Lab1

2026 年 1 月 21 日

目录

1 基本概念

1.1 Tokenizer

1.1.1 Word-level tokenizer

将每个单词定义为一个 token，这样相较于 Byte-level tokenizer 来说，对于一个长句子，会产生更少的 token，但由于需要为每个词都定义一个序号，必然会存在某些词并不在定义的字典当中，从而无法解析

1.1.2 Byte-level tokenizer

将每一个字符定义为一个 token，其字典大小为 0-255，这样可以覆盖到每一个可见的字母，但缺点是如果一个单词的长度很长，会使得 token 值特别大，在模型训练的时候会更浪费时间

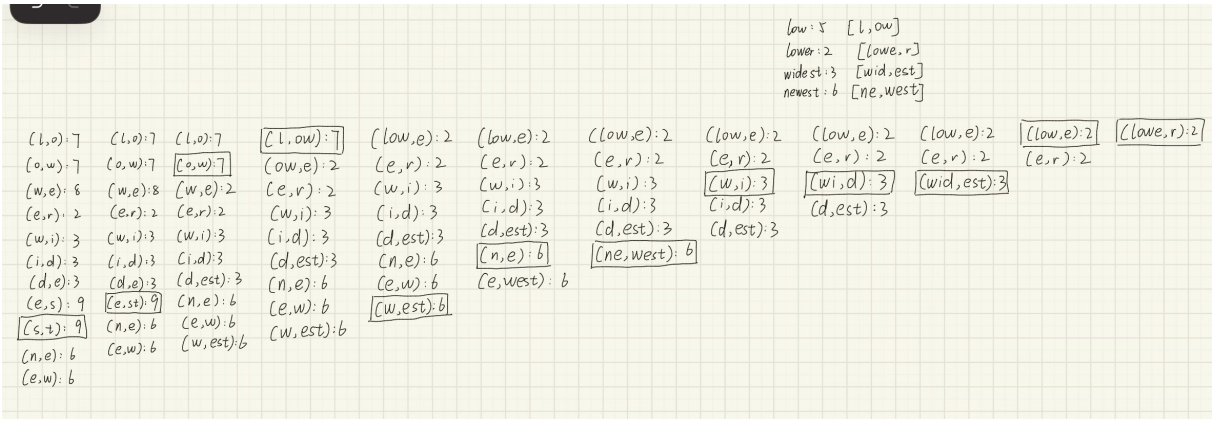
1.1.3 Subword tokenizer(BPE)

结合 Word-level tokenizer 和 Byte-level tokenizer 的优点，得到了 BPE 这种压缩算法，即 Byte-pair encoding

对于一个句子，我们的字典最初就是 Byte-level tokenizer 的字典，序号是 0-255，然后遍历整个句子，每次将相邻的两个单词组成一个 pair，然后记录该 pair 出现的次数，遍历完以后，将出现次数最高的 pair 定义为一个新的字典序号，并加入到字典当中，然后将该 pair 看成是一个 word，持续遍历整个句子，直到合并完成。直接做的缺点有每次我们都需要完整遍历语料库，计算量非常大，其次，如果是相邻的标点符号不一样，也会生成一个新的 ID，尽管标点前面的单词是一样的

在最初提出的 BPE 中，字典里存在一些 `Special tokens`，比如 `<|endoftext|>`，其用途是告诉程序什么时候停止生成 token，`special tokens` 不会被分割成多个 token，在该实验的 BPE 中不考虑 special token

给定 `low`、`lower`、`widest`、`newest`，根据 BPE 的步骤进行分词



下面是一个Token count网页，可以在网页内选择不同的模型，看相同的一个句子其生成的 token 的方式的不同

2 问题部分

2.1 Problem unicode1

Problem (unicode1): Understanding Unicode (1 point)

(a) What Unicode character does `chr(0)` return?

Deliverable: A one-sentence response.

(b) How does this character's string representation (`__repr__()`) differ from its printed representation?

Deliverable: A one-sentence response.

(c) What happens when this character occurs in text? It may be helpful to play around with the following in your Python interpreter and see if it matches your expectations:

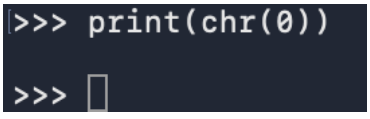
```
>>> chr(0)
>>> print(chr(0))
>>> "this is a test" + chr(0) + "string"
>>> print("this is a test" + chr(0) + "string")
```

Deliverable: A one-sentence response.

