StoryWeaverGPT

Dataset, Tokenizer and Embedding

Group1

December 2, 2024

1. Introduction
2. Dataset
3. Tokenizer
4. Embedding

Dataset and Tokenizer

- Dataset: Reddit WritingPrompts Dataset
- Tokenizer: BPE Tokenizer

Dataset: WritingPrompts Dataset

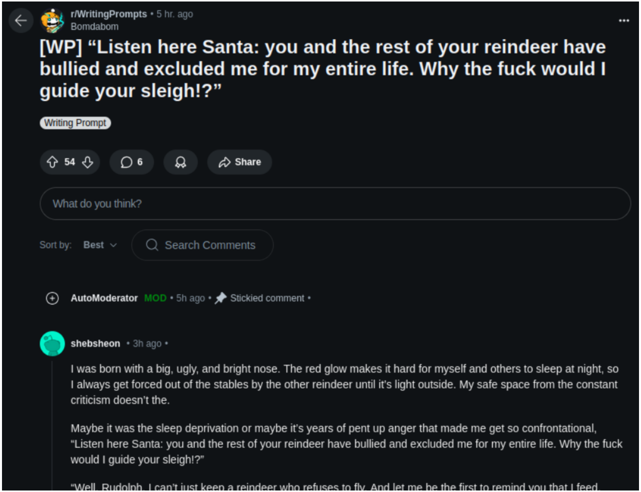


Figure: WritingPrompts Dataset

Tokenizer: BPE Tokenizer

- Iteratively merge the most frequent pairs of tokens to build a subword vocabulary.

Iteration	Tokens	Most Frequent Pair
Initial	h e l l o	l l
1	h e ll o	e ll
2	h ell o	h ell
3	hell o	None

Table: Example of BPE Tokenization for "hello"

- Key Idea: The sequence evolves by merging pairs to minimize the overall vocabulary size while retaining meaning.

Tokenizer: BPE Tokenizer

- BPE Tokenizer performs better when trained on same dataset as dataset the main model is being trained on
- Our tokenizers are trained on the WritingPrompts dataset, with 8192 merges.
- With multiprocessing training took 28 hours.

Tokenizer Code

```python
class BytePairTokenizer:
    def __init__(self, data_path:str=None) -> None:
        """
        BytePairTokenizer object
        """
        if data_path:
            self.load_model(data_path)
            return

        self.special_tokens:Dict[str, int] = {
            '<BOT>': 0,  # Beginning of Text
            '<EOT>': 1,   # End of Text
            '</w>': 2     # end of word
        }
        self.inv_special_tokens:Dict[int, str] = {i: t for t, i in self.special_tokens.items()}

        self.token_map: Dict[str, int] = self.special_tokens.copy()
        self.inv_map: Dict[int, str] = self.inv_special_tokens.copy()
        self.bpe_codes: Dict[Tuple[str, str], int] = {}
```

Figure: Initializing Tokenizer

Tokenizer Code

```python
def train(self, corpus: List[str], num_merges: int, verbose:bool = False) -> None:
    """
    Train the Byte Pair Tokenizer to process sentences.
    """
    # Build the vocabulary: map token sequences to their frequencies
    vocab = {}
    if verbose:
        print("Building vocabulary...")
    for sentence in tqdm(corpus):
        # Split sentence into words with leading whitespace preserved
        words = re.findall(r'\s*\S+|\s+', sentence)
        for word in words:
            # Skip special tokens
            if word in self.special_tokens.keys():
                continue
            chars = list(word) + ['</w>']
            word_tuple = tuple(chars)
            vocab[word_tuple] = vocab.get(word_tuple, 0) + 1

    if verbose:
        print("Vocabulary built.\nTraining BPE...")
    token_id = len(self.token_map)  # Starting token ID
    symbols = set()
    for word_tuple in vocab.keys():
        symbols.update(word_tuple)
    for symbol in symbols:
        if symbol not in self.token_map:
            self.token_map[symbol] = token_id
            token_id += 1
    self.inv_map = {i: t for t, i in self.token_map.items()}
```

