##u': 33, '##k': 42, 'f': 14, '##c': 52, '##al': 109, '##p': 46, 'deep': 80, 'x': 29, 'e': 13, '##e': 43, '##ar': 65, 'k': 18, 'o': 22, '##y': 49, 'u': 27, 'Excited': 119, 'm': 20, '😍': 32, 'generated': 136, '##L': 53, 'de': 70, '##n': 44, '##ted': 103, 'generat': 126, 'ea': 92, 'r': 24, 'NL': 85, 'pipeline': 137, 'pi': 98, 'd': 12, '##mp': 112, 'NLP': 120, '##ri': 104, '.': 1, 'co': 91, '##ited': 108, 'y': 30, '##ll': 110, 'W': 8, '##iz': 63, 'the': 82, '##i': 37, 'learning': 81, 'E': 3, '##to': 102, '##ine': 116, '##cited': 115, '##ch': 77, 'g': 15, '##ization': 69, 'tuto': 131, 'model': 128, '##erat': 111, 'th': 73, 'h': 16, 'Ex': 84, 'z': 31, '##ion': 64, 'c': 11, 'p': 23, 'first': 135, '##ok': 57, '##pel': 113, '##at': 59, '?!': 83, '##d': 50, '##!': 55, '##a': 38, '!': 0, 'b': 10, '##P': 54, 'tutorial': 138, 'step': 130, 'comparing': 134, '##o': 35, '##in': 58, '##oken': 60, 'will': 132, '##del': 114, '##ed': 75, '##aring': 118, 'We': 88, 'Th': 86, 't': 26, '##rat': 106, '##t': 34, 'mu': 96, 'Tokenization': 122, '##on': 62, 'This': 121, 'firs': 125, 'n': 21, 'w': 28, '?': 2, 'be': 89, '##h': 40, 'le': 72, '##g': 47, '##arning': 79, '##rial': 133, '##ning': 76, 'i': 17, 'st': 99, '##ep': 66, 'tokenization': 78, 'much': 127, '##x': 51, 'pipel': 129, '##el': 74, 'a': 9, 'comp': 123, '##s': 41, '##is': 107, '##z': 48, 'wi': 101, 's': 25, 'tokens': 117, 'Token': 87, 'gen': 94, 'tu': 100, '##ing': 67, 'N': 5, '##ation': 68, 'T': 7, 'fi': 93, '##en': 56, 'P': 6, 'token': 61, '##m': 45, 'L': 4, 'mo': 97, '##l': 39, 'in': 95, '##r': 36}

As we can see the number of extracted tokens for BPE model is less than WordPiece model (113 < 139). In total, They did a good job and they both extracted important sub-words such as “ing” and “tokenization”. Also, they both could tokenize emoji! They both extracted words like: “Excited” and “comparing”. A word like “NLP” is extracted much sooner by BPE than WodPiece! (for BPE on 63rd and for WordPiece on 120th merges). In a whole both of them tokenized well for this little text, but we should see the results for bigger texts!

Now, I should do the learner part of the two BPE & WordPiece models on the **Gutenberg book text**.

The codes are in the notebook & the results are: (the extracted tokens were too much so I did not put it in the report).

- BPE Model

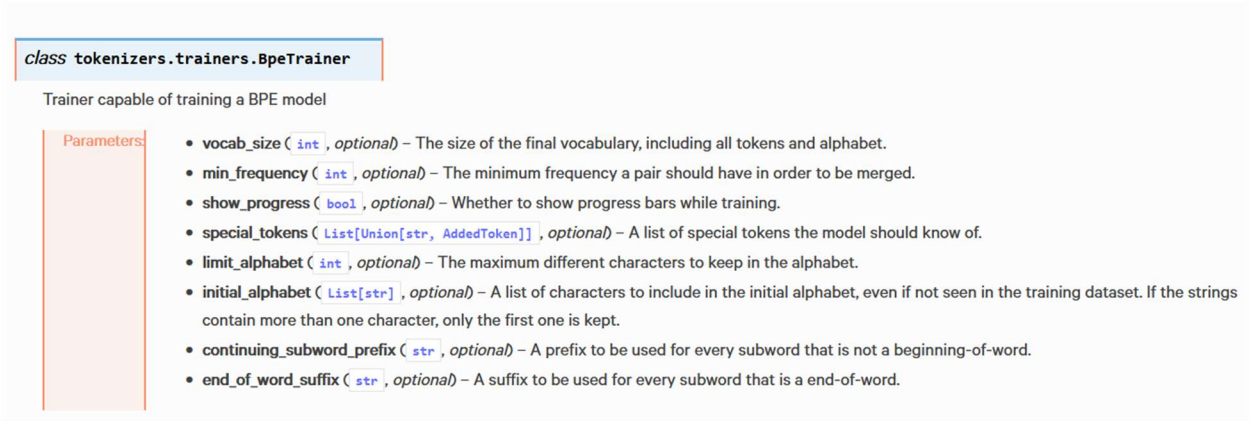
  The number of extracted tokens: 16533

- WordPiece Model

  The number of extracted tokens: 17566

Now, I should do the learner part of the two BPE & WordPiece models on the **English Wikipedia text**.

