Subword

2110572: Natural Language Processing Systems

Peerapon Vateekul & Ekapol Chuangsuwanich
Department of Computer Engineering,
Faculty of Engineering, Chulalongkorn University

Outline

- Introduction
-

Introduction

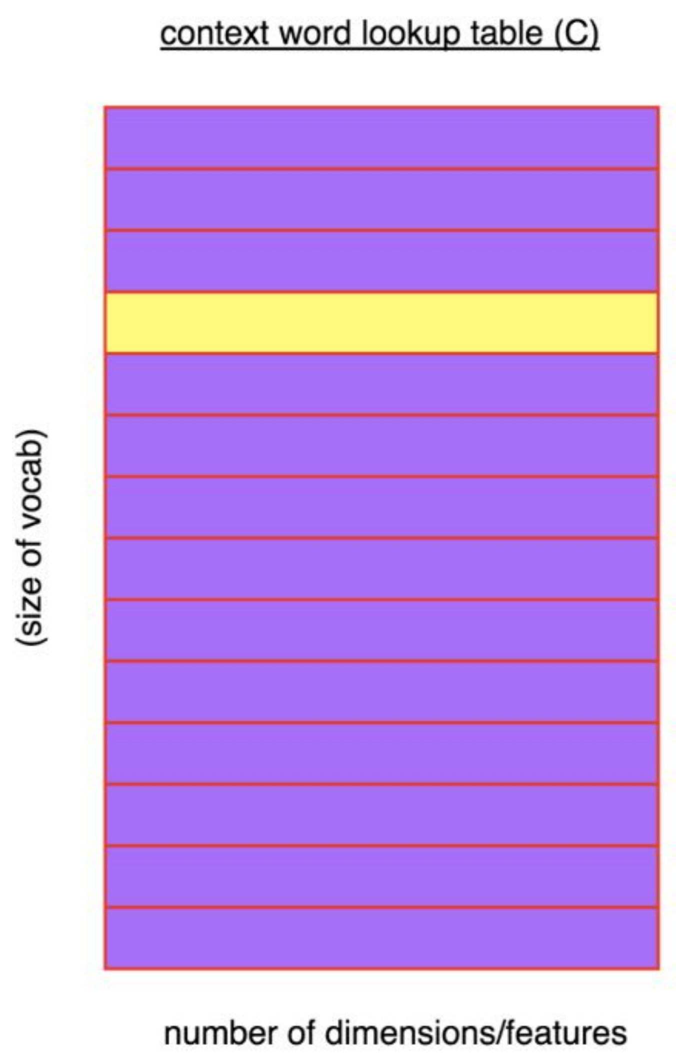
Introduction

Problem:

1) out-of-vocab
2) large vocabulary size

Solution: subword embedding

1) Byte-Pair Encoding (BPE)
2) WordPiece
3) Unigram
4) Sentencepiece



Byte-Pair Encoding (BPE)

Byte-Pair Encoding (BPE)

BPE was introduced in Neural Machine Translation of Rare Words with Subword Units (Sennrich et al., 2015).

Used in GPT-2, Roberta, and even ChatGPT

Relies on a pre-tokenizer that splits the training data into words.

Next, BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns merge rules to form a new symbol from two symbols of the base vocabulary (similar to huffman coding; frequencies).

+ BPE example(1 sentence)

- aaabdaaaba
- **Z**abd**Z**abac
  - Z=aa
- Z**Y**dZ**Y**ac
  - Y=ab
  - Z=aa
- **X**d**X**ac
  - X=ZY
  - Y=ab
  - Z=aa

- พราว และ ขาว นั่ง บน ราว ดู ข่าว คราว บน ดาว
- พร**x** และ ข**x** นั่ง บน ร**x** ดู ข่**x** คร**x** บน ด**x**
  - x=าว
- พ**y** และ ข**x** นั่ง บน **y** ดู ข่**x** ค**y** บน ด**x**
  - x=าว
  - y=รx
- พ**y** และ ข**x** นั่ง **z** **y** ดู ข่**x** ค**y** z ด**x**
  - x=าว
  - y=รx
  - z=บน

BPE - training

Example corpus

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

The most frequent symbol pair is "u" followed by "g", occurring *10 + 5 + 5 = 20* times in total. Thus, the first merge rule the tokenizer learns is to group all "u" symbols followed by a "g" symbol together. Next, "ug" is added to the vocabulary.

```
("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

BPE - usage

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

Tokenization algorithm

Tokenization follows the training process closely, in the sense that new inputs are tokenized by applying the following steps:

1. Normalization
2. Pre-tokenization
3. Splitting the words into individual characters
4. Applying the merge rules learned in order on those splits

Let's take the example we used during training, with the three merge rules learned:

