Armenian Tokenizer

Generative AI Final Project Report

Students: Anna Shaljyan, Naira Maria Barseghyan

This project aims to develop a tokenization system for the Armenian language and train it using a corpus derived from Armenian Wikipedia. Tokenization is a crucial step in natural language processing (NLP) that involves breaking down text into individual units or tokens, which could be words, characters, or other meaningful elements. The project will explore various tokenization methods that are suitable for the nuances of the Armenian language. By evaluating the effectiveness of these methods, the project aims to determine which approach best captures the linguistic characteristics and structures of Armenian text, leading to improved tokenization accuracy and performance. The ultimate goal is to create a robust tool that can accurately segment Armenian text into tokens, facilitating further NLP tasks such as parsing, part-of-speech tagging, and sentiment analysis.

Tiktoken:

As one of the methods for tokenization of Armenian Wikipedia, we decided to use tiktoken, which is a fast open-source tokenizer created by OpenAI. Initially, we used its base models, such as cl100k_base and p50k_base, which allowed us to load the dataset and the models and automatically encode and later decode any text examples. This created an easy and fast means for us to try this tokenization, and we also noticed that there is not much difference between the two used base models, so we didn't try with other ones. However, we decided to test how it would be to train on our dataset using tiktoken and noticed only one means to do so, by using SimpleBytePairEncoding from tiktoken._educational, which visualizes how the GPT-4 encoder encodes text, making the process of training slow and more suitable for educational purposes only. As the process was slow, we decided to train not on our whole dataset but on parts of it, saving computational resources and making the process time efficient. That's why we trained one time on 10 lines of our dataset, corresponding to 10 articles from Armenian Wikipedia, with a vocabulary of 400, and then 1000 lines of our dataset with the same vocabulary size. We also experimented with 1000 vocabulary size again on the portion of our dataset. All the results of the experiments were saved as pkl files.

BPE tokenizer:

For our tokenization approach, we also employed Byte Pair Encoding (BPE) by creating a custom BpeTokenizer class. This implementation takes advantage of the flexibility and effectiveness of BPE for segmenting text into tokens. Initially, we preprocessed the corpus text by lowercasing, adding spaces around special characters, removing redundant spaces, and fixing HTML entities. Next, we counted occurrences of pairable and unpairable characters in the corpus, initializing the vocabulary accordingly. During BPE learning, we iteratively merged byte pairs based on their frequency, expanding the vocabulary. This iterative process allowed us to efficiently generate a vocabulary suited to our needs. We then experimented with parameters such as vocabulary size and character sets to optimize tokenization performance. Finally, we encoded and decoded text using the learned vocabulary to see how our implementation works. We saved the learned model for reproducibility and the vocabulary of 30000 to both CSV and Excel files for visualizations.

WordPiece tokenizer:

Besides BPE and tiktoken, we also employed the WordPiece algorithm, implemented through a custom WordPieceTokenizer class. This algorithm breaks down text into meaningful subwords based on their frequency within a given corpus. Before tokenization, the text undergoes preprocessing, including lowercase conversion, special character space addition, redundant space removal, and HTML entity fixing. Subsequently, we count all possible substrings (subwords) in the corpus, using these frequencies to construct the vocabulary. During vocabulary learning, we iteratively select the most frequent yet unseen subwords until reaching our desired vocabulary size. Again by experimenting with parameters such as vocabulary size and minimum frequency, we want to reach efficient tokenization. Tokenization involves splitting text into subwords according to the learned vocabulary, with unknown subwords represented by special tokens. As a result, our WordPiece implementation creates means for effective encoding and decoding of text while utilizing as few resources as possible.

SentencePiece tokenizer:

As our fourth tokenization approach, we utilized the SentencePiece library to train a model based on the provided text corpus from Armenian Wikipedia. The training process follows parameters similar to those employed in training Llama 2 for Armenian text. Firstly, the text corpus is moved to a .txt file, which is a specific requirement of the SentencePiece training algorithm. The training algorithm, which is based on Byte Pair Encoding (BPE), aims to construct a vocabulary of subword units tailored to the corpus's characteristics. Key settings include a vocabulary size of 30,000, maximum sentence length, and character coverage. Rare word treatment mechanisms, such as splitting digits and handling whitespace, ensure that the resulting vocabulary achieves logical outcomes. Special tokens, including unknown, beginning of sequence, end of sequence, and padding tokens, are integrated into the vocabulary. Since the model was very heavy, we trained it not on the full data but only on half of the data (150000 articles). Before training the final model we experiment with various vocabulary and data sizes. Even at a vocabulary size of 400 the model was already doing very decent tokenization. This shows that SentencePiece is a very powerful tool that has advanced capabilities and is straightforward to use.

To conclude, we trained our corpus based on the Armenian Wikipedia, using four different approaches: tiktoken, BPE, SentencePiece, and WordPiece, which utilized both custom classes created by us and defined libraries from various Python packages. We saved the resulting models and vocabularies for reproducibility and created word visualizations based on saved vocabularies to understand the logic behind our implemented approaches.