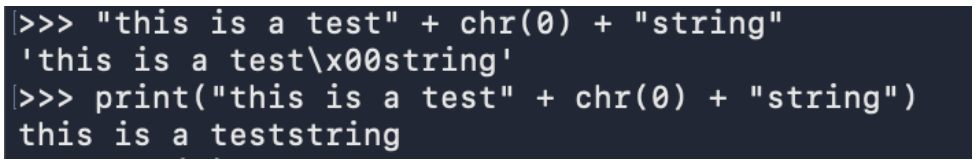
2.1.1 Answer

(a) 在终端中使用 python 环境进行实现

(b) `chr(0)` 是空字符，直接在命令行输入命令或调用 `__repr__()` 时会用转义字符来表示，对于非打印字符来说直接打印是不可见的



(c) `print()` 是位于用户视图，直接在交互式命令行输入命令或调用 `__repr__()` 是位于开发者视图，因此一个可见空字符，一个不可见空字符



2.2 Problem unicode2

Problem (unicode2): Unicode Encodings (3 points)

(a) What are some reasons to prefer training our tokenizer on UTF-8 encoded bytes, rather than UTF-16 or UTF-32? It may be helpful to compare the output of these encodings for various input strings.

Deliverable: A one-to-two sentence response.

(b) Consider the following (incorrect) function, which is intended to decode a UTF-8 byte string into a Unicode string. Why is this function incorrect? Provide an example of an input byte string that yields incorrect results.

```
def decode_utf8_bytes_to_str_wrong(bytestring: bytes):
    return "".join([bytes([b]).decode("utf-8") for b in bytestring])

>>> decode_utf8_bytes_to_str_wrong("hello".encode("utf-8"))
'hello'
```

Deliverable: An example input byte string for which `decode_utf8_bytes_to_str_wrong` produces incorrect output, with a one-sentence explanation of why the function is incorrect.

(c) Give a two byte sequence that does not decode to any Unicode character(s).

Deliverable: An example, with a one-sentence explanation.

2.2.1 Answer

(a) UTF-8 相较于 UTF-16 和 UTF-32 有以下几个优点：

- UTF-8 完全兼容 ASCII 码，而 UTF-16 和 UTF-32 处理 ASCII 字符时会插入大量空字节
- UTF-8 是变长编码，通常是 1 到 4 个字节不等，对比 UTF-16 和 UTF-32 更节省空间
- UTF-8 是以单字节为单位处理的，不存在大端序和小端序转换的问题，而 UTF-16 和 UTF-32 必须处理字节序问题，否则会出现乱码
- 如果文件在传输过程中丢失了一个字节或受损，UTF-8 能根据后续的字节特征快速找到下一个字符的起始位置

(b) 这段代码会出错，因为在函数当中处理 bytestring 的方式是一个字节一个字节的解码，但 UTF-8 是变长编码，如果输入的是多个字节组成的字符会导致 **UnicodeDecodeError**

```
def decode_utf8_bytes_to_str_wrong(bytestring: bytes):
    return "".join([bytes([b]).decode("utf-8") for b in bytestring])

if __name__ == "__main__" :
    print(decode_utf8_bytes_to_str_wrong("hello,你好".encode("utf-8")))
```

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe4 in position 0: unexpected end of data
```

(c) `0XC0 0XAF`，在现代解码器中任何以 `0XC0`、`0XC1` 开头的字节序列都会被抛出 **UnicodeDecodeError**

2.3 Problem train_bpe

Problem (train_bpe): BPE Tokenizer Training (15 points)

Deliverable: Write a function that, given a path to an input text file, trains a (byte-level) BPE tokenizer. Your BPE training function should handle (at least) the following input parameters:

`input_path: str` Path to a text file with BPE tokenizer training data.

`vocab_size: int` A positive integer that defines the maximum final vocabulary size (including the initial byte vocabulary, vocabulary items produced from merging, and any special tokens).

`special_tokens: list[str]` A list of strings to add to the vocabulary. These special tokens do not otherwise affect BPE training.

Your BPE training function should return the resulting vocabulary and merges:

`vocab: dict[int, bytes]` The tokenizer vocabulary, a mapping from `int` (token ID in the vocabulary) to `bytes` (token bytes).

`merges: list[tuple[bytes, bytes]]` A list of BPE merges produced from training. Each list item is a `tuple` of `bytes` (`<token1>`, `<token2>`), representing that `<token1>` was merged with `<token2>`. The merges should be ordered by order of creation.

To test your BPE training function against our provided tests, you will first need to implement the test adapter at `[adapters.run_train_bpe]`. Then, run `uv run pytest tests/test_train_bpe.py`. Your implementation should be able to pass all tests. Optionally (this could be a large time-investment), you can implement the key parts of your training method using some systems language, for instance C++ (consider `cppyy` for this) or Rust (using PyO3). If you do this, be aware of which operations require copying vs reading directly from Python memory, and make sure to leave build instructions, or make sure it builds using only `pyproject.toml`. Also note that the GPT-2 regex is not well-supported in most regex engines and will be too slow in most that do. We have verified that Oniguruma is reasonably fast and supports negative lookahead, but the `regex` package in Python is, if anything, even faster.

2.3.1 Answer

根据问题内容，我们首先应该在**./tests/adapters.py** 完善其 **run_train_bpe** 函数，打开文件后，我们不难发现整个函数只有一句