```
("u", "g") -> "ug"
("u", "n") -> "un"
("h", "ug") -> "hug"
```

How to use

- bug = [“b”, “ug”] (“b” in dict)
- mug = [“UNK”, “ug”] (“m” not in dict)
- thug = [“UNK”, “hug”] (“t” not in dict)

Wordpiece

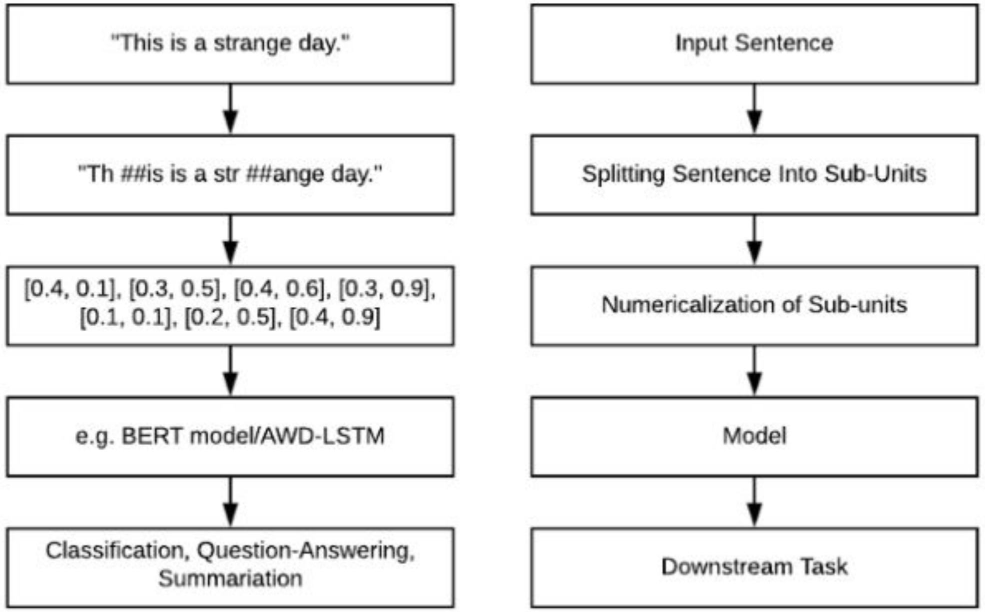
WordPiece

- Google NMT(GNMT) uses a variant of this
  - V1: wordpiece model
  - V2: sentencepiece model
- Rather than char n-gram count, uses a greedy approximation to maximizing language model log likelihood to choose the pieces (add n-gram that maximally reduces perplexity)
- Like BPE, WordPiece learns merge rules. The 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:

$$\text{score} = (\text{freq\_of\_pair})/(\text{freq\_of\_first\_element} \times \text{freq\_of\_second\_element})$$

WordPiece (cont.)

- WordPiece is the subword tokenization algorithm used for models such as BERT, DistilBERT, and Electra.
- There are 2 types of tokens: start token (no ##), and continuing token (##)



https://jacky2wong.medium.com/understanding-sentencepiece-under-standing-sentence-piece-ac8da59f6b08

WordPiece - training

Corpus

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

The splits here will be:

```
("h" "##u" "##g", 10), ("p" "##u" "##g", 5), ("p" "##u" "##n", 12), ("b" "##u" "##n", 4), ("h" "##u" "##g
```

so the initial vocabulary will be `["b", "h", "p", "##g", "##n", "##s", "##u"]`

WordPiece - training (cont.)

Corpus

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

From initial vocab ["b", "h", "p", "##g", "##n", "##s", "##u"]

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")

```
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)
```

+ WordPiece - usage

Tokenization differs in WordPiece and BPE in that WordPiece only saves the final vocabulary, not the merge rules learned.

```
Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]
```

How to use: "the longest subword"

- hugs = ["hug", "##s"]

If not possible to find subwords, tokenize the whole word as UNK.

- mug = ["UNK"]
- bum = ["UNK"] ~~(not ["b", "##u", UNK])~~

+ WordPiece - usage

Tokenization differs in WordPiece and BPE in that WordPiece only saves the final vocabulary, not the merge rules learned.

