Tokenization Strategies

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Introduction to Large Language Models

Sub-word Tokenization

- Approach
  - A middle ground between word and character-based tokenization strategies
  - Frequently used words should not be split into smaller subwords
  - Rare words should be decomposed into meaningful subwords
- Three common algorithms:
  - Byte-Pair Encoding (BPE) (Sennrich et al., 2016)
  - WordPiece (Schuster and Nakajima, 2012)
  - Unigram language modeling tokenization (Kudo, 2018)
- These algorithms use data, guide the tokenization process:
  - A **token learner** that processes a raw training corpus to generate a vocabulary (a collection of tokens).
  - A **token segmenter** that tokenizes a raw test sentence based on the generated vocabulary.

Byte-Pair Encoding (BPE)

Introduction

- Simplest and commonly used algorithm for tokenization
- Originally introduced as a **text compression** strategy
- The algorithm depends on a pre-tokenizer that divides the training data into individual words.

Widely adopted as a tokenization technique in models like **GPT, GPT-2, RoBERTa, BART,** and **DeBERTa**.

Paper: Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Training Algorithm

function BYTE-PAIR ENCODING(strings C, number of merges k) **returns** vocab V

$V \leftarrow$ all unique characters in C # initial set of tokens is characters
for $i = 1$ **to** k **do** # merge tokens til k times
  $t_L, t_R \leftarrow$ Most frequent pair of adjacent tokens in C
  $t_{NEW} \leftarrow t_L + t_R$ # make new token by concatenating
  $V \leftarrow V + t_{NEW}$ # update the vocabulary
  Replace each occurrence of t_L, t_R in C with t_{NEW} # and update the corpus
return V

Algorithm from Speech and Language Processing book by Daniel Jurafsky and James H. Martin

Breakdown Of BPE Algorithm

1. **Pre-tokenization:**
   - Input: The algorithm starts with a corpus of text data.
   - Pre-tokenization: The corpus is pre-tokenized, usually by splitting the text into words. Pre-tokenization can involve breaking the text at spaces, punctuation, or using more complex rules.
   - After pre-tokenization, the algorithm creates a list of all unique words in the corpus, along with their frequency of occurrence.
2. **Base Vocabulary:**
   - The base vocabulary is initialized with all unique characters (or symbols) found in the list of unique words. For example, if the word "hello" is in the corpus, the symbols 'h', 'e', 'l', 'o' would be part of the initial vocabulary.
   - Each word in the corpus is then represented as a sequence of symbols from this base vocabulary. For instance, "hello" would be represented as ['h', 'e', 'l', 'l', 'o'].

Breakdown Of BPE Algorithm

3. Pair Merging:

- Bigram Counts: The algorithm counts the frequency of adjacent symbol pairs (bigrams) in the list of unique words. For example, in "hello", the bigrams would be ('h', 'e'), ('e', 'l'), ('l', 'l'), ('l', 'o').
- Merging: The most frequent bigram is then merged into a new symbol, and the words in the corpus are updated to reflect this merge. For example, if ('l', 'l') is the most frequent bigram, it is merged into a new symbol, say 'll', and "hello" would be updated to ['h', 'e', 'll', 'o'].

This process continues iteratively until the desired vocabulary size is reached.

Example

- Let us consider the toy corpus which consists of pre-tokenized text and its frequency.

Word	Frequency
cat	10
bat	5
bag	12
tag	4
cats	5

- Form the base vocabulary by taking all the characters that occur in the training corpus.
  - Base vocabulary: a, b, c, g, s, t
- Once we have the initial base vocabulary, we continue adding new tokens until we achieve the target vocabulary size by learning and applying merges.

Example

- At each stage of the training, the BPE algorithm identifies the most frequent pair of existing tokens. This most frequent pair is then merged.
- We split each word into its constituent characters (tokens) as per the base vocabulary.

Tokens	Frequency
c, a, t	10
b, a, t	5
b, a, g	12
t, a, g	4
c, a, t, s	5

Token Pair	Frequency
ca	15
at	20
ba	17
ag	16
ts	5

...

Example

- Select the most frequent pair (a, t) and merge them into a single symbol. Add this newly created symbol to the vocabulary.
  - Vocabulary : a, b, c, g, s, t, at
- The first merge rule learned by the tokenizer is a, t → at and the pair should be merged in all the words of the corpus.

