

# 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 saturation). 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 efficient than UTF-16/32, because UTF-16/32 use 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 decodes 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 ‘accomplishment’, which is 15 bytes long and its id is 7160.

- (b) Profile 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)

- (a) Train a byte-level BPE tokenizer on the OpenWebText dataset, using a maximum vocabulary size of 32,000. Serialize the resulting vocabulary and merges to disk for further inspection. What is the longest token in the vocabulary? Does it make sense?

**answer:** It takes 4810.98s (1h 20m 10.98s). The longest token is ‘aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa’ (strange, I can’t tell what it is) and ‘-----’ (64 dashes). It does make sense because I found them in the owt.

- (b) Compare and contrast the tokenizer that you get training on TinyStories versus OpenWebText.  
bytes/token (lower is better compression)

TS tok on TS: 3.979507

TS tok on OWT: 3.159600

OWT tok on TS: 3.890763

OWT tok on OWT: 4.344127

### 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 Output Normalization and Embedding

### 3.3 Remark: Batching, Einsum and Efficient Computation

(nothing to solve)

### 3.4 Basic Building Blocks: Linear and Embedding Modules

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

implemented in /cs336\_basics/linear.py

**Problem (embedding): Implement the embedding module (1 point)**

implemented in /cs336\_basics/embedding.py

### 3.5 Pre-Norm Transformer Block

**Problem (rmsnorm): Root Mean Square Layer Normalization (1 point)**

implemented in /cs336\_basics/normalization.py

**Problem (positionwise\_feedforward): Implement the position-wise feed-forward network (2 points)**

implemented in /cs336\_basics/positionwise\_feedforward.py

**Problem (rope): Implement RoPE (2 points)**

implemented in /cs336\_basics/rope.py

**Problem (softmax): Implement softmax (1 points)**

implemented in /cs336\_basics/attention.py

**Problem (scaled\_dot\_product\_attention): Implement scaled dot-product attention (5 points)**

implemented in /cs336\_basics/attention.py

**Problem (multihead\_self\_attention): Implement causal multi-head self-attention**

implemented in /cs336\_basics/attention.py

### 3.6 The Full Transformer LM

**Problem (transformer\_block): Implement the Transformer block (3 points)**

implemented in /cs336\_basics/transformer.py

**Problem (transformer\_lm): Implement the Transformer block (3 points)**

implemented in /cs336\_basics/transformer.py