What does an LLM do?

Given a sequence of tokens, an LLM predicts the next token in the sequence. The LLM is trained on a large corpus of text data and learns the probability distribution of the next token given the previous tokens in the sequence. This probability distribution is used to generate text by sampling from it. The token with highest probability is chosen as the next token in the sequence.

Byte Pair Tokenization

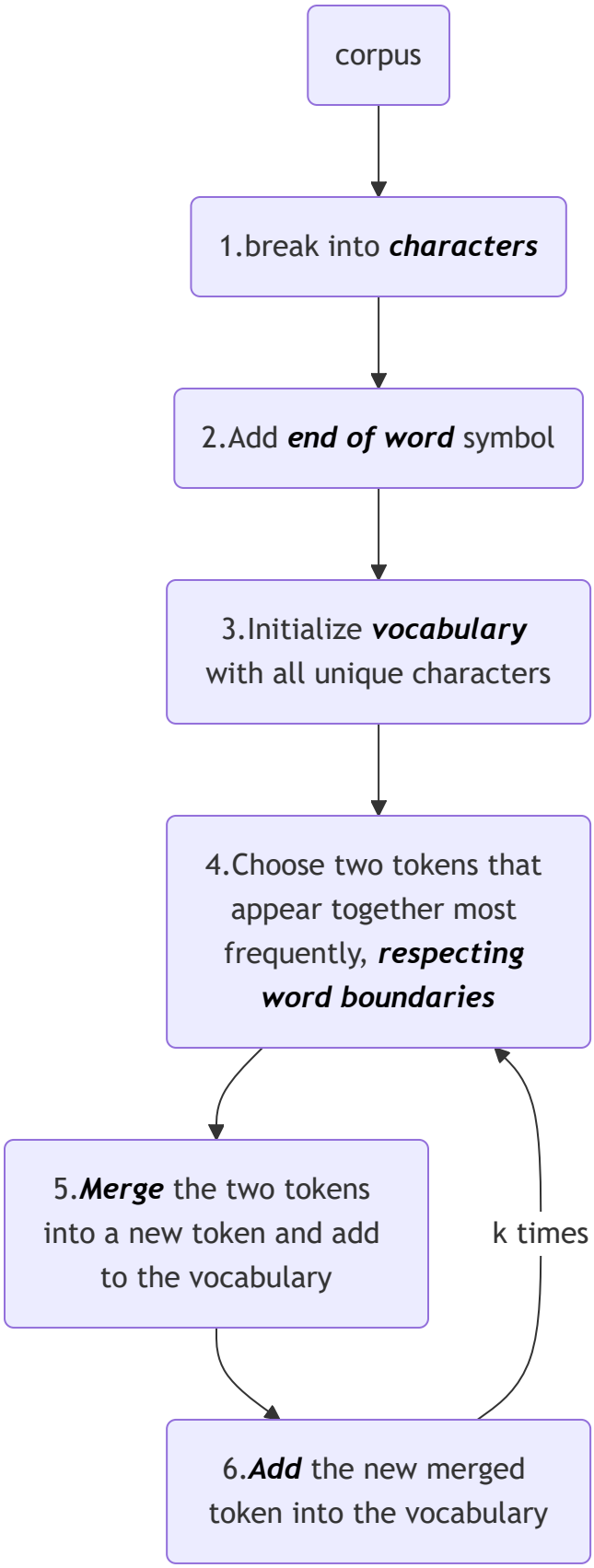
History

The 2016 BPE paper by Sennrich et al. introduced the concept of Byte Pair Encoding (BPE) for tokenization. BPE is a data compression algorithm that replaces the most frequent pair of bytes in a data stream with a single byte. This process is repeated iteratively until a predefined number of iterations or until a predefined vocabulary size is reached. The resulting vocabulary consists of the most frequent tokens in the corpus, which can be used for tokenization.

Algorithm

The tokenization algorithms like BPE (Byte Pair Encoding) identify sub-words which are most prevalent in the corpus. This way most commonly available tokens (units of text) are identified that is extent in the corpus data with which the model is trained. And those are considered separate tokens. Tokens are merged and new bigger tokens are created accordingly based on what is most commonly available.

BPE is a method to build vocabulary by generated tokens from the corpus.
This is used in GPT-2, GPT-4, Llama2, BERT, etc.



Finer points

1. white space symbol is honored because "esteem" and "dearest" are different words. So, <est\w> is important.

2. vocabulary has characters and sub-words, honoring the word boundaries. start with characters and keep augmenting it with subwords and words.
3. what should be k - when to stop - how many iterations:
    i. number of iterations
    ii. vocab size limit
4. select our data set - more data set, better will be the tokenization.
5. tokenization is suited for language. However, for mathematics such tokenization is not well suited.