Tokens	Frequency
c, at	10
b, at	5
b, a, g	12
t, a, g	4
c, at, s	5

Token Pair	Frequency
ag	16
cat	15
ba	12
bat	5
ats	5

...

Example

- The most frequent pair at this stage is (a, g).
- The second merge rule learned is a, g → ag. Adding that to the vocabulary and merging all existing occurrences leads us to:
  - Vocabulary : a, b, c, g, s, t, at, ag

Tokens	Frequency
c, at	10
b, at	5
b, ag	12
t, ag	4
c, at, s	5

Example

- Now the most frequent pair is (c, at), so we learn the merge rule c, at → cat.
- After three merges, the vocabulary and corpus are as follows:
  - Vocabulary : a, b, c, g, s, t, at, ag, cat

Tokens	Frequency
cat	10
b, at	5
b, ag	12
t, ag	4
cat, s	5

- We keep iterating through these steps until the vocabulary reaches the desired size.

BPE Algorithm

- To tokenize a text, we run each merge learned from the training data in a **greedy manner**, **successively in the order we learned them,** i.e.

a, t → at

a, g → ag

c, at → cat

- For example, the word “bag” would be tokenized as follows:
  - We begin by splitting the word into its constituent characters: bags → b, a, g, s.
  - We go through the merge rules until we find one we can apply. Observe the second rule can be applied and merge the characters a and g: bags → b, ag, s
  - When we have exhausted all the merge rules, the tokenization process is complete.

bags → b, ag, s

BPE Algorithm

- If the word being tokenized includes a character that was not present in the training corpus, that character will be converted to the unknown token (<UNK>).
  - mat → [UNK], at
- To avoid <UNK>, the base vocabulary must include every possible character or symbol. This can be extensive, specially since there are about ~149K unicode symbols.
- GPT-2 and RoBERTa uses bytes as the base vocabulary (size 256) and then applies BPE on top of this sequence (with some rules to prevent certain types of merges).
- In practice, it is common to add a special end of word symbol "__" before space.

WordPiece Tokenization

Introduction

- The WordPiece algorithm, like Byte-Pair Encoding (BPE), is used for subword tokenization, but it employs a different approach to determine which symbol pairs to merge.

Unlike BPE, merges in WordPiece algorithm are determined by **likelihood**, and **not frequency**.

Adopted as a tokenization technique in language models like **BERT, DistilBERT, MobileBERT, Funnel Transformers,** and **MPNET**

Paper: Schuster, Mike, and Kaisuke Nakajima. "Japanese and korean voice search." In 2012 IEEE international conference on acoustics, speech and signal processing (ICASSP), pp. 5149-5152. IEEE, 2012.

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K. and Klingner, J., 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.

Training Algorithm

- The WordPiece algorithm uses special markers to indicate word-initial and word-internal tokens, which is model specific.
  - For BERT, ## is added as a prefix for any word-internal token.
- To form the base vocabulary, split each word by adding the WordPiece prefix to all word-internal characters. For example, the word “token” would be splitted as:
  - token → t ##o ##k ##e ##n
- At each stage, a score is computed for each pair of tokens in our vocabulary:

$$score = \frac{freq\ of\ pair}{freq\ of\ first\ token\ *freq\ of\ second\ token}$$

The pair of tokens with highest score is selected to be merged.

- Add the merged pair to the vocabulary and continue the process until the vocabulary reaches the desired size.

Breakdown Of WordPiece Algorithm

- Steps 1 (Pre-tokenization) and 2 (Base Vocabulary) remain the same as BPE.

3. **Pair Merging:**
   - While BPE merges the most frequent symbol pair at each step, WordPiece chooses the symbols that maximize the likelihood of the training data when added to the vocabulary.
   - Merge Criteria: The algorithm identifies the symbol pair whose merge would maximize the likelihood of the training data. This is determined by selecting the pair with the highest score, defined as the probability of the merged symbol divided by the product of the probabilities of its individual components.
   - The selected symbol pair is then merged into a new symbol, and the vocabulary is updated accordingly.

This process is repeated iteratively until the desired vocabulary size is reached.