```
Vocabulary: ["b", "h", "p", "##g", "##n", "##s", "##u", "##gs", "hu", "hug"]
```

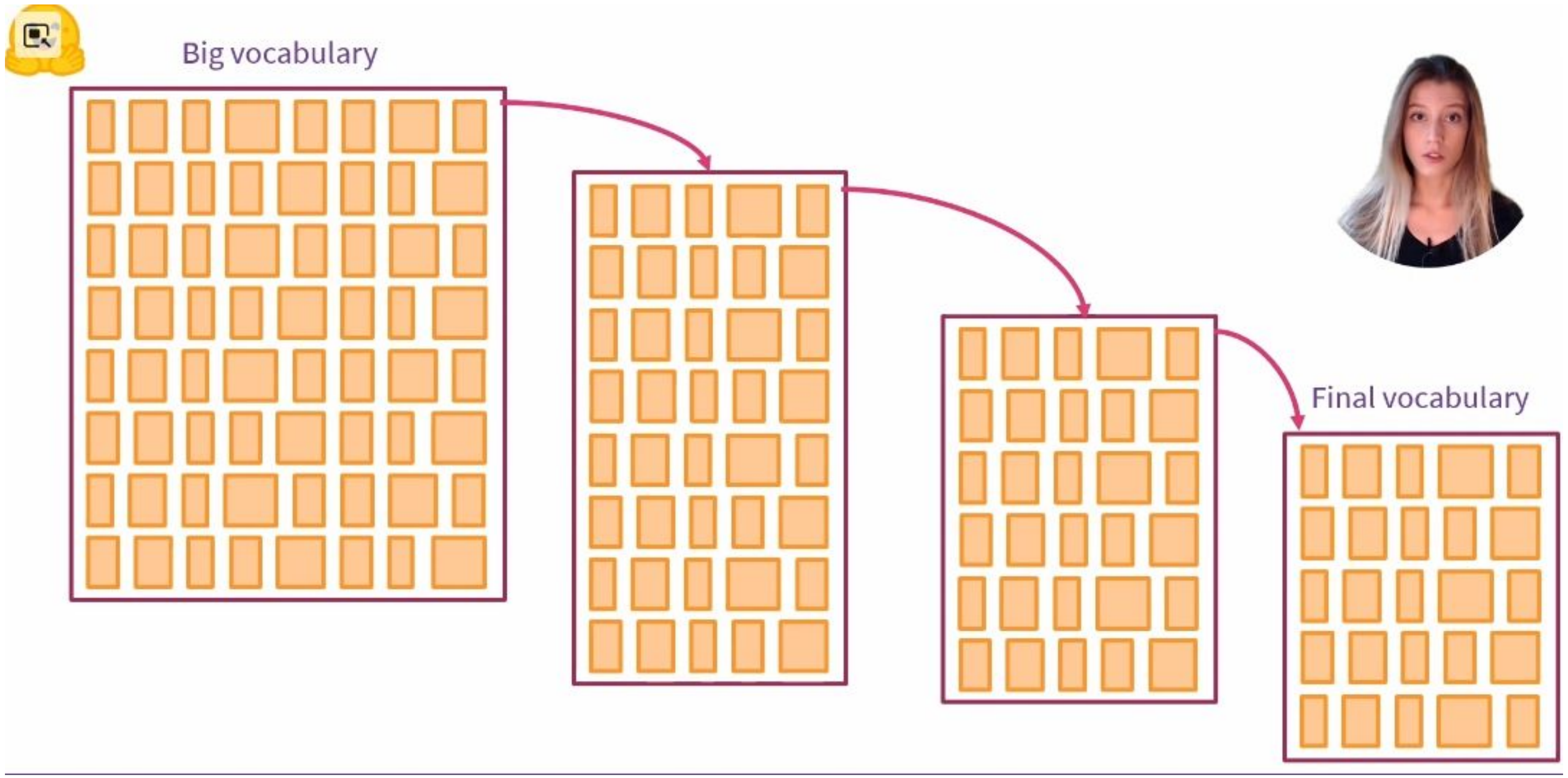
if we use the vocabulary learned in the example above, for the word "hugs" the **longest** subword starting from the beginning that is inside the vocabulary is "hug", so we split there and get ["hug", "##s"]. We then continue with "##s", which is in the vocabulary, so the tokenization of "hugs" is ["hug", "##s"].

When the tokenization gets to a stage where it's not possible to find a subword in the vocabulary, the whole word is tokenized as unknown — so, for instance, "mug" would be tokenized as ["[UNK]"], as would "bum" (even if we can begin with "b" and "##u", "##m" is not the vocabulary, and the resulting tokenization will just be ["[UNK]"], not ["b", "##u", "[UNK]"]). This is another difference from BPE, which would only classify the individual characters not in the vocabulary as unknown.

Unigram

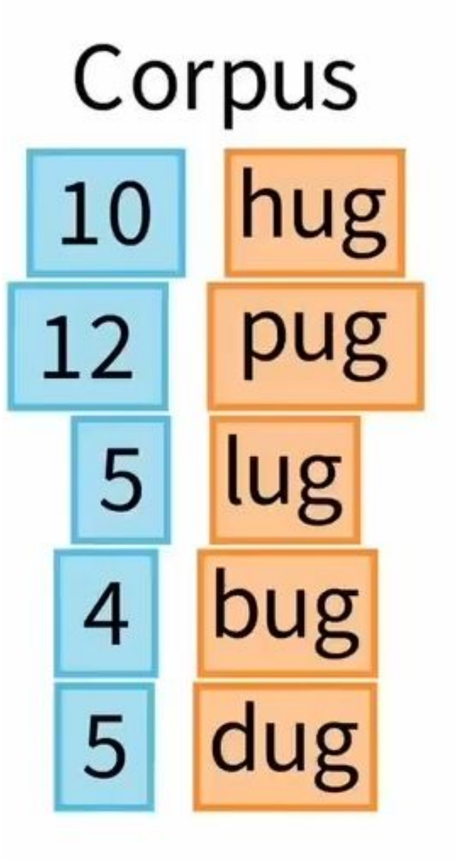
Unigram

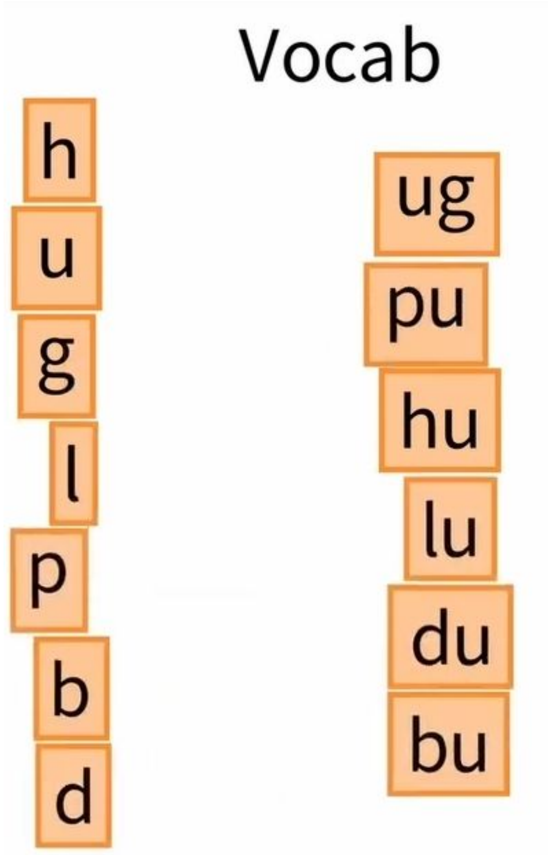
Start with a big vocab and reduce it based on a unigram LM loss



Unigram - training

Initial vocab = all substring of corpus





Unigram - training (cont.)

Corpus

10	hug
12	pug
5	lug
4	bug
5	dug

Vocab

h, u, g, l, p, b, d

ug, pu, hu, lu, du, bu

1st iteration of EM

The E step. Select the split for each word in the corpus with highest prob.

Vocab

h	10/180	ug	36/180
u	36/180	pu	12/180
g	36/180	hu	10/180
l	5/180	lu	5/180
p	12/180	du	5/180
b	4/180	bu	4/180
d	5/180		

Possible splits for "hug"