Implementation of BPE

1. Corpus

```
text = """The Dark Knight Rises is a superhero movie released in 2012. It is the final part of Christopher Nolan Da
"""
```

2. Character tokenization

Convert the entire text corpus into ids for each character.

```
ids = [ord(ch) for ch in text]
print(ids)

Output >>
[84, 104, 101, 32, 68, 97, 114, 107, 32, 75, 110, 105, 103, 104, 116, 32, 82, 105, 115, 101, 115, 32, 105, 115, 32,
```

2. Count all adjacent pairs

```
def get_stats(ids):
    counts = {}
    for pair in zip(ids, ids[1:]):
        counts[pair] = counts.get(pair, 0) + 1
    return counts

stats = get_stats(ids)
print(stats)

Output >>
{(84, 104): 3, (104, 101): 8, (101, 32): 10, (32, 68): 3, (68, 97): 3, (97, 114): 6, (114, 107): 3, (107, 32): 3, (
```

3. Select the pair with highest frequency

```
pair = max(stats, key=stats.get)
print(pair)
# for readability define char_pair to print characters corresponding to the pair with highest frequency.
char_pair = (chr(pair[0]), chr(pair[1]))

Output >>
(115, 32)
char_pair = ('s', ' ')
```

4. Assign a new token id to the pair

```
# assuming only ascii characters are in the text, the max id through ord(ch) is 127. So, initializing the next toke
i=1
idx = 127 + i
```

5. Replace all the occurrence of the most frequent pair with the new token id

```
def merge(ids, pair, idx):
    newids = []
    i = 0
    while i < len(ids):
        if i < len(ids) - 1 and ids[i] == pair[0] and ids[i + 1] == pair[1]:
            newids.append(idx)
            i += 2
        else:
            newids.append(ids[i])
            i += 1
    return newids

ids = merge(ids, pair, idx)
print(ids)

Output >>
# The 21st and 22nd ids = (115,32) are now replaced with 128. Followed by 105.
# Similarly all other occurrences of (115,32) are replaced with 128.
[84, 104, 101, 32, 68, 97, 114, 107, 32, 75, 110, 105, 103, 104, 116, 32, 82, 105, 115, 101, 147, 105, 147, 97, 32,
```

6. Repeat the process

```
def get_stats(ids):
    counts = {}
    for pair in zip(ids, ids[1:]):
        counts[pair] = counts.get(pair, 0) + 1
    return counts

def merge(ids, pair, idx):
    newids = []
    i = 0
    while i < len(ids):
        if i < len(ids) - 1 and ids[i] == pair[0] and ids[i + 1] == pair[1]:
            newids.append(idx)
            i += 2
        else:
            newids.append(ids[i])
            i += 1
    return newids

vocab_size = 148  # the desired final vocabulary size
num_merges = vocab_size - 128
ids = list(tokens)  # copy so we don't destroy the original list

merges = {}  # (int, int) -> int
for i in range(num_merges):
    # 1) Count all adjacent pairs in our current sequence 'ids'.
    stats = get_stats(ids)
    pair = max(stats, key=stats.get)
    idx = 128 + i
    # Decode the characters of the pair for display
    char_pair = (chr(pair[0]), chr(pair[1]))
    print(f"merging {pair} ({char_pair[0]}{char_pair[1]}) into a new token {idx}")
    ids = merge(ids, pair, idx)
    merges[pair] = idx

Output >>
merging (115, 32) (s ) into a new token 128
merging (101, 32) (e ) into a new token 129
merging (104, 129) (h) into a new token 130
merging (97, 114) (ar) into a new token 131
merging (110, 32) (n ) into a new token 132
merging (116, 32) (t ) into a new token 133
merging (105, 103) (ig) into a new token 134
merging (134, 104) (h) into a new token 135
merging (101, 114) (er) into a new token 136
merging (114, 101) (re) into a new token 137
merging (97, 132) (a) into a new token 138
merging (66, 97) (Ba) into a new token 139
merging (84, 130) (T) into a new token 140
merging (68, 131) (D) into a new token 141
merging (141, 107) (k) into a new token 142
```

```
merging (142, 32) (□ ) into a new token 143
merging (143, 75) (□K) into a new token 144
merging (144, 110) (□n) into a new token 145
merging (145, 135) (□□) into a new token 146
merging (105, 115) (is) into a new token 147
```

7. Final vocabulary

```
print(tokens)
print(ids)
print("tokens length:", len(tokens))
print("ids length:", len(ids))
print(f"compression ratio: {len(tokens) / len(ids):.2f}X")

Output >>
[84, 104, 101, 32, 68, 97, 114, 107, 32, 75, 110, 105, 103, 104, 116, 32, 82, 105, 115, 101, 115, 32, 105, 115, 32,
[140, 146, 133, 82, 147, 101, 128, 105, 128, 97, 32, 115, 117, 112, 136, 104, 136, 111, 32, 109, 111, 118, 105, 129
tokens length: 309
ids length: 217
compression ratio: 1.42X
```

Jupyter Notebook with code

Byte Pair Encoding & Tiktokenizer library code