Figure: Counting Frequency

Tokenizer Code

```python
if verbose:
    print("Token map built.\nMerging tokens...")
# Perform BPE merges
for i in tqdm(range(num_merges)):
    pairs = self._get_pair_counts(vocab)
    if not pairs:
        break
    best_pair = max(pairs, key=pairs.get)
    vocab = self._merge_vocab(best_pair, vocab)
    self.bpe_codes[best_pair] = i # Record the BPE merge rule
    new_symbol = ''.join(best_pair)
    if new_symbol not in self.token_map:
        self.token_map[new_symbol] = token_id
        token_id += 1
        self.inv_map[self.token_map[new_symbol]] = new_symbol
```

Figure: Performing Merges

Tokenizer Code

```python
def _get_pair_counts(self, vocab: Dict[Tuple[str], int]) -> Dict[Tuple[str, str], int]:
    """
    Get counts of symbol pairs in the vocabulary
    """
    pairs = {}
    for word, freq in vocab.items():
        symbols = word
        for i in range(len(symbols) - 1):
            pair = (symbols[i], symbols[i + 1])
            pairs[pair] = pairs.get(pair, 0) + freq
    return pairs
```

Figure: get_pair_count

Tokenizer Code

```python
def _merge_vocab_single(self, pair: Tuple[str, str], vocab: Dict[Tuple[str], int]) -> Dict[Tuple[str], int]:
    """
    Merge all occurrences of the given pair in the vocabulary
    """
    new_vocab = {}
    bigram = ''.join(pair)
    for word, freq in vocab.items():
        w = []
        i = 0
        while i < len(word):
            # Merge the pair if found
            if i < len(word) - 1 and word[i] == pair[0] and word[i + 1] == pair[1]:
                w.append(bigram)
                i += 2
            else:
                w.append(word[i])
                i += 1
        new_vocab[tuple(w)] = freq
    return new_vocab
```

Figure: merge_pair

Tokenizer Code

```python
@staticmethod
def _process_word(args):
    pair, word_freq = args
    word, freq = word_freq
    bigram = ''.join(pair)
    w = []
    i = 0
    while i < len(word):
        if i < len(word) - 1 and word[i] == pair[0] and word[i + 1] == pair[1]:
            w.append(bigram)
            i += 2
        else:
            w.append(word[i])
            i += 1
    return tuple(w), freq

def _merge_vocab(self, pair: Tuple[str, str], vocab: Dict[Tuple[str], int]) -> Dict[Tuple[str], int]:
    """
    Parallel merge of all occurrences of the given pair in the vocabulary using multiprocessing.
    """
    with Pool() as pool:
        results = pool.map(self._process_word, [(pair, word_freq) for word_freq in vocab.items()])

    new_vocab = {word: freq for word, freq in results}
    return new_vocab
```

Figure: merge_pair

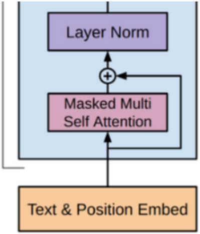
Tokenizer Code


```
{
    "token_map": {
        "<BOT>": 0,
        "<EOT>": 1,
        "</w>": 2,
        "l": 3,
        "w": 4,
        "t": 5,
        "o": 6,
        "d": 7,
        "r": 8,
        "i": 9,
        "n": 10,
        "e": 11,
        " ": 12,
        "s": 13,
        " l": 14,
        " lo": 15,
        " low": 16,
        "es": 17,
        "est": 18,
        "est</w>": 19,
        " low</w>": 20,
        " lowe": 21,
        " lower": 22,
        " lower</w>": 23
    },
    "bpe_codes": {
        "[\" \", \"l\"]": 0,
        "[\" l\", \"o\"]": 1,
        "[\" lo\", \"w\"]": 2,
        "[\"e\", \"s\"]": 3,
        "[\"es\", \"t\"]": 4,
        "[\"est\", \"</w>\"]": 5,
        "[\" low\", \"</w>\"]": 6,
        "[\" low\", \"e\"]": 7,
        "[\" lowe\", \"r\"]": 8,
        "[\" lower\", \"</w>\"]": 9
    }
}
```


Figure: Sample model

Embedding

- Embedding is a linear map from a one-hot encoded vector to a dense vector.
- $E : \mathbb{R}^V \to \mathbb{R}^d$
- Where V is vocab dimension, and d is the embedding dimension.
- This is first linear map/layer applied to tokenized input, and is learned with the model.