Example

- Let us take a look at the toy corpus, which we will use to train the WordPiece tokenizer.

Word	Frequency
sunflower	1
sun	2
flower	1
flow	1
flowers	1
flowing	2
flows	2
flowed	1

Example

- To create the initial vocabulary, we break down each word in the training corpus into its constituent letters and prepend the WordPiece prefix to the word-internal characters.

Word	Frequency
s, ##u, ##n, ##f, ##l, ##o, ##w, ##e, ##r	1
s, ##u, ##n	2
f, ##l, ##o, ##w, ##e, ##r	1
f, ##l, ##o, ##w	1
f, ##l, ##o, ##w, ##e, ##r, ##s	1
f, ##l, ##o, ##w, ##i, ##n, ##g	2
f, ##l, ##o, ##w, ##s	2
f, ##l, ##o, ##w, ##e, ##d	1

- By retaining only a single occurrence of each element, we get the following vocabulary:
  - ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s

Example

- We will calculate the score for each pair.

Token Pair	Score
s, ##u	0.33
##u, ##n	0.2
##n, ##f	0.2
##f, ##l	0.11
##l, ##o	0.11
. . .	. . .
##e, ##r	0.25
##e, ##d	0.25

- Select the pair with highest score to be merged - s, ##u

s + ##u → su

Example

- We add the merged pair to the vocabulary and apply the merge to the words in the corpus.
  - Vocabulary : ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su
  - Corpus:

Word	Frequency
su, ##n, ##f, ##l, ##o, ##w, ##e, ##r	1
su, ##n	2
f, ##l, ##o, ##w, ##e, ##r	1
f, ##l, ##o, ##w	1
f, ##l, ##o, ##w, ##e, ##r, ##s	1
f, ##l, ##o, ##w, ##i, ##n, ##g	2
f, ##l, ##o, ##w, ##s	2
f, ##l, ##o, ##w, ##e, ##d	1

Example

- We continue this process for a few more steps.
- Compute score for all pair of tokens.

Token Pair	Score
su, ##n	0.2
##n, ##f	0.2
##f, ##l	0.11
##e, ##r	0.25
. . .	. . .
##e, ##d	0.25

- The best score is shared by ##e, ##r and ##e, ##d. Let's say, we select ##e, ##r as the best pair and merge them.

##e + ##r → ##er

Example

- At this point of the training algorithm, we have
  - Vocabulary : ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su, ##er
  - Corpus:

Word	Frequency
su, ##n, ##f, ##l, ##o, ##w, ##er	1
su, ##n	2
f, ##l, ##o, ##w, ##er	1
f, ##l, ##o, ##w	1
f, ##l, ##o, ##w, ##er, ##s	1
f, ##l, ##o, ##w, ##i, ##n, ##g	2
f, ##l, ##o, ##w, ##s	2
f, ##l, ##o, ##w, ##e, ##d	1

Example

- Next ##e, ##d has the highest score and is merged.

Token Pair	Score
su, ##n	0.2
##n, ##f	0.2
##f, ##l	0.11
. . .	. . .
##e, ##d	1

##e + ##d → ##ed

- Vocabulary : ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su, ##er, ##ed
- We continue this process until we reach the desired vocabulary size.

Tokenization Algorithm

- Tokenization in WordPiece differs from BPE.
- In WordPiece retains only the final vocabulary and does not store the merge rules learned during the process.
- To tokenize a word, WordPiece identifies the longest subword available in the vocabulary and then performs the split based on that subword.

Example

- For example, let's take the word – *fused*.
- If we use the vocabulary learned in the example, for the word "fused" the longest subword starting from the beginning that is present in the vocabulary is "f", so we split there and get "f", "##used".
- For "##used," the longest subword in the vocabulary is "##u," so you split into ["##u", "##sed"].
- Next, for "##sed," the longest subword in the vocabulary is "##s," so you split into ["##s", "##ed"].
- Finally, "##ed" is a complete token in the vocabulary, so no further splitting is needed.
- So the full breakdown for "fused" would be:

["f", "##u", "##s", "##ed"]

- Vocabulary : ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su, ##er, ##ed

Example

- If it's not possible to find a subword in the vocabulary, the **whole word** is tokenized as **unknown**

funny => "f", "##u", and "##n" present but "##y" is absent. *funny* will be replaced by <UNK>