h u g $\frac{10}{180} \times \frac{36}{180} \times \frac{36}{180} = 2.22e-03$

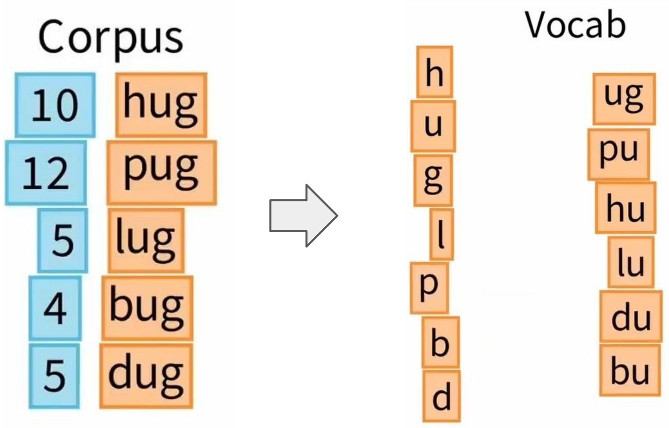
Choose 1 randomly

hu g $\frac{10}{180} \times \frac{36}{180} = 1.11e-02$

h ug $\frac{10}{180} \times \frac{36}{180} = 1.11e-02$

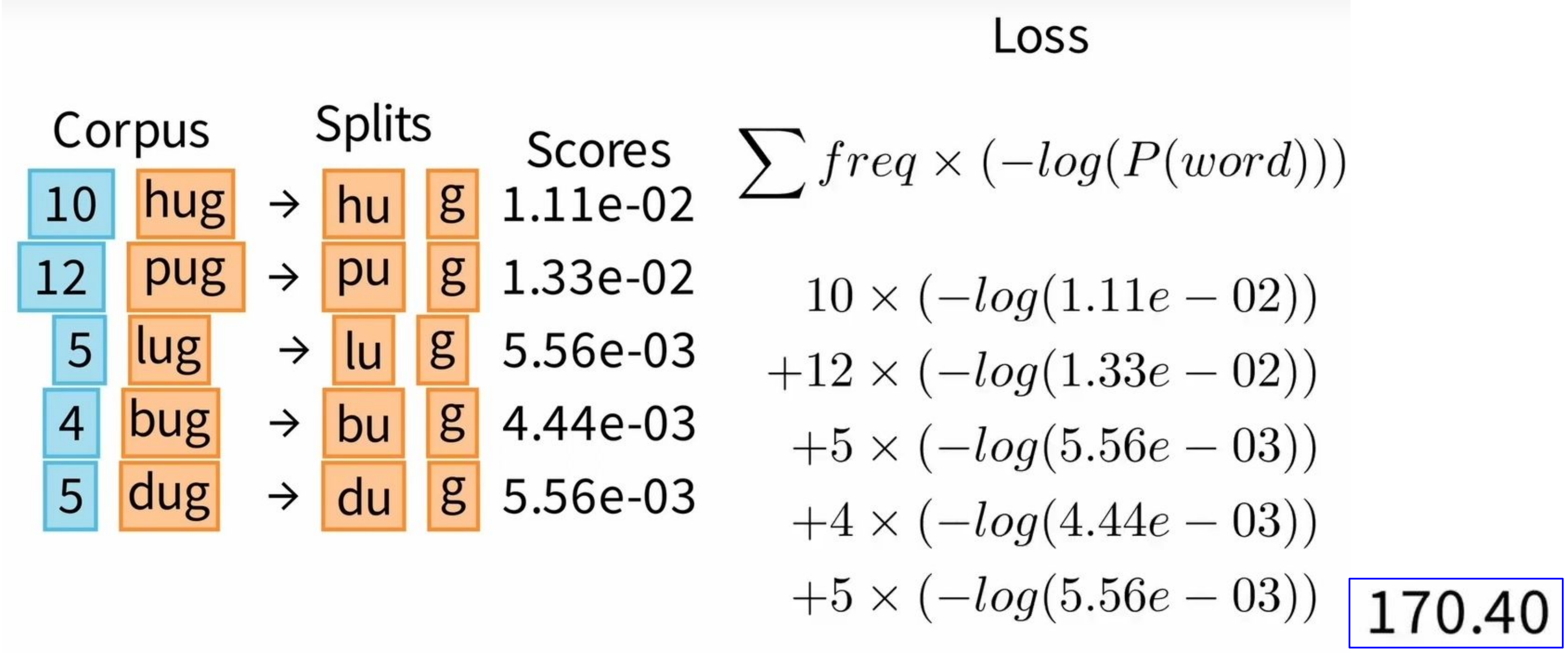
hug 0

Unigram - training (cont.)

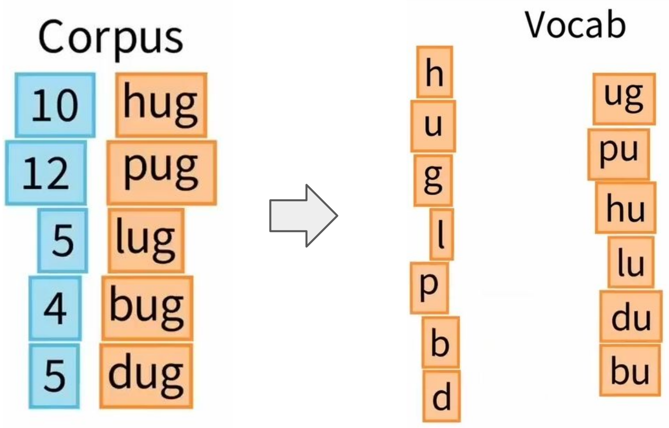


1st iteration of EM

The E step. Calculate loss.



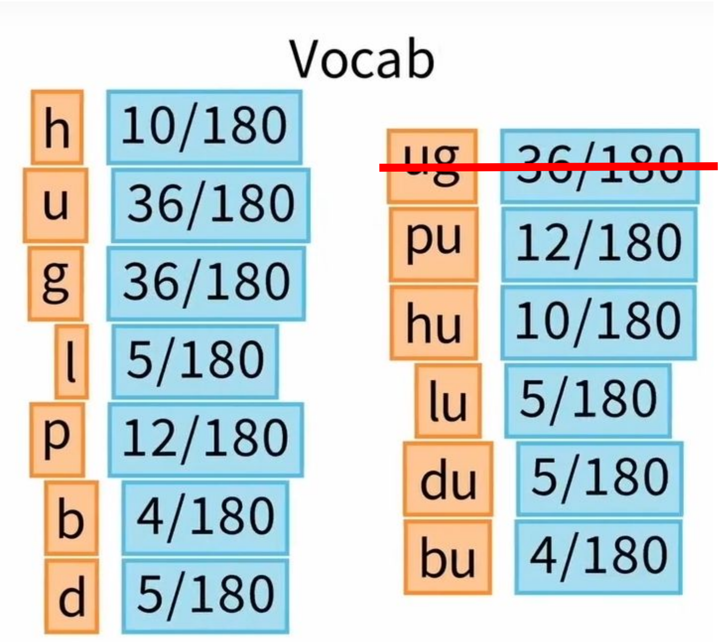
Unigram - training (cont.)



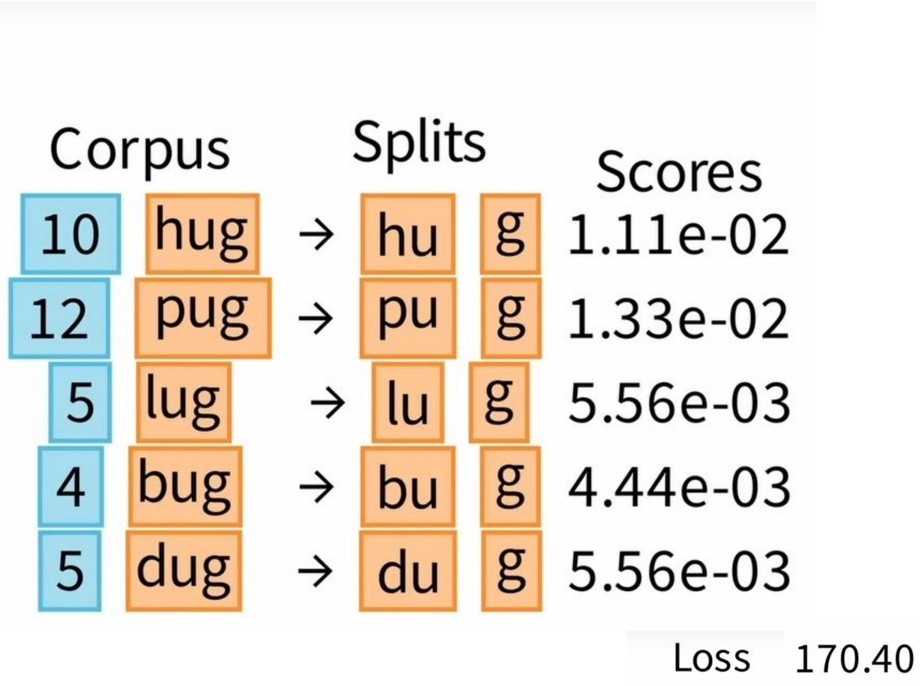
1st iteration of EM

The M step. Remove the tokens that least impacts the loss (remove p% at a time)

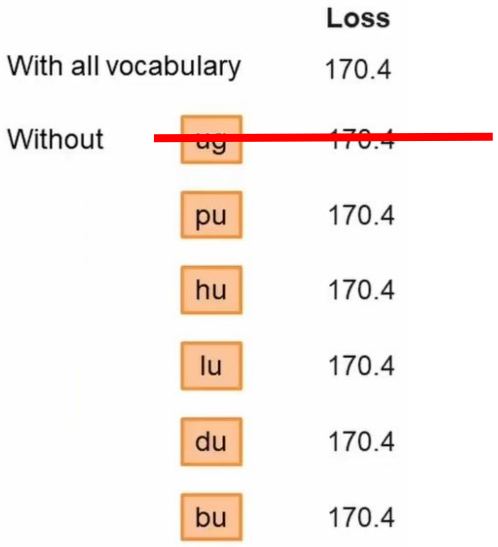
Try removing **ug**



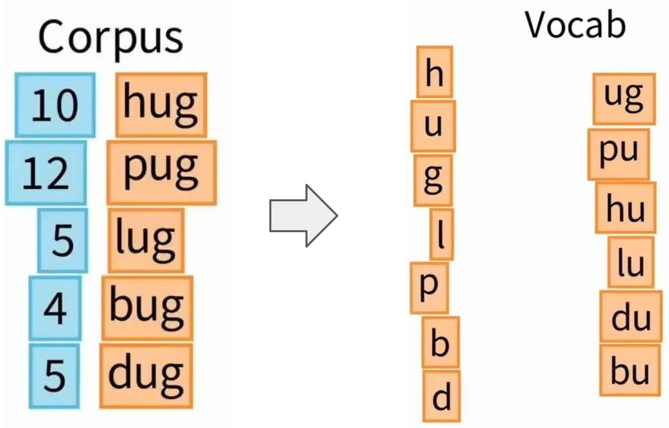
Loss is still the same



