Algorithm Analysis

Byte-Pair Encoding (BPE) versus WordPiece algorithm (BERT Implementation)

AI 351: NLP
By Juan Carlos Roldan, MEngAI

We had these results on average between about 1000 runs of the BPE and WordPiece implementations.

```
Unset
Average BPE Compression Ratio: 0.48
Average Wordpiece Compression Ratio: 0.74
Average BPE Compression Time: 0.00605200
Average Wordpiece Compression Time: 0.00295964
Average Intersection Percentage: 16.56%
```

The results provided suggest that WordPiece consistently outperforms BPE, due to BPE consistently producing more tokens. The BPE Compression ratio being at 0.39, whereas WordPiece compression being at 0.75, means that BPE compresses the input text but ends up with way more tokens than WordPiece. This leads us to believe that WordPiece is much more efficient at compression, with respect to token vocabulary lengths.

The reason as to why the BPE algorithm tends to produce more might be due to the fact that it being a greedy bottom-up implementation of tokenization, will try to happily merge subwords together, regardless of semantic information. WordPiece is a little bit more statistically conservative. For example, “unhappyness” might be broken down by BPE into [“un”, “ha”, “ppiness”] which doesn’t seem to stem from the root word “happy”. On the other hand, WordPiece would attempt to break down based on maximizing likelihoods from an existing language model (which takes semantics into consideration and common stem words as structural components). So the above would be [“un”, “##happi”, “##ness”]. WordPiece uses the “##” strategy where it appends that indicator for base words, even in the presence of infrequent inflections of words from that base word (say, “unhappiness” was a rare occurrence, it would still be able to produce tokenization “happi” most of the time, maybe because “happy” occurs a lot of times). That would lead to us preventing “over”-tokenization.

Apart from that, the WordPiece is also more efficient, and the first reason is that it doesn’t tend to over-tokenize. Once it senses common occurrences of base words, it can stop there. This means that BPE might not be used in places where memory consumption is a critical metric in

performance (e.g. completion tasks using transformers), but it might still be useful in cases where we are expected to catch all of the potential inflections, which is the case for languages like Turkish or German.

Algorithm citations:

- The algorithm used to segment the sentences above was through the SpaCy library. See reference below on the SpaCy documentation website.

References:

BPE vs WordPiece Tokenization - when to use / which? (2020, May 6). Retrieved September 4, 2024, from https://datascience.stackexchange.com/questions/75304/bpe-vs-wordpiece-tokenization-when-to-use-which

Summary of the tokenizers. (2023). Retrieved September 4, 2024, from https://huggingface.co/docs/transformers/en/tokenizer_summary

Tripathi, A. (2020, May 4). How to Perform Sentence Segmentation or Sentence Tokenization using spaCy | NLP Series | Part 5. Retrieved September 4, 2024, from https://ashutoshtripathi.com/2020/05/04/how-to-perform-sentence-segmentation-or-sentence-tokenization-using-spacy-nlp-series-part-5/

Kida, T., Takeda, M., Shinohara, A., and Shinohara, T. 1999. Byte Pair Encoding: A Text Compression Scheme That Accelerates Pattern Matching. Retrieved September 4, 2024, from https://www.researchgate.net/publication/2310624

Shi, Y. 2021. Segmented Translation Algorithm of Complex Long Sentences Based on Semantic Features. Journal of Physics: Conference Series, 1881, 042093. DOI: https://doi.org/10.1088/1742-6596/1881/4/042093