In BPE, only the missing token will be replaced by <UNK>
funny => "f", "##u", and "##n" present but "##y" is absent. *funny* = > "f" + "##u" + "##n" + "##n" + <UNK>

- Vocabulary : ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su, ##er, ##ed

Unigram Tokenization

Introduction

- Observation: There exists ambiguities in subword segmentation, as a single word or sentence can be divided into different subwords even when using the same vocabulary.
- Problem: The BPE algorithm doesn't support multiple segmentations because it operates using a deterministic and greedy approach.
- Solution: Introduce multiple subword candidates during the training
- The algorithm is based on Expectation–Maximization (EM) algorithm.

Commonly employed in **SentencePiece**, the tokenization method used by models such as **ALBERT**, **T5**, **mBART**, **Big Bird**, and **XLNet**

Paper: Taku Kudo. 2018. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 66–75, Melbourne, Australia. Association for Computational Linguistics.

Breakdown of Unigram LM Tokenization

1. **Initializing the Base Vocabulary**
   - There are various methods for creating the seed vocabulary. A common approach is to include all characters and the most frequent substrings found in the corpus.
2. **Log-Likelihood Loss Computation**
   - Unigram Language Model: At each training step, the algorithm uses a unigram language model, which assumes that each token in the vocabulary is independent of the others.
   - The algorithm calculates the log-likelihood loss over the entire training data based on the current vocabulary. The loss measures how well the current set of symbols can represent the training data.
   - The goal is to minimize this loss, meaning the algorithm aims to find the smallest and most effective vocabulary that still adequately represents the training data.

Breakdown of Unigram LM Tokenization

3. **Finding Candidates for Removal**
   - For each symbol in the vocabulary, the algorithm calculates the potential increase in log-likelihood loss that would occur if that symbol were removed.
   - Symbols that contribute less to the overall representation of the data (i.e., those that cause the smallest increase in loss when removed) are marked as candidates for removal.
4. **Progressive Vocabulary Pruning**
   - The algorithm removes a small percentage (typically 10% to 20%) of the symbols that have the least impact on the log-likelihood loss. These are the symbols for which the increase in training loss is the lowest if they are removed.

This pruning process is repeated iteratively. After each round of pruning, the log-likelihood loss is recalculated with the updated vocabulary, and the process continues until the vocabulary reaches the desired size.

Example

- Let us consider the following toy corpus.

Word	Frequency
run	3
bug	5
fun	13
sun	10

- We will include all possible strict substrings in the initial vocabulary.
  - Vocabulary: r, u, n, ru, un, b, g, bu, ug, f, fu, s, su
- We will start with the first iteration of the algorithm.

Example

- Let's compute the frequency of different tokens in the vocabulary.

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su
Freq	3	31	26	3	26	5	5	5	5	13	13	10	10

- The probability of a specific token is calculated by dividing its frequency in the original corpus by the total sum of frequencies of all tokens in the vocabulary.
  - For example, the probability of the subword *hu* is $\frac{15}{155}$.

Example (E-step)

- Let's compute the frequency of different tokens in the vocabulary.

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su
Freq	3	31	26	3	26	5	5	5	5	13	13	10	10

Token	r	u	n	ru	un	b	g	bu	ug	f	fu	s	su
Freq	0.0194	0.2	0.1677	0.0194	0.1677	0.0323	0.0323	0.0323	0.0323	0.0839	0.0839	0.0645	0.0645

- The probability of a specific token is calculated by dividing its frequency in the original corpus by the total sum of frequencies of all tokens in the vocabulary.
  - For example, the probability of the subword *su* is $\frac{10}{155}$.

Example

- To tokenize a given word using the Unigram model, we first consider all possible segmentations of the word into tokens.
- Since the Unigram model treats all tokens as independent, the probability of a specific segmentation is simply the product of the probabilities of each token in that segmentation.

