Tokenization

CS XXX: Introduction to Large Language Models

Contents

- Tokenization
- Word Level Tokenization
- Character Level Tokenization
- Subword Tokenization
- Tokenizer
- Byte Pair Encoding
- WordPiece
- Unigram Language Model Tokenization

Tokenization

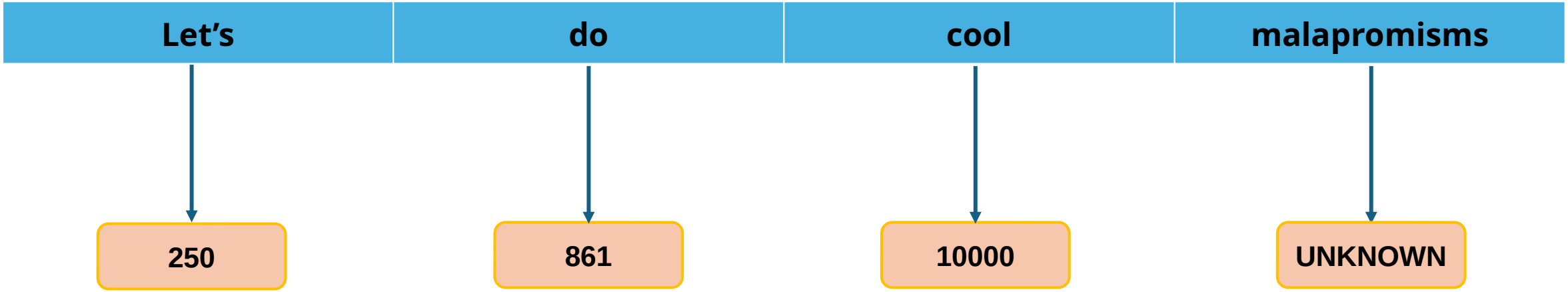
- How do we represent an input text?
- So far in this class… we chop it up into words

Input text: students opened their books

Input token IDs:	11	298	34	567

Word Level Tokenization

- Word Based: Splitting a raw text into words
  - Out of vocabulary words result in a loss of information



Word Level Tokenization

- Not as simple as split on whitespace and punctuation…

- Consider the example

  “Dr. Smith's email is john.doe@example.com, and he said: 'Meet me at 5:00 p.m.!”

  If you tokenize based on whitespace and punctuation, you might get:

```
["Dr", "Smith", "s", "email", "is", "john", "doe", "example", "com", "and", "he", "said", "Meet", "me", "at", "5", "00", "p", "m"]
```

Word Level Tokenization

- Consider the example

  “Dr. Smith's email is john.doe@example.com, and he said: 'Meet me at 5:00 p.m.!”

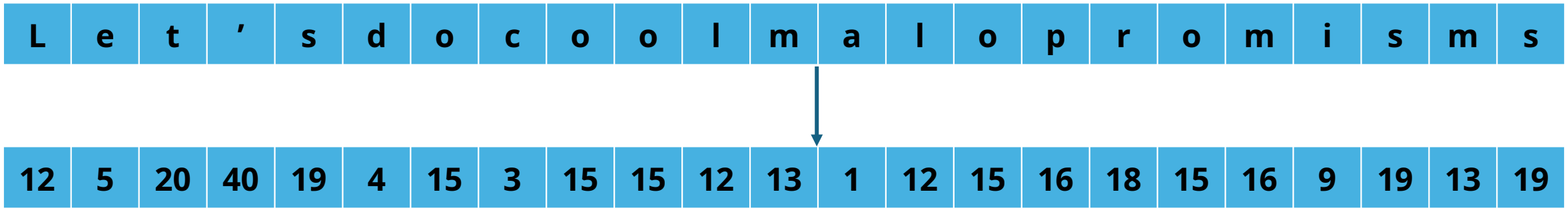
  If you tokenize based on whitespace and punctuation, you might get:

```
["Dr", "Smith", "s", "email", "is", "john", "doe", "example", "com", "and", "he",
                "said", "Meet", "me", "at", "5", "00", "p", "m"]
```

- Problems
  - **Incorrect Tokenization of Possessives**: "Smith's" becomes ["Smith", "s"], which loses the possessive meaning.
  - **Email Address Splitting**: "john.doe@example.com" is broken into ["john", "doe", "example", "com"], which isn't useful.
  - **Time Format Issue**: "5:00 p.m." turns into ["5", "00", "p", "m"], making it hard to understand as a time reference.
  - **Quote Handling**: Quotation marks are separated, making reconstructing the spoken sentence difficult.

