Foundations of Natural Language Processing

['▁Found', 'ations', '▁of', '▁Natural', '▁Language', '▁Process', 'ing'][1]

Peking University, 2024

Lab 2: Due Sunday, April 28, 2024 by 23:59

1. Directions

PLEASE read these instructions to ensure you receive full credit on your homework.

- Submit your homework as a zip file through **Course**, which should include one report in PDF and your source code in python.
- The report should be generated from the **LaTeX template** provided in Lab 1 and you do not need to submit the data you used.
- The code should be paired with a README file describing dependencies, code structures, etc.
- There is no need to submit the data you used and the model weights. Your grade will be based on the contents of the report and the source code.

LATE SUBMISSION POLICY

- Late homework will have 5% deducted from the final grade for each day late.
- No submission will be accepted after May 5, one week after the due date. Your homework submission time will be based on the time of your last submission to Course.

Therefore, do not re-submit after midnight on the due date unless you are confident that the new submission is significantly better to overcompensate for the credits lost. You can resubmit as much as you like, but each time you resubmit be sure to upload all files you want to be graded! Submission time is non-negotiable and will be based on the time you submitted your last file to Course. The number of points deducted will be rounded to the nearest integer.

[1] LLaMA-2's tokenization result of 'Foundations of Natural Language Processing'.

2. Problem Description

Tokenization is an important step in building NLP systems. In this lab, you will build your own BPE tokenizer and explore the tokenizers of latest LLMs.

2.1 Implementation of Byte-Pair Encoding (60%)

The rare/unknown word issue is ubiquitous in neural text generation. Typically, a relatively small vocabulary is constructed from the training data, and the words not appearing in the vocabulary are replaced with a special `<unk>` symbol.

Byte-pair encoding (BPE) is currently a popular technique to deal with this issue. The idea is to encode text using a set of automatically constructed types, instead of conventional word **types**. A type can be a character or a subword unit; and the types are built through an iterative process, which we now walk you through.

Suppose we have the following tiny training data:

```
it unit unites
```

The first step is to append, to each word, a special `<s>` symbol marking the end of a word. Our initial types are the characters and the special end-of-word symbol: `{i, t, u, n, e, s, <s>}`. Using the initial type set, the encoding of the training data is

```
i t <s> u n i t <s> u n i t e s <s>
```

In each training iteration, the most frequent type bigram is merged into a new symbol and then added to the type vocabulary. Ties can be broken at random.

Iteration 1

Bigram to merge (with frequency 3): `i t`

Updated data: `it <s> u n it <s> u n it e s <s>`

Iteration 2

Bigram to merge (with frequency 2): `it <s>`

Updated data: `it<s> u n it<s> u n it e s <s>`

Iteration 3

Bigram to merge (with frequency 2): `u n`

Updated data: `it<s> un it<s> un it e s <s>`

In this example, we end up with the type vocabulary `{i, t, u, n, e, s, <s>, it, it<s>, un}`. The stopping criterion can be defined by setting a target vocabulary size.

Applying BPE to new words is done by first splitting the word into characters, and then iteratively applying the merge rules in the same order as in training. Suppose we want to encode text `unite itunes`, it is first split into `u n i t e <s> i t u n e s <s>`. Then

Iteration 1

Bigram to merge: `i t`

Encoded word: `u n it e <s> it u n e s <s>`

Iteration 2

Bigram to merge: `u n`

Encoded word: `un it e <s> it un e s <s>`

The above procedure is repeated until no merge rule can be applied. In this example we get the encoding `un it e <s> it un e s <s>`.

Q1. Implement the BPE algorithm and run it on the text data `bpe-training-data.txt` in the attachment. What we want you to produce is a scatterplot showing points *(x; y)*, each corresponding to an iteration of the algorithm, with *x* the current size of the type vocabulary, and *y* the length of the training corpus (in tokens) under that vocabulary's types. Run the algorithm until the frequency of the most frequent type bigram is one. How many types do you end up with? What is the length of the training data under your final type vocabulary?

Q2. In the above running example, neither `unite` or `itunes` appears in our training data. We nevertheless managed to encode them meaningfully using BPE types, which would otherwise be `<unk>` if we were using conventional word types. Encode the article `bpe-testing-article.txt` in the attachment using the BPE encoding you just trained. What is the length of the article under your final type

vocabulary? Do you get an <unk>? More generally, justify why it is less likely that we encounter an <unk> using the BPE encoding.

Please submit the code of training and the result after applying the BPE algorithm, and answer the above questions in the report.

2.2 Analysis of LLM Tokenizers (40%)

There are various tokenizers apart from BPE, like WordPiece, Unigram, and BBPE. Please survey the prevalent tokenizers used in Large Language Models (LLMs) and compare their advantages and disadvantages.

Possible aspects:

- How do they handle languages without space, such as Chinese?
- How do they handle out-of-vocabulary words/characters?
- How do they deal with numbers?
- Are some tokenizers more efficient than others?

You may delve into other aspects if you want. To support your argument, we highly recommend that you include **specific examples or experiments** in the report.

3. References and Useful resources

For Problem 2.1:

Problem 2.1 is adapted from Assignment 8 of CSE 447 and 517: Natural Language Processing - University of Washington, Winter 2022.

BPE has an open-source implementation: https://github.com/rsennrich/subword-nmt. You can use it to check your implementation or in your own research. Please refrain from copy-pasting code for this assignment.

For Problem 2.2:

https://huggingface.co/docs/transformers/tokenizer_summary provides a brief summary of tokenizers. It can be a great starting point of your analysis.

Here are some famous open-source LLMs you can choose from for your analysis: LLaMA, Mistral, Baichuan, Qwen, Deepseek, ChatGLM, BLOOM…

You can refer to their papers/technical reports and download their tokenizers from HuggingFace or ModelScope.