Assignment #5

CSE 447: Natural Language Processing
Eric Boris: 1976637
Collaborators: Allyson Ely, Ardi Madadi

Implementation Problem

Problem 2: Answer

1. **Yes**, because if there are multiple words per line then different words will be joined rather than their letters being joined.
2. **No**, each additional iteration over the data produces a new BPE encoding.
3. See Figure 1

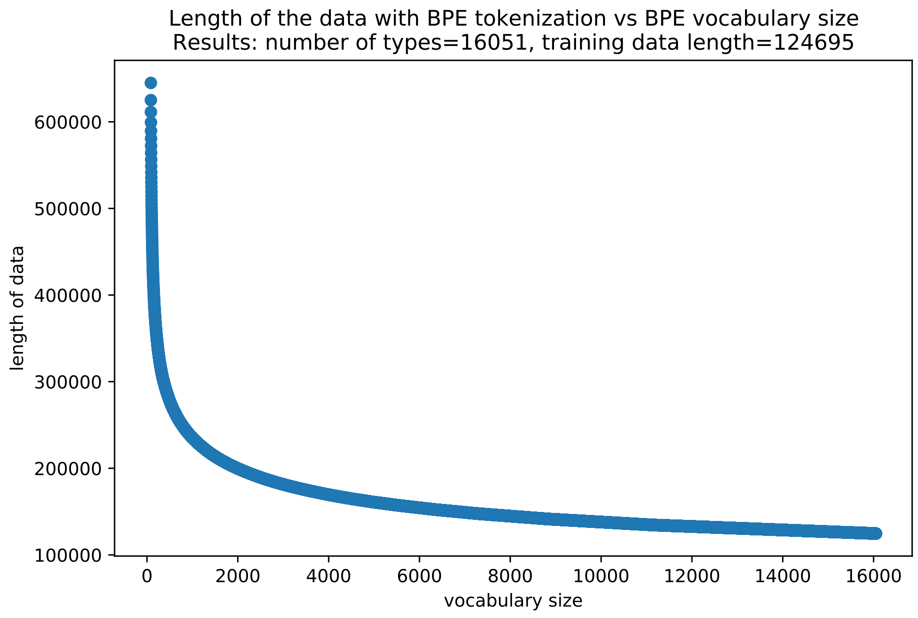

Figure 1

4. **No**, we don't get any <unk> tokens. We're unlikely to get <unk> because that will only happen if the entire word cannot be encoded by BPE which will only occur if the sequence of letters composing that word hasn't been encoded during training. Using large, diverse training datasets reduces the likelihood of this occuring.

Results of encoding 10 uncommon english words using BPE

	Word	Result
1	fudgel	f ud g el<s>
2	nudiustertian	nu di ust er tian<s>
3	selcouth	sel cou th<s>
4	zabernism	z ab ern ism<s>
5	douceur	dou ce ur<s>
6	pauciloquent	pa uc il o qu ent<s>
7	defenestration	defen estr ation<s>
8	hiraeth	h ir ae th<s>
9	limerence	li mer ence<s>
10	sonder	s ond er<s>

5. Compared to this implementation of BPE, character encoding is quite fast since there are fewer comparisons. But it won't perform as well a BPE encoding because it's too fine grained and can't recognize word structures.