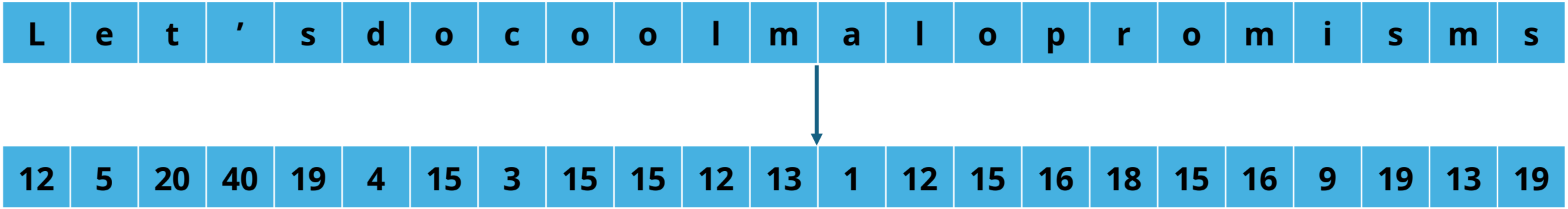
Character Level Tokenization

- Character Based: Splitting a raw text into characters
  - Out of vocabulary words result in a loss of information



Character Level Tokenization

- Character Based: Splitting a raw text into characters
  - Out of vocabulary words result in a loss of information

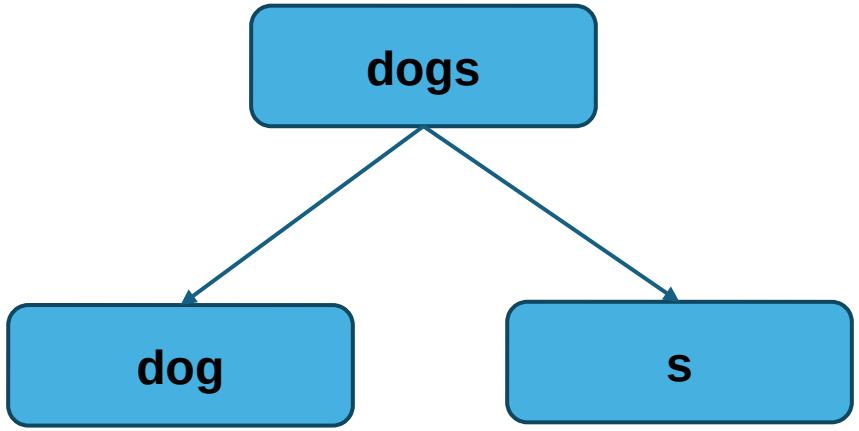


- While character tokenization reduces vocabulary size, it comes at the cost of increased sequence length, making it impractical for large-scale models unless handled efficiently

Sub-word Based Tokenization

- Sub-word Based: Splitting a raw text into subwords
  - Middle ground between word and character based algorithms

Rare words should be decomposed into meaningful subwords



Sub-word Based Tokenization

- Most models obtaining state-of-the-art results in English today use some kind of subword-tokenization algorithm

Byte-Pair Encoding

WordPiece

Unigram

Tokenizer

- At training time, the tokenizer processes the raw training corpus and learns a vocabulary by applying a tokenization algorithm (e.g., BPE, WordPiece, or Unigram).
- At test (or inference) time, the pre-learned vocabulary is used to tokenize new, unseen text without modifying the vocabulary. The text is split into tokens according to the rules learned during training.

Byte Pair Encoding (BPE)

- 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.

Byte Pair Encoding (BPE)

- The BPE algorithm has three steps:
  - Pre-tokenization
  - Base vocabulary
  - Pair Merging

Byte Pair Encoding (BPE)

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.

   - 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

Byte Pair Encoding (BPE)

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’].
   - Form the base vocabulary by taking all the characters that occur in the training corpus.
   - Base vocabulary: a, b, c, g, s, t

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

Byte Pair Encoding (BPE)

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.

Byte Pair Encoding (BPE)

3. Pair Merging:
   - 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.

Vocabulary = [a, b, c, g, s, t]

Split into characters

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

Unique Bigrams

Tokens	Frequency
ca	15
at	20
ba	17
ag	16
ts	5

Byte Pair Encoding (BPE)

3. Pair Merging:
   - 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.

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

Tokens	Frequency
ag	16
cat	15
ba	12
bat	5
ats	5

Byte Pair Encoding (BPE)

3. Pair Merging:
   - 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

Byte Pair Encoding (BPE)

3. Pair Merging:
   - 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.

Byte Pair Encoding (BPE)

- On test or inference, 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

Byte Pair Encoding (BPE)

- Handling Unknown Words
  - 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.

