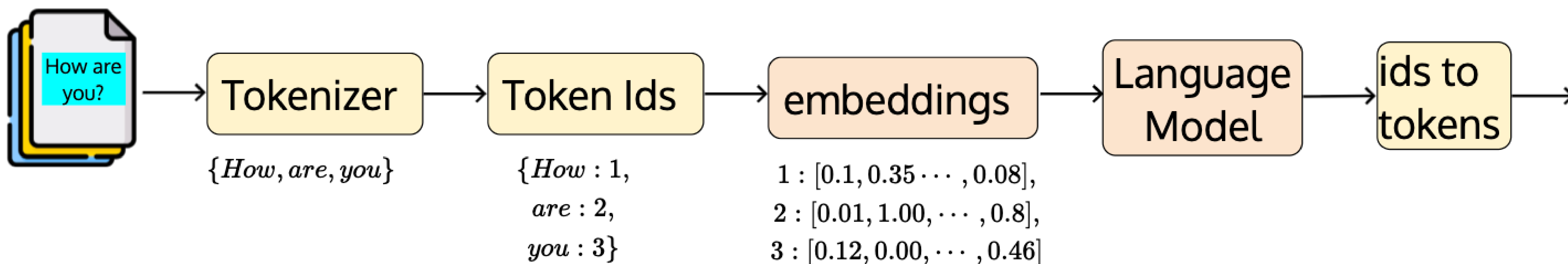
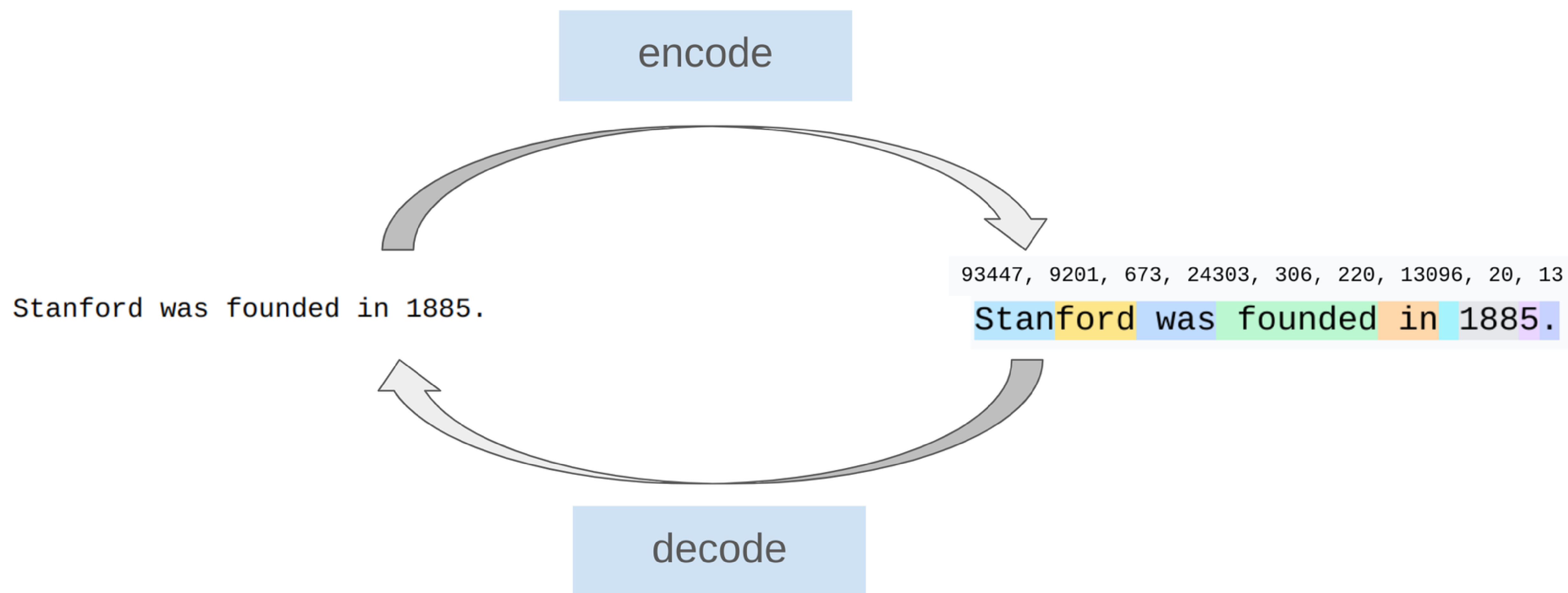
Tokenizers

COL772
Natural Language Processing
Kausik Hira

Slides by Mitesh Khapra, Tanmoy Chakraborty, Arun Prakash,
Afshine Amidi and Shervine Amidi

Why do we need tokenizers?