```
raise NotImplementedError
```

其主要目的就是告诉我们的测试程序应该去哪里找到我们自己编写的函数，以及传入和接收的一些参数值

```
from cs336_basics.train_bpe import train_bpe
return train_bpe(
    input_path = input_path,
    vocab_size = vocab_size,
    special_tokens = special_tokens
)
```

首先**./cs336_basics/train_bpe.py** 为无任何优化的原始版本，代码中有较为详细的注释，执行以下命令测试代码，警告无需在意

```
uv run pytest tests/test_train_bpe.py
```

```
============================= test session starts ==============================
platform darwin -- Python 3.13.3, pytest-8.4.1, pluggy-1.6.0
rootdir: /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics
configfile: pyproject.toml
plugins: jaxtyping-0.3.2
collected 3 items

tests/test_train_bpe.py::test_train_bpe_speed PASSED
tests/test_train_bpe.py::test_train_bpe PASSED
tests/test_train_bpe.py::test_train_bpe_special_tokens PASSED

============================== warnings summary ===============================
tests/adapters.py:303
  /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics/tests/adapters.py:303: SyntaxWarning: invalid escape sequence '\T'
    rope_theta (float): The RoPE $\Theta$ parameter.

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
======================== 3 passed, 1 warning in 5.53s =========================
```

接下来对无任何优化的分词器根据文档中的提示进行优化，代码见**./cs336_basics/train_bpe_ parallel.py**，详细注释见代码，需要注意的是我们在并行运算的代码中需要将文本进行切块，就需要用到**./cs336_basics/pretokenization_example.py** 中的函数，我们需要先给该代码添加一个 main 函数，否则运行时会报错

```
## Usage
if __name__ == "__main__" :
    with open(..., "rb") as f:
        num_processes = 4
        boundaries = find_chunk_boundaries(f, num_processes, b"<|endoftext|>")
```

接着修改好 **adapters.py** 中调用的函数后，依然是执行

```
uv run pytest tests/test_train_bpe.py
```

不难发现，速度的改变还是非常显著的

```
============================= test session starts ==============================
platform darwin -- Python 3.13.3, pytest-8.4.1, pluggy-1.6.0
rootdir: /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics
configfile: pyproject.toml
plugins: jaxtyping-0.3.2
collected 3 items

tests/test_train_bpe.py::test_train_bpe_speed PASSED
tests/test_train_bpe.py::test_train_bpe PASSED
tests/test_train_bpe.py::test_train_bpe_special_tokens PASSED

============================== 3 passed in 0.63s ==============================
```

2.4 Problem train_bpe_tinystories

Problem (train_bpe_tinystories): BPE Training on TinyStories (2 points)

(a) Train a byte-level BPE tokenizer on the TinyStories dataset, using a maximum vocabulary size of 10,000. Make sure to add the TinyStories `<|endoftext|>` special token to the vocabulary. Serialize the resulting vocabulary and merges to disk for further inspection. How many hours and memory did training take? What is the longest token in the vocabulary? Does it make sense?

Resource requirements: ≤ 30 minutes (no GPUs), ≤ 30GB RAM

Hint You should be able to get under 2 minutes for BPE training using `multiprocessing` during pretokenization and the following two facts:

(a) The `<|endoftext|>` token delimits documents in the data files.

(b) The `<|endoftext|>` token is handled as a special case before the BPE merges are applied.

Deliverable: A one-to-two sentence response.

(b) Profile your code. What part of the tokenizer training process takes the most time?

Deliverable: A one-to-two sentence response.

2.4.1 Answer

- 首先利用原生的 BPE 分词器进行实验，即不加任何优化和多线程等方式，然后在终端利用

```
uv run python -m cProfile -o workspace/tinystories.prof cs336_basics/
    train_bpe_tinystories.py
```

得到如下结果

```
Saved vocab -> workspace/tinystories_bpe_vocab.pkl
Saved merges -> workspace/tinystores_bpe_merges.pkl
Elapsed: 24.54s (0.41 min, 0.0068 hr)
RSS (approx): 0.05 GB  (install psutil for this number)
Longest token id=8857, bytes_len=16
Longest token (decoded): ' congratulations'
```

在删减版训练集上用时 24.54s，占用 0.05GB 内存，最长的 token 是’ congratulations’，有前导空格

- 在终端利用

