Homework 4

11-967

Due date: 02/16/2025 11:59 PM EST

In this homework, you will learn how to tokenize data.

Problem 1: BPE Tokenizer

Formally, a tokenizer is a function that takes a string $s \in \mathcal{A}^\star$ in alphabet $\mathcal{A}$ to a sequence of tokens $\mathbf{x} \in \mathcal{V}^\star$ in vocabulary $\mathcal{V}$.[1] By definition, tokenizers are complete in their alphabet: *any string over $\mathcal{A}$ can be tokenized.* Tokenizers are essential in NLP pipelines because they convert natural language to symbols that machines can parse, and vice versa.

[Question 1.1] (***Writing, 12 points***) We may expect a tokenizer to satisfy the properties below:

- **Injective**: for all $s \neq s'$, $\text{tokenize}(s) \neq \text{tokenize}(s')$.
- **Invertible**: for any string s, $s = \text{detokenize}(\text{tokenize}(s))$.
- **Preserves concatenation**: for two strings s, s', $\text{tokenize}(s + s') = \text{tokenize}(s) + \text{tokenize}(s')$.

However, all three properties are violated by the tokenizers of multiple popular language models.

DELIVERABLES FOR Q1.1

Answer the following questions.

A. Provide counterexamples showing how a popular, publicly-available tokenizer fails each of the above properties. Write code in `tests/test_tokenizer.py` that causes the assertions in `test_not_injective()`, `test_not_invertible()` and `test_not_preserving_concat()` to all pass. Then, in your report, write down no more than two sentences about each test, describing the tokenizer you chose and the violation you identified.

B. Comment on why it is generally impossible to build non-trivial tokenizers that preserve concatenation.[a]

[a] A trivial tokenizer is one that uses the alphabet as vocabulary.

[Question 1.2] (***Coding, 25 points***) Byte-pair encoding (BPE) is a technique of *learning* a useful tokenizer from a text corpus. BPE first initializes the set of tokens as all characters in some alphabet, and iteratively merges the most frequent pair of adjacent tokens (bigram) as a new token. Each iteration adds a new token to the vocabulary while retaining the existing tokens.

DELIVERABLES FOR Q1.2

Make the following modifications to the starter code.

A. Implement the BPE algorithm by filling in `ASCIIBPETokenizer.merge()` in `src/tokenizer/bpe.py`. Your implementation should pass all of the `test_*_merge()` test cases in `tests/test_tokenizer.py`.

B. Now, implement `encode()` and `decode()` functions in `src/tokenizer/bpe.py`. Your implementation should pass `test_encode()` and `test_decode()`.

Additional automatic tests may be performed after submission.

[1] For example, the alphabet $\mathcal{A}$ could be the ASCII or the Unicode character set.

[Question 1.3] (***Writing, 6 points***) You have implemented an ASCII tokenizer that handles the ASCII characters. One limitation of ASCII is that it does not include non-Latin characters, for example those of Hindi or Chinese. In contrast, the Unicode standard supports chraracters from all writing systems.

The instructors have provided a unicode BPE tokenizer code implementation, `src/tokenizer/bpe.py`, and a tokenizer file, `data/english-tokenizer.json`, that has been trained only on English text.
Run `python src/tokenizer/visualize.py` to load in this tokenizer and observe the learned tokens.

DELIVERABLES FOR Q1.3

Answer the following questions with at most two sentences each:

A. What are the longest token that you see, and what does this tell you about the training data?

B. How can BPE tokenization compromise the privacy of its training data?

[Question 1.4] (***Writing, 6 points***) Try to tokenize a piece of provided English text and its translation in Thai by running `python src/tokenizer/multilingual.py`.

DELIVERABLES FOR Q1.4

Answer the following questions with at most two sentences each:

A. Is there a big difference between the number of tokens used to tokenize the same document in English and Thai? Explain why.

B. How could this phenomenon be problematic for users of low-resource languages?