Removing any token results in the same loss so choose randomly again

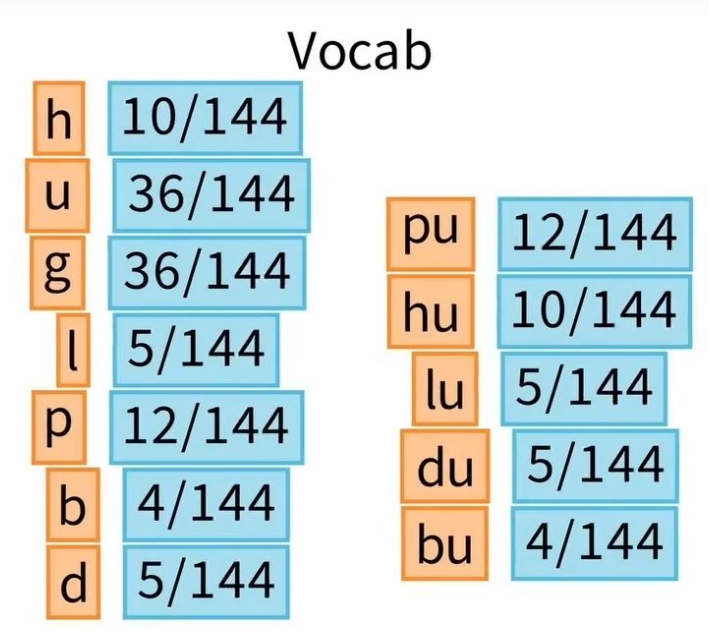


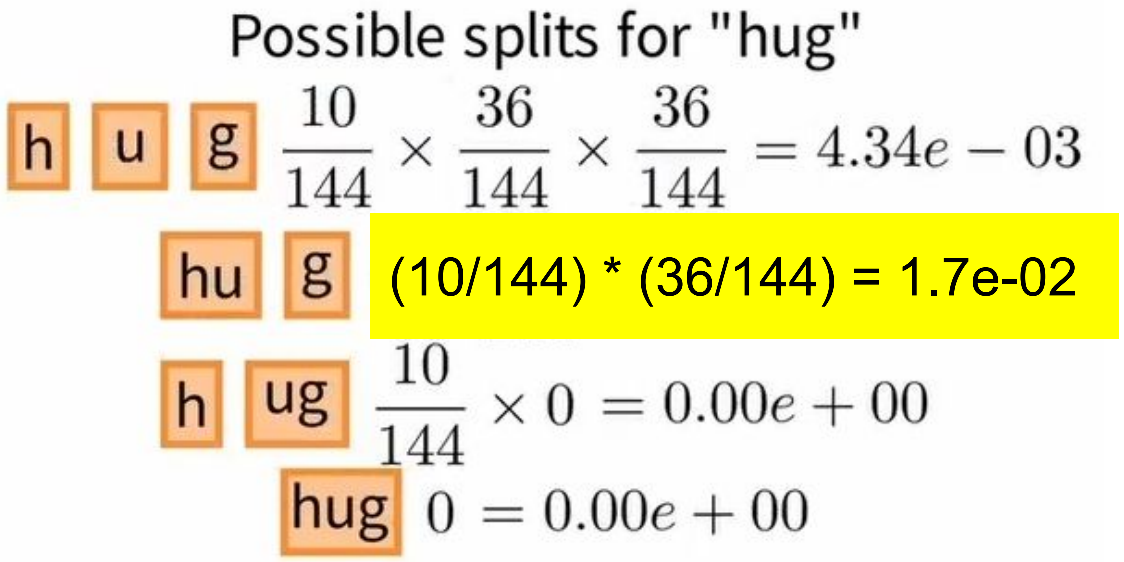
Unigram - training (cont.)



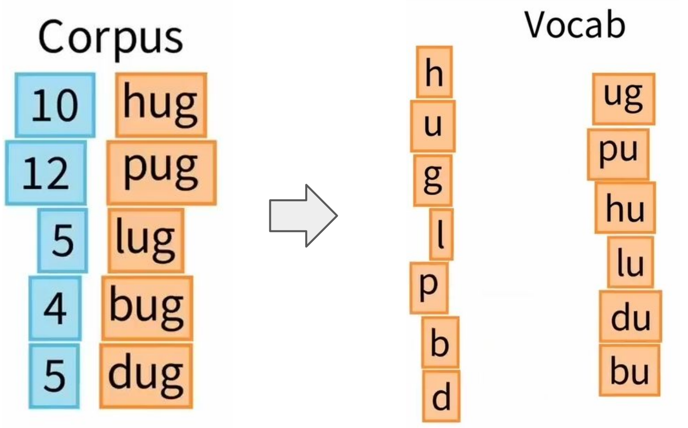
2nd iteration of EM

The E step. Select the split for each word in the corpus with highest prob.



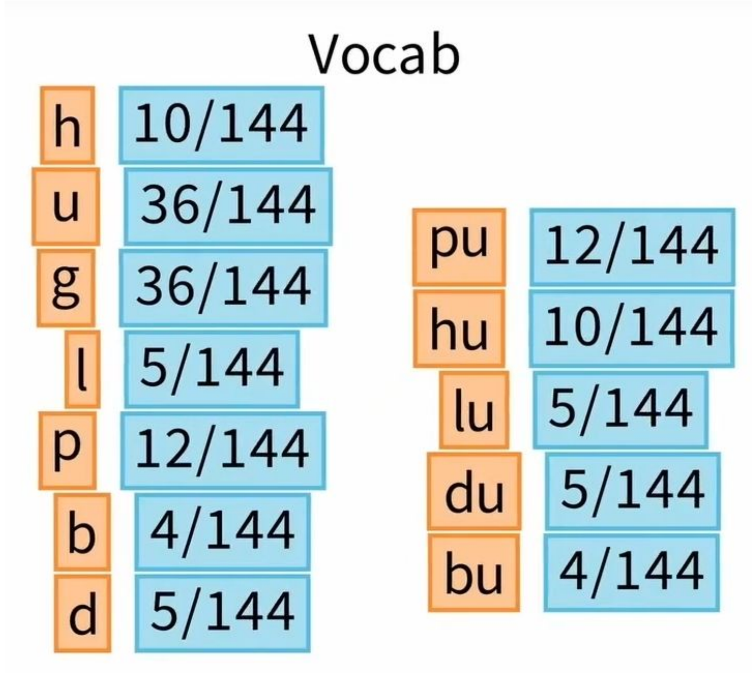


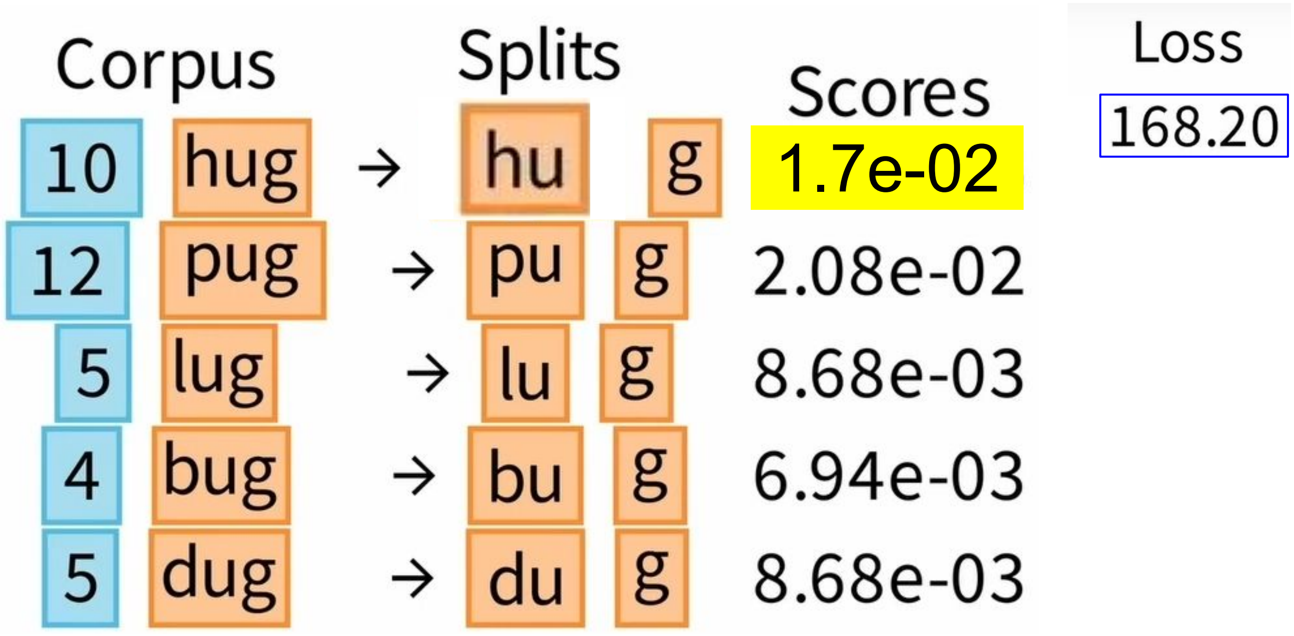
Unigram - training (cont.)



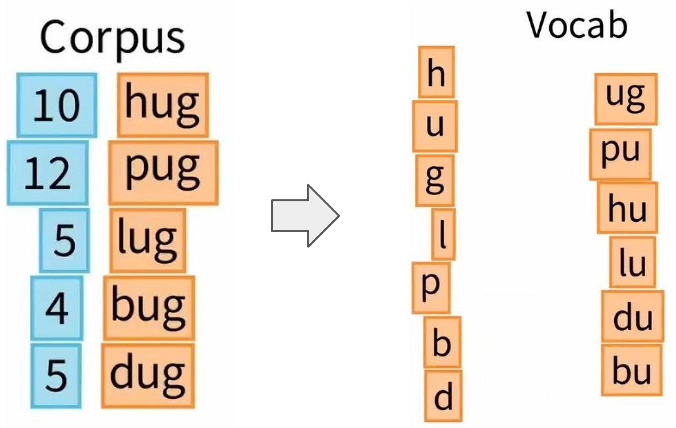
2nd iteration of EM

The E step. Calculate loss.



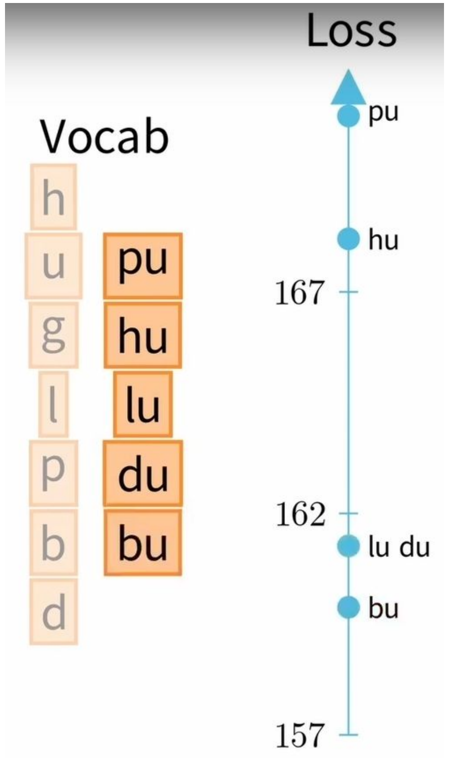


Unigram - training (cont.)



2nd iteration of EM

The M step. Remove the tokens that least impacts the loss (remove p% at a time)



Removing "bu" gives the least loss so "bu" is removed

SentencePiece

SentencePiece

- SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing (Kudo et al., 2018)
- It aims to solve 2 issues.
- Issue 1: Which one should be the correct detokenization?
  - Tokenize("World.") == Tokenize("World_.")
- Issue 2: End-to-End to avoid the need of language-specific tokenization.

WangchanBERTa We name our pretrained language models according to their architectures, tokenizers and the datasets on which they are trained on. The models can be found on HuggingFace[12].

	Architecture	Dataset	Tokenizer
wangchanberta-base-wiki-spm	RoBERTa-base	Wikipedia-only	SentencePiece
wangchanberta-base-wiki-newmm	RoBERTa-base	Wikipedia-only	word (newmm)
wangchanberta-base-wiki-ssg	RoBERTa-base	Wikipedia-only	syllable (ssg)
wangchanberta-base-wiki-sefr	RoBERTa-base	Wikipedia-only	SEFR
wangchanberta-base-att-spm-uncased	RoBERTa-base	Assorted Thai Texts	SentencePiece

Table 3: WangchanBERTa model names

SentencePiece (cont.)

Introduces "_ (U+2581)" to preserve whitespace for detokenization

For the sake of clarity, SentencePiece first escapes the whitespace with a meta symbol _ (U+2581), and tokenizes the input into an arbitrary subword sequence, for example:

- **Raw text:** Hello_world.
- **Tokenized:** [Hello] [_wor] [ld] [.]

As the whitespace is preserved in the tokenized text, we can detokenize the tokens without any ambiguities with the following Python code.

```
detok = ''.join(tokens).replace('_', ' ')
```

Feature	SentencePiece
Supported algorithm	BPE, unigram, char, word

https://github.com/google/sentencepiece

https://aclanthology.org/D18-2012.pdf