```
uv run python - <<'EOF'
import pstats
p = pstats.Stats("workspace/tinystories.prof")
p.sort_stats("tottime").print_stats(30)
EOF
```

得到如下结果

```
Fri Jan 30 10:46:39 2026    workspace/tinystories.prof

         573154956 function calls (573153700 primitive calls) in 73.119 seconds

   Ordered by: internal time
   List reduced from 1089 to 30 due to restriction <30>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1   45.980   45.980   73.090   73.090 /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics/cs336_basics/train_bpe.py:14(train_bpe)
168610077    9.277    0.000    9.277    0.000 {method 'get' of 'dict' objects}
206005225/206005100    6.441    0.000    6.441    0.000 {built-in method builtins.len}
135784757    5.936    0.000    5.936    0.000 {method 'append' of 'list' objects}
15236/15235    3.074    0.000    4.935    0.000 {built-in method builtins.max}
 53695350    1.861    0.000    1.861    0.000 /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics/cs336_basics/train_bpe.py:91(<lambda>)
  6422070    0.352    0.000    0.352    0.000 /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics/cs336_basics/train_bpe.py:66(<genexpr>)
  1263131    0.078    0.000    0.078    0.000 {method 'group' of '_regex.Match' objects}
  1263142    0.077    0.000    0.077    0.000 {method 'encode' of 'str' objects}
      176    0.003    0.000    0.003    0.000 {method 'split' of 'str' objects}
       72    0.003    0.000    0.003    0.000 {built-in method marshal.loads}
     6458    0.003    0.000    0.003    0.000 {method 'finditer' of '_regex.Pattern' objects}
        3    0.003    0.001    0.003    0.001 {built-in method _imp.create_dynamic}
    29479    0.002    0.000    0.002    0.000 {method 'items' of 'dict' objects}
       72    0.001    0.000    0.001    0.000 {built-in method _io.open_code}
       72    0.001    0.000    0.005    0.000 <frozen importlib._bootstrap_external>:782(_compile_bytecode)
  273/272    0.001    0.000    0.003    0.000 {built-in method builtins.__build_class__}
        2    0.001    0.001    0.001    0.001 {built-in method _codecs.utf_8_decode}
      408    0.001    0.000    0.001    0.000 {built-in method posix.stat}
        2    0.001    0.001    0.001    0.001 {built-in method _pickle.dump}
        1    0.001    0.001    0.003    0.003 /Users/zhangjinqi/Desktop/acm/learn/CS336/assignment1-basics/.venv/lib/python3.13/site-packages/regex/_regex_c
ore.py:1(<module>)
        2    0.001    0.000    0.002    0.001 {method 'read' of '_io.TextIOWrapper' objects}
       72    0.001    0.000    0.001    0.000 {method 'read' of '_io.BufferedReader' objects}
       34    0.001    0.000    0.001    0.000 {built-in method builtins.eval}
    10000    0.001    0.000    0.001    0.000 cs336_basics/train_bpe_tinystories.py:28(<lambda>)
      197    0.001    0.000    0.003    0.000 <frozen importlib._bootstrap_external>:1624(find_spec)
    78/30    0.000    0.000    0.001    0.000 /Users/zhangjinqi/.local/share/uv/python/cpython-3.13.3-macos-aarch64-none/lib/python3.13/re/_parser.py:511(_p
arse)
       34    0.000    0.000    0.001    0.000 /Users/zhangjinqi/.local/share/uv/python/cpython-3.13.3-macos-aarch64-none/lib/python3.13/collections/__init__
.py:358(namedtuple)
       18    0.000    0.000    0.000    0.000 {built-in method posix.listdir}
     1059    0.000    0.000    0.001    0.000 <frozen importlib._bootstrap_external>:131(_path_join)
```

耗时最多的首先是 train_bpe，因为没有添加任何优化和多线程，意料之内，其次是 dict 类型查找参数耗时比较长

行内公式示例：$E = mc^2$。

3 Latex 代码模板

在笔记中插入代码是非常常见的需求，特别是 CS 专业的笔记。

定义 3.1: 梯度下降

梯度下降（Gradient Descent）是一种用于优化函数的迭代算法。为了找到函数的局部最小值，我们需要向当前点梯度的**反方向**迈步。

定理 3.1: 泰勒公式

如果函数 $f(x)$ 在点 x_0 处具有 n 阶导数，则有：

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \tag{1}$$

```python
def hello_world():
    print("Hello, LaTeX!")
    return True
```

Listing 1: Python 示例代码

4 重点总结

注意

考试重点：请务必记住上述定理的适用条件，不要混淆！