Tokenization

Dr M Janaki Meena

Byte-Pair Encoding tokenization

- Developed as an algorithm to compress texts, and then used by OpenAI for tokenization when pretraining the GPT model.
- Used by a lot of Transformer models, including GPT, GPT-2, RoBERTa, BART, and DeBERTa

BPE Training algorithm

- Starts by computing the unique set of words used in the corpus
- Build vocabulary by taking all the symbols used to write those words
- Eg: If our corpus uses these five words:
- "hug", "pug", "pun", "bun", "hugs"
- base vocabulary will then be ["b", "g", "h", "n", "p", "s", "u"]

BPE Training algorithm

- For real-world cases, base vocabulary will contain all the ASCII characters, at the very least
- Probably some Unicode characters as well
- If text that we are tokenizing uses a character that is not in the training corpus, that character will be converted to the unknown token
- That’s one reason why lots of NLP models are very bad at analyzing content with emojis, for instance

BPE Training algorithm

- GPT-2 and RoBERTa tokenizers (which are pretty similar) have a clever way to deal with this
- They don't look at words as being written with Unicode characters, but with bytes
- Hence the base vocabulary has a small size (256), but every character you can think of will still be included and not end up being converted to the unknown token.
- This trick is called byte-level BPE

BPE Training algorithm

- After getting base vocabulary, add new tokens until the desired vocabulary size is reached by learning merges
- Merges – rules to merge two elements of the existing vocabulary together into a new one
- At the beginning these merges will create tokens with two characters, and then, as training progresses, longer subwords.
- At any step during the tokenizer training, the BPE algorithm will search for the most frequent pair of existing tokens

BPE Training algorithm

- By “pair,” here we mean two consecutive tokens in a word
- Most frequent pair will be merged
- rinse and repeat for the next step
- Let’s assume the words had the following frequencies:

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- Meaning "hug" was present 10 times in the corpus, "pug" 5 times, "pun" 12 times, "bun" 4 times, and "hugs" 5 times

BPE Training algorithm

- We start the training by splitting each word into characters (the ones that form our initial vocabulary) so we can see each word as a list of tokens:

("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)

- ("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)

BPE Training algorithm

- ***("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)***
- Look at pairs - ("h", "u") - present in words "hug" and "hugs", so 15 times total in corpus
- Not the most frequent pair
- Pair ("u", "g") is the most frequent pair, which is present in "hug", "pug", and "hugs", for a total of 20 times in the vocabulary.

BPE Training algorithm

- First merge rule learned by the tokenizer is ("u", "g") -> "ug"
- "ug" will be added to the vocabulary, and the pair should be merged in all the words of the corpus
- Vocabulary: ***["b", "g", "h", "n", "p", "s", "u", "ug"]***
- Corpus: ***("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)***

BPE Training algorithm

- Pair ("h", "ug"), for instance (present 15 times in the corpus)
- Most frequent pair at this stage is ("u", "n"), however, present 16 times in the corpus, so the second merge rule learned is ("u", "n") -> "un"
- Adding that to the vocabulary and merging all existing occurrences leads us to:
- Vocabulary: ***["b", "g", "h", "n", "p", "s", "u", "ug", "un"]***
- Corpus: ***("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)***

BPE Training algorithm

- Most frequent pair is ("h", "ug"), so we learn the merge rule ("h", "ug") -> "hug"
- Vocabulary: ***["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]***
- Corpus: ***("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)***
- We continue like this until we reach the desired vocabulary size

Sample Tokenization

- Vocabulary: ***["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]***
- *Tokenize bug and mug*
- bug *– ["b", "ug"]*
- mug *– ["[UNK]", "ug"]*

WordPiece Tokenization

- tokenization algorithm Google developed to pretrain BERT
- reused in quite a few Transformer models based on BERT, such as DistilBERT, MobileBERT, Funnel Transformers, and MPNET
- Very similar to BPE in terms of the training, but the actual tokenization is done differently.

WordPiece Training algorithm

- Like BPE, WordPiece starts from a small vocabulary
- Vocabulary – Includes the special tokens used by the model and the initial alphabet
- It identifies subwords by adding a prefix (## for BERT), each word is initially split by adding that prefix to all the characters inside the word
- So, for instance, "word" gets split like this:
- ***w ##o ##r ##d***

WordPiece Training algorithm

- Thus, initial alphabet contains all the characters present at beginning of a word and characters present inside a word preceded by the WordPiece prefix
- ***w ##o ##r ##d***
- Then apply merge rules like BPE

WordPiece Training algorithm

- Main difference is the way the pair to be merged is selected
- Instead of selecting the most frequent pair, WordPiece computes a score for each pair, using the following formula:
- ***score=(freq_of_pair)/(freq_of_first_element×freq_of_second_element)***
- Then apply merge rules like BPE

WordPiece Training algorithm

- By dividing frequency of pair by product of frequencies of each of its parts, the algorithm prioritizes merging of pairs where individual parts are less frequent in vocabulary
- For instance, it won’t necessarily merge ("un", "##able") this pair occurs very frequently in the vocabulary
- because the two pairs "un" and "##able" appear a lot of other words and have a high frequency

WordPiece Training algorithm

- In contrast, a pair like ("hu", "##gging") will probably be merged faster (assuming the word “hugging” appears often in the vocabulary) since "hu" and "##gging" are likely to be less frequent individually.
- Let’s look at the same vocabulary we used in the BPE training example
- ***("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)***
- ***("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g" "##s", 5)***

WordPiece Training algorithm

- Initial vocabulary
- ***["b", "h", "p", "##g", "##n", "##s", "##u"]***
- ***(if we forget about special tokens for now)***
- Most frequent pair is ("##u", "##g") (present 20 times)
- But the individual frequency of "##u" is very high, so its score is not the highest (it’s 1 / 36)

WordPiece Training algorithm

- All pairs with a "##u" actually have that same score (1 / 36), so the best score goes to the pair ("##g", "##s") — the only one without a "##u"
- at 1 / 20, and the first merge learned is ("##g", "##s") -> ("##gs")
- We remove the ## between the two tokens, so we add "##gs" to the vocabulary
- **Vocabulary:** [***"b", "h", "p", "##g", "##n", "##s", "##u", "##gs"]***
- **Corpus:** ***("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##gs", 5)***highest (it's 1 / 36)

WordPiece Training algorithm

- Now "##u" is in all the possible pairs, so they all end up with the same score
- Let’s say that in this case, the first pair is merged, so ("h", "##u") -> "hu". This takes us to:
- **Vocabulary:** ***["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu"]***
- **Corpus:** ***("hu" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)***

WordPiece Training algorithm

- Next best score is shared by ("hu", "##g") and ("hu", "##gs") (with 1/15, compared to 1/21 for all the other pairs), so the first pair with the biggest score is merged:
- **Vocabulary:** ***["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]***
- **Corpus:** ***("hug", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("hu" "##gs", 5)***
- Continue like this until we reach the desired vocabulary size