WordPiece Tokenization

- 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 likely-hood, and not frequency.

WordPiece Tokenization

- 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 wordinternal 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:

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

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

WordPiece Tokenization

- Like BPE, the WordPiece Algorithm has the following steps:
  - Pre-tokenziation
  - Base Vocabulary
  - Pair Merging

WordPiece Tokenization

- 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

WordPiece Tokenization

- Pre-tokenization

Word	Frequency
s, ##u, ##n, ##f, ##l, ##o, ##w, ##e, ##r 1	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

WordPiece Tokenization

- Base Vocabulary

Vocabulary = [##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s]

WordPiece Tokenization

- Pair Merging

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

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

- ##u occurs 3 times
- su occurs 3 times

$$\text{score}_{s,\#\#u} = \frac{3}{3 \times 3} = 0.33$$

WordPiece Tokenization

- Pair Merging
- We will calculate the score for each pair.

Token Pair	Frequency
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

$$s + ##u \rightarrow su$$

WordPiece Tokenization

- Pair Merging
- 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]

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

WordPiece Tokenization

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

Token Pair	Frequency
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.

$$\#\#e + \#\#r \rightarrow \#\#er$$

WordPiece Tokenization

- Pair Merging
- 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]

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

WordPiece Tokenization

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

Token Pair	Frequency
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.

WordPiece Tokenization

- On test or inference, 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.

WordPiece Tokenization

Vocabulary = [##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w, f, s, su, ##er, ##ed]

- 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"]

WordPiece Tokenization

- Handling Unknown Words
- Unlike BPE, 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>

Unigram Language Model Tokenization

- A single word or sentence can be divided into different subwords even when using the same vocabulary.
  - For example: run can be divided into r,u,n r,un ru,n
- Unlike additive techniques like BPE, Unigram Language Model tokenization introduces multiple subword candidates during training and prunes less probable tokens from an initially large vocabulary.

Unigram Language Model Tokenization

- A single word or sentence can be divided into different subwords even when using the same vocabulary.
  - For example: run can be divided into r,u,n r,un ru,n
- Unlike additive techniques like BPE, Unigram Language Model tokenization introduces multiple subword candidates during training and prunes less probable tokens from an initially large vocabulary.

Unigram Language Model Tokenization

- The Unigram Language Model tokenization algorithm can be divided into 4 steps
  1. Initializing the Base Vocabulary
  2. Log-Likelihood Loss Computation
  3. Finding Candidates for Removal
  4. Progressive Vocabulary Pruning

Unigram Language Model 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.
- 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]

Unigram Language Model Tokenization

2. Log-Likelihood Loss Computation

- Let's start by computing 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.

$$\mathrm{P}(\mathrm{su}) = \frac{10}{155}$$

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

Unigram Language Model Tokenization

2. Log-Likelihood Loss Computation

- To tokenize a given word using the Unigram model, we first consider all possible segmentations of the word into tokens.
  - Let us consider the word run.

- 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.
  - 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 word run could be tokenized as either (ru, n) or (r, un), let' s say we select (ru, n).

Unigram Language Model Tokenization

2. Log-Likelihood Loss Computation

- 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

- $\text{Loss} = \sum_{i=1}^{N} \text{Freq} \times -\log(\text{Score})$

$$= (3 \times -\log(0.00325338)) + (5 \times -\log(0.00104329))$$
$$+ (13 \times -\log(0.01407003)) + (10 \times -\log(0.01081665)) = 66.102$$

Unigram Language Model Tokenization

3. Finding Candidates For Removal

- 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)

Unigram Language Model Tokenization

3. Finding Candidates For Removal

- 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)
  - 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

Unigram Language Model Tokenization

3. Finding Candidates For Removal

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*)

$$= (3 \times -\log(0.00325338)) + (5 \times -\log(0.00104329)) + (13 \times -\log(0.01407003)) + (10 \times -\log(0.01081665)) = 66.102 \text{ (unchanged)}$$

Unigram Language Model Tokenization

3. Finding Candidates For Removal

- 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.

Unigram Language Model Tokenization

4. Vocabulary Pruning

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

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.

Unigram Language Model Tokenization

4. Vocabulary Pruning

- 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
$= 3 \times (-\log(0.00469728)) + 5 \times (-\log(0.00150544)) + 13 \times (-\log(0.02032128)) + 10 \times (-\log(0.015624))$
$= 61.155$

Unigram Language Model Tokenization

4. Vocabulary Pruning

- 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.

References

- IIT Delhi. (2024-25). Large language models (LLMs): Introduction and recent advances (ELL881/AIL821, Semester I).