Example

- Let us consider the word *run*.
  - Possible segmentations: (r, u, n); (ru, n); (r, un)
  - Probability :
    - P (r, u, n) = P(r) x P(u) x P(n) = 0.0194 x 0.2 x 0.1677 = 0.000650676
    - P (ru, n) = P(ru) x P(n) = 0.0194 x 0.1677 = 0.00325338
    - P (r, un) = P(r) x P(un) = 0.0194 x 0.1677 = 0.00325338
- The tokenization of a word using the Unigram model is chosen as the one with the highest probability among all possible segmentations.
  - The word *run* could be tokenized as either (ru, n) or (r, un), let' s say we select (ru, n).

In practice, the **Viterbi algorithm** is used to find the most probable segmentation of a word/sentence from the set of possible segmentation candidates.

Example

- At each stage of training, the loss is calculated by tokenizing every word in the corpus using the current vocabulary.

Word	Freq	Split	Score
run	3	ru, n	0.00325338
bug	5	bu, g	0.00104329
fun	13	fu, n	0.01407003
sun	10	su, n	0.01081665

- Loss = $\sum freq * (-loss(P(word)))$

  = 3 x (-log(0.00325338)) + 5 x (-log(0.00104329)) + 13 x (-log(0.01407003)) +

  10 x (-log(0.01081665)) = 66.102

Example (M-step)

- Our goal is to reduce the vocabulary size.
- To determine which token to remove, we will calculate the associated loss for each token in the vocabulary that is not an elementary token, then compare these losses.
- For example, let's remove the token *un*.

Token	r	u	n	ru	b	g	bu	ug	f	fu	s	su
Freq	0.0194	0.2	0.1677	0.0194	0.0323	0.0323	0.0323	0.0323	0.0839	0.0839	0.0645	0.0645

- Possible segmentations for the word *run*: (r, u, n); (ru, n); (r, un)

Example (M-step)

- Our goal is to reduce the vocabulary size.
- To determine which token to remove, we will calculate the associated loss for each token in the vocabulary that is not an elementary token, then compare these losses.
- For example, let's remove the token *un*.

Token	r	u	n	ru	b	g	bu	ug	f	fu	s	su
Freq	0.0194	0.2	0.1677	0.0194	0.0323	0.0323	0.0323	0.0323	0.0839	0.0839	0.0645	0.0645

- Possible segmentations for the word *run*: (r, u, n); (ru, n); ~~(r, un)~~
  - Probability :
    - P (r, u, n) = P(r) x P(u) x P(n) = 0.0194 x 0.2 x 0.1677 = 0.000650676
    - P (ru, n) = P(ru) x P(n) = 0.0194 x 0.1677 = 0.00325338

Example

Word	Freq	Split	Score
run	3	ru, n	0.00325338
bug	5	bu, g	0.00104329
fun	13	fu, n	0.01407003
sun	10	su, n	0.01081665

- Loss (after removal of token un)

  = 3 x (-log(0.00325338)) + 5 x (-log(0.00104329)) + 13 x (-log(0.01407003)) + 10 x (-log(0.01081665))

  = 66.102 (unchanged)

Example

- For the first iteration, removing any token would not affect the loss.

Token removed from vocabulary	Loss
ru	66.102
un	66.102
bu	66.102
ug	66.102
fu	66.102
su	66.102

- Randomly, we select the token *un* and remove it from the vocabulary. We proceed with the second iteration.

Example (E-step)

- We recompute the probabilities after removal of the token *un*.

Token	r	u	n	ru	b	g	bu	ug	f	fu	s	su
Freq	3	31	26	3	5	5	5	5	13	13	10	10

Token	r	u	n	ru	b	g	bu	ug	f	fu	s	su
Freq	0.0233	0.2403	0.2016	0.0233	0.0388	0.0388	0.0388	0.0388	0.1008	0.1008	0.0775	0.0775

Word	Freq	Split	Score
run	3	ru, n	0.00469728
bug	5	bu, g	0.00150544
fun	13	fu, n	0.02032128
sun	10	su, n	0.015624

Loss = 3 x (-log(0.00469728)) + 5 x (-log(0.00150544))
+ 13 x (-log(0.02032128)) + 10 x (-log(0.015624))
= 61.155

Example (M-step)

- Next, we compute the impact of each token on the loss.

Token removed from vocabulary	Loss
ru	63.0126
bu	61.155
ug	61.155
fu	69.205
su	67.347

- Assuming we are removing one token at each step, we can remove either *bu* or *ug* from the vocabulary at this iteration.