Problem 2: Code

```
# CSE 447 A5 Problem 2

from collections import Counter
import matplotlib.pyplot as plt
import numpy as np

def load(path):
    '''
    Return the contents of the given file located in path.
    '''
    with open(path, 'r') as f:
        return f.read()

def save(path, contents):
    '''
    Save the contents to the file at the given path.
    '''
    with open(path, 'w') as f:
        f.write(contents)

def plot(title, subtitle, x_label, y_label, x, y, path, dim=(8, 5)):
    '''
    Plot the given data as a scatter plot.
    '''
    plt.figure(figsize=dim)
    plt.title(f'{title}\n{subtitle}')
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.plot(x, y, 'o')
    plt.savefig(path)
    plt.show()

class BPE:
    def __init__(self):
        self.end_sym = '<s>'

    def get_vocab(self, data):
        '''
        Return the unique characters appearing in the data plus the end symbol.
        '''
        vocab = set(data)
        vocab.remove(' ')
        vocab = list(vocab)
        vocab.append(self.end_sym)
        return vocab

    def tokenize(self, data):
        '''
        Return the data as a tokenized list of characters with words separated
        by the end symbol.
        '''
        tokens = []
        for word in data.split():
            for char in word:
                tokens.append(char)
            tokens.append(self.end_sym)
        return tokens

    def get_pairs(self, tokens):
        '''
        Return a mapping of token pairs -> count of token pair occurences.
        '''
        pairs = Counter()
        for i in range(len(tokens)-1):
            # Prevent merge across words.
            if not tokens[i].endswith(self.end_sym):
                pairs[tokens[i], tokens[i+1]] += 1
        return pairs

    def merge(self, tokens, best):
        '''
        Merge all the best token pairs in tokens and return the merged list.
        '''
        i = 0
        while i < len(tokens) - 1:
            if (tokens[i], tokens[i+1]) == best:
                tokens[i : i+2] = [''.join(tokens[i : i+2])]
            i += 1
        return tokens

    def train(self, data, min_freq=1, verbose=False, print_every=10):
        '''
        Train the bpe model and return the resultant vocabulary and merged
        tokens and the size of the vocab and tokens at each step.
        '''
        self.vocab = self.get_vocab(data)
        tokens = self.tokenize(data)
        pairs = self.get_pairs(tokens)

        # Let this be the tuple from pairs with the highest occurence.
        best = max(pairs, key=pairs.get)

        # Store the best transformations for applying BPE to new words
        # after training.
        self.transforms = []

        # These will be returned and used for plotting data.
        n_vocab = [len(self.vocab)]
        n_tokens = [len(tokens)]

        # Let this maintain the current iteration for displaying progress.
        i = 0

        while pairs[best] > min_freq:
            self.vocab.append(''.join(best))
            self.transforms.append(best)

            tokens = self.merge(tokens, best)
            pairs = self.get_pairs(tokens)
            best = max(pairs, key=pairs.get)

            n_vocab.append(len(self.vocab))
            n_tokens.append(len(tokens))

            if verbose and i % print_every == 0:
                print(f'{i} {pairs[best]}')
```

```python

            i += 1

        return (' ').join(tokens), self.vocab, n_tokens, n_vocab

    def encode(self, data):
        '''
        Apply the tranformations found during training to encode the given data
        into the trained BPE scheme.
        '''
        tokens = self.tokenize(data)
        for best in self.transforms:
            tokens = self.merge(tokens, best)
        return (' ').join(tokens)

if __name__ == '__main__':
    # Load the training data.
    load_path = '../data/A5-data.txt'
    data = load(load_path)

    # Build and train the model.
    bpe = BPE()
    encoded, vocab, n_tokens, n_vocab = bpe.train(data, verbose=True, print_every=100)

    # Display the results.
    print(f'encoded={encoded}\nvocab={vocab}\nn_tokens={n_tokens}\nn_vocab={n_vocab}')

    # Save the encoded output.
    save_path = '../data/output.txt'
    save(save_path, encoded)

    # Plot the effects of training on data length and vocab size.
    plot(title='Length of the data with BPE tokenization vs BPE vocabulary size',
        subtitle=f'Results: number of types={n_vocab[-1]}, training data length={n_tokens[-1]}',
        x_label='vocabulary size',
        y_label='length of data',
        x=n_vocab,
        y=n_tokens,
        path='../figures/plot.pdf')

    # Encode new words using the trained BPE model.
    new_load_path = '../data/least_common_words.txt'
    new_words = load(new_load_path)
    new_encoded = bpe.encode(new_words)

    # Display the results.
    print(new_encoded)

    # Save the new encoded output.
    new_save_path = '../data/new_output.txt'
    save(new_save_path, new_encoded)
```