CS336 Assignment 1 (basics): Building a Transformer LM

Answer Sheet

NewSunH

1 Assignment Overview

(nothing to solve)

2 Byte-Pair Encoding (BPE) Tokenizer

2.1 The Unicode Standard

Problem (`unicode1`): Understanding Unicode (1 point)

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

answer: `'\x00'`, or Unicode character `U+0000`, represents NUL (the null character).

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

answer: `'\x00'` in string representation, and invisible when using `print()`.

(c) What happens when this character occurs in text?

answer: Adding the NUL character to a string doesn't change what it looks like (in the most satuation). But there is actually a invisible character adding to the storage. When using UTF-8 typically, a zero byte (`0x00`) was added in the place of it.

2.2 Unicode Encodings

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.

answer: UTF-8 is byte-based, and ASCII compatible. UTF-8 is much more effient than UTF-16/32, because UTF-16/32 using zeros to fill in the blanks, which will also influence tokens.

(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.

answer: It literally decode each byte. UTF-8 character might be multiple bytes, e.g. Chinese characters and emoji. e.g. `' 你好，世界'`.

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

answer: e.g. `0x8080`, which is invalid because continuation bytes cannot appear without a valid leading byte.

2.3 Subword Tokenization

2.4 BPE Tokenizer Training

(nothing to solve)

2.5 Experimenting with BPE Tokenizer Training

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

implemented in `/cs336_basics/bpe.py`

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?
answer: It takes 117.98s, 10GB RAM in total. The longest token is `Ġaccomplishment`, which is 15 bytes long and its id is `7160`.

(b) Profle your code. What part of the tokenizer training process takes the most time?
answer: In this dataset, pretokenize takes 1:19s, over 2/3 of total time was taken.

Problem (`train_bpe_expts_owt`): BPE Training on OpenWebText (2 points)

(untrained, that's too demanding.)

2.6 BPE Tokenizer: Encoding and Decoding

Problem (`tokenizer`): Implementing the tokenizer (15 points)

implemented in `/cs336_basics/tokenizer.py`

2.7 Experiments

Problem (`tokenizer_experiments`): Experiments with tokenizers (4 points)

(a) Sample 10 documents from TinyStories and OpenWebText. Using your previously-trained TinyStories and OpenWebText tokenizers (10K and 32K vocabulary size, respectively), encode these sampled documents into integer IDs. What is each tokenizer' s compression ratio (bytes/token)?
unfinished

(b)

3 Transformer Language Model Architecture

3.1 Transformer LM

3.2 Token Embeddings

3.3 Pre-norm Transformer Block

3.4 Output Normalization and Embedding

3.5 Remark: Batching, Einsum and Efficient Computation

(nothing to solve)

3.6 Basic Building Blocks: Linear and Embedding Modules

Problem (`linear`): Implementing the linear module (1 point)

implemented in `/cs336_basics/linear.py`