Embedding

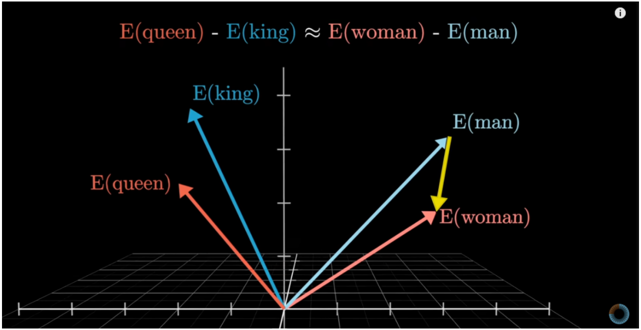


Figure: Demonstration of Embedding space

Embedding

```python
from openai import OpenAI
from dotenv import load_dotenv

load_dotenv()
client = OpenAI()

def get_embedding(text, model="text-embedding-3-small"):
   text = text.replace("\n", " ")
   return client.embeddings.create(input = [text], model=model).data[0].embedding

def cosine_similarity(embedding1, embedding2):
    embedding1 = np.array(embedding1)
    embedding2 = np.array(embedding2)
    return np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.linalg.norm(embedding2))

def main():
    model = "text-embedding-3-small"

    e_queen = get_embedding("Queen", model)
    e_king = get_embedding("King", model)
    e_woman = get_embedding("Woman", model)
    e_man = get_embedding("Man", model)

    e_comb = np.subtract(np.add(e_king, e_woman), e_man)
    print(cosine_similarity(e_queen, e_comb))

    e_hitler = get_embedding("Hitler", model)
    e_italy = get_embedding("Italy", model)
    e_germany = get_embedding("Germany", model)
    e_mussolini = get_embedding("Mussolini", model)

    e_comb = np.subtract(np.add(e_italy, e_hitler), e_germany)
    print(cosine_similarity(e_mussolini, e_comb))

    foo = "Hello"
    bar = "Jane Doe"
    e_foo = get_embedding(foo, model)
    e_bar = get_embedding(bar, model)
    print(cosine_similarity(e_foo, e_bar))

if __name__ == "__main__":
    main()
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE COMMENTS

```
(venv) nemit@nemit-ThinkBook-14-G6-IRL:~/Documents/mlgroup1/test$ python embedding_demo.py
0.726366032276112
0.6189826089726571
0.26895878577119053
(venv) nemit@nemit-ThinkBook-14-G6-IRL:~/Documents/mlgroup1/test$
```

Live Share Ln 19, Col 12 Spaces: 4

Embedding

- Outside NLP/NLG model, embeddings are used for similarity, clustering, and visualization.
- RAG, Retrival Augmented generation, uses embeddings to retrive relevant information.

Positional Encoding

- Positional encoding is added to embeddings to give the model information about the position of the token.
- Positional encoding is a sine and cosine function of the position.
- $$P[i,j] = \begin{cases} \sin\left(\frac{i}{10000^{2j/d}}\right) & \text{if j is even} \\ \cos\left(\frac{i}{10000^{2j/d}}\right) & \text{if j is odd} \end{cases}$$

Embedding and Positional Encoding Code

```python
class Embedding:
    def __init__(self, input_dim: int, output_dim: int) -> None:
        self.input_dim = input_dim
        self.output_dim = output_dim
        self.weights: Tensor = torch.randn(input_dim, output_dim) * 0.01
        self.grad_weights: Tensor = torch.zeros_like(self.weights)

    def forward(self, input_indices: Tensor) -> Tensor:
        self.input_indices = input_indices
        self.output = self.weights[input_indices]
        return self.output
```

Figure: Embedding Class

```python
class PositionalEncoding:
    def __init__(self, max_seq_len: int, embed_size: int):
        self.embed_size = embed_size
        self.pos_encoding = torch.zeros(max_seq_len, embed_size)

        position = torch.arange(0, max_seq_len, dtype=torch.float).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, embed_size, 2) * (-torch.log(torch.tensor(10000.0)) / embed_size))
        self.pos_encoding[:, 0::2] = torch.sin(position * div_term)
        self.pos_encoding[:, 1::2] = torch.cos(position * div_term)

    def forward(self, x: Tensor) -> Tensor:
        seq_length, embed_size = x.shape
        pos_encoding = self.pos_encoding[:seq_length, :]  # Slice for the current sequence length
        return x + pos_encoding.to(x.device)
```

Figure: Positional Encoding