The Wikipedia dataset is big, so I used the vocab_size parameter to set the final size of vocabulary to get more tokens. (I set it to 500000. Without defining this parameter, it got me 30000 tokens). Here are some of the other parameters:



class **tokenizers.trainers.BpeTrainer**

Trainer capable of training a BPE model

Parameters:

- **vocab_size** (`int` , *optional*) – The size of the final vocabulary, including all tokens and alphabet.
- **min_frequency** (`int` , *optional*) – The minimum frequency a pair should have in order to be merged.
- **show_progress** (`bool` , *optional*) – Whether to show progress bars while training.
- **special_tokens** (`List[Union[str, AddedToken]]` , *optional*) – A list of special tokens the model should know of.
- **limit_alphabet** (`int` , *optional*) – The maximum different characters to keep in the alphabet.
- **initial_alphabet** (`List[str]` , *optional*) – A list of characters to include in the initial alphabet, even if not seen in the training dataset. If the strings contain more than one character, only the first one is kept.
- **continuing_subword_prefix** (`str` , *optional*) – A prefix to be used for every subword that is not a beginning-of-word.
- **end_of_word_suffix** (`str` , *optional*) – A suffix to be used for every subword that is a end-of-word.

The results are: (the extracted tokens were too much so I did not put it in the report).

- BPE Model

  The number of extracted tokens: 500,000

- WordPiece Model

  The number of extracted tokens: 500,000

In this part I show the segmenter part.
I apply 4 models which I learned to the **text in text-box**. Results:

- BPE Model – Gutenberg Book

  The number of extracted tokens: 54
  Encoded tokens: ['This', 'is', 'a', 'deep', 'learning', 'to', 'ken', 'ization', 't', 'ut', 'or', 'ial', '.', 'T', 'ok', 'en', 'ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'N', 'L', 'P', 'pi', 'pe', 'line', '.', 'We', 'will', 'be', 'comparing', 'the', 'to', 'k', 'ens', 'generated', 'by', 'each', 'to', 'ken', 'ization', 'model', '.', 'Ex', 'c', 'ited', 'much', '?', '!']

- WordPiece Model – Gutenberg Book

  The number of extracted tokens: 54
  Encoded tokens: ['This', 'is', 'a', 'deep', 'learning', 'to', '##ken', '##ization', 't', '##ut', '##oria', '##l', '.', 'To', '##ken', '##ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'N', '##L', '##P', 'pip', '##el', '##ine', '.', 'We', 'will', 'be', 'comparing', 'the', 'to', '##ken', '##s', 'generated', 'by', 'each', 'to', '##ken', '##ization', 'model', '.', 'Ex', '##ci', '##ted', 'much', '?', '##!', '[UNK]']

- BPE Model – English Wiki

  The number of extracted tokens: 40
  Encoded tokens: ['This', 'is', 'a', 'deep', 'learning', 'token', 'ization', 'tutorial', '.', 'Token', 'ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'NL', 'P', 'pipeline', '.', 'We', 'will', 'be', 'comparing', 'the', 'tokens', 'generated', 'by', 'each', 'token', 'ization', 'model', '.', 'Excited', 'much', '?', '!']

- WordPiece Model – English Wiki

  The number of extracted tokens: 40
  Encoded tokens: ['This', 'is', 'a', 'deep', 'learning', 'token', '##ization', 'tutorial', '.', 'Token', '##ization', 'is', 'the', 'first', 'step', 'in', 'a', 'deep', 'learning', 'NL', '##P', 'pipeline', '.', 'We', 'will', 'be', 'comparing', 'the', 'tokens', 'generated', 'by', 'each', 'token', '##ization', 'model', '.', 'Excited', 'much', '[UNK]', '[UNK]']

As we can see the two models on English wiki have less tokens (40<54) and have better results than the 2 other models which were trained on Gutenberg book because the English Wiki dataset is bigger, and the models saw more words and can tokenize words of the text better with more accuracy. Also, the tokens aremore meaningful. Also, some words were UNK word because they were not in the training data such as “NLP” and the emoji or “tokenization” or “excited”.

Step #3

I apply 4 models which I learned to the **Gutenberg book**. Results: (I did not show the Encoded tokens because they were long; they can be seen in the notebook file).

- BPE Model – Gutenberg Book

  The number of extracted tokens: 122778

- WordPiece Model – Gutenberg Book

  The number of extracted tokens: 122805

- BPE Model – English Wiki

  The number of extracted tokens: 127598

- WordPiece Model – English Wiki

  The number of extracted tokens: 124373

Trained Tokenizer on all Wikipedia dateset	Trained Tokenizer on Gutenberg Book	Algorithm	row
127598	122778	BPE	1
124373	122805	WordPiece	2

Table 1 Number of tokens on Gutenberg Book

*Pay attention that these numbers are with vocab_size parameter of 500,000 for Wikipedia dataset.

The 2 models which were trained on Gutenberg book could encode the Gutenberg book better and had a smaller number of tokens because the vocabulary was more related to the book than Wikipedia dataset and could tokenize the book better, so the unknown words are less. On WordPiece algorithm there are some [Unk] words in the encoded text but this special word ([UNK]) did not appear in the encoded taxt tokenized by BPE algorithm.

Conclusion

In this assignment I studied some famous and important tokenizers (BPE & WordPiece). I implemented the BPE algorithm from scratch. I compared these two algorithms and now I can use them based on the use of my project.

Totally, as I saw and study the results in this project when the training dataset is bigger, we can get better results with considering that our training dataset is related to the test set and also, when we have big training dataset the WordPiece algorithm gets better results than BPE algorithm in a fixed big size training dataset (based on the table in step 3). I think the WordPiece algorithm usually gets better results than BPE because it prioritizes the merges by likelihood and estimate best pair to merge not just by the frequency of the occurrence (BPE). Although both algorithms are so important and can be used in a lot of tasks and there is not an absolute preference between these two in all tasks.

*** There is an ipynb file of my notebook also there are two html files of that notebook which in one of them all the outputs, vocabularies and encoded texts are shown, in the other one these lines of codes to show outputs of Gutenberg and Wiki datasets are commented to avoid long text and keep it clear and brief. **Make sure you have the Sample.txt file and the wikitext-103-raw folder to run the code.**