Tokenizer: strings <-> tokens (indices)

Tokenization

A cute teddy bear is reading.

A	cute	teddy bear	is	reading	.	Random

A	cute	teddy	bear	is	reading	.	Word based

A	cute	ted	##dy	bear	is	read	##ing	.	Subword

A	_	c	u	t	e	_	t	e	d	d	y	_	b	e	a	r	_	i	s	_	r	e	a	d	i	n	g	.	char-acter

Word – level tokenizer

Pros:

1. Simple

Cons:

1. Huge vocabulary
2. Very large no of tokens for each input
3. Computationally expensive

Character level tokenizer

Pros:

1. Very low chances of OOV.
2. Handles rare words, typos.

Cons:

1. Very long sequences - memory-inefficient larger embedding size, and computationally harder to compute attention.
2. Embeddings aren't interpretable.
3. Doesn't leverage knowledge of root.

Byte-level tokenizer

Pros:

1. Limited no of unicodes
2. Vocabulary length within bounds
3. No risk of OOV.

Cons:

1. Embeddings aren't interpretable.
2. Doesn't leverage knowledge of root

Sub-word level tokenizer

Pros:

1. Considers root of the word.
2. Leverages common prefixes and suffixes.

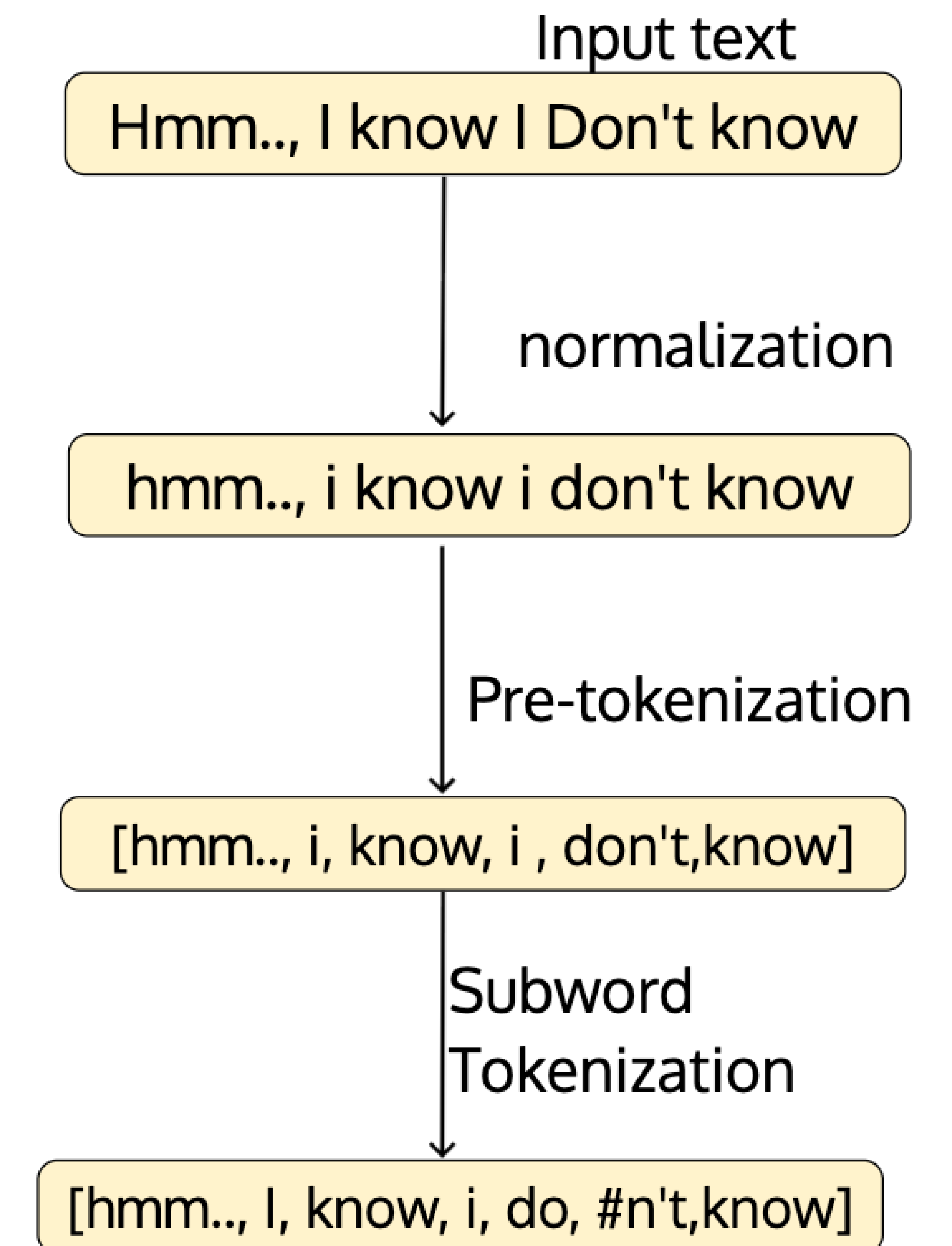
Cons:

1. Risk of OOV, though less than word-level.

Sub-word level tokenizer

The 3 most common sub-word tokenizer algorithms are:

1. Byte - Pair Encoding
1. Wordpiece Tokenizer
1. Sentencepiece/Unigram Tokenizer



Byte-Pair Encoding (BPE)

Byte Pair Encoding (BPE)

- Input: corpus of text data.

- Pre-tokenization: splitting text into words. Pre-tokenization can involve breaking the text at spaces, punctuation, or using more complex rules.

- Base vocabulary: all unique characters.

- Each word in the corpus is then represented as a sequence of symbols from this base vocabulary. "hello" as ['h', 'e', 'l', 'l', 'o'].

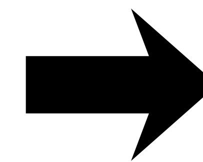
Byte Pair Encoding (BPE)

- Pair Merging:
  - Bigram Counts: count frequency of adjacent symbol pairs in the list of unique words.
  - Merging: The most frequent bigram is then merged into a new symbol, and the words in the corpus are updated to reflect this merge.
  - continue iteratively until desired vocab size is reached.

Example

- Split the corpus on \s (Pretokenize) and note freq.
- Split each words into characters
- Form Base vocabulary: a, b, c, g, s, t

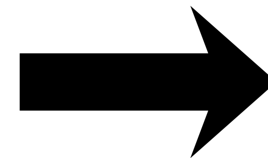
Word	Freq
cat	10
bat	5
bag	12
tag	4
cats	5



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

Example

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

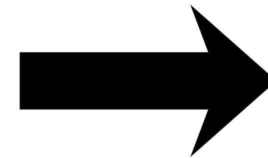


Word	Freq
c,a	15
a,t	20
b,a	17
t,a,g	4
c,a,t,s	5

Example

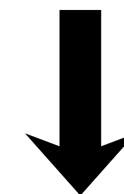
- [a,t] ⟶ [at]

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



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

[a,g] ⟶ [ag]



Example

Word	Freq
c,at	10
b,at	5
b,ag	12
t,ag	4
c,at,s	5

[c,at]
↓
[cat]

Word	Freq
cat	10
b,at	5
b,ag	12
t,ag	4
cat,s	5

- All the merged rules are stored sequentially.

Tokenization

- To tokenize a text, we run each merge learned from the training data in a greedy manner, successively in the order we learned them.
- If the word being tokenized includes a character that wasn’t present in training corpus, that character will be converted to the unknown token (<UNK>).
- GPT-2 and RoBERTa uses bytes as the base vocabulary (size 256) and then applies BPE.
- In practice, it is common to add a special end-of-word symbol like “__”.

WordPiece Algorithm

Wordpiece Algorithm

- 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.

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

Wordpiece 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 split as:

  token → t ##o ##k ##e ##n

Wordpiece Algorithm

- 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.

Example

Word count

Word	Frequency
'k n o w i n g'	3
't h e'	1
'n a m e'	1
'o f'	1
's o m e t h i n g'	2
'i s'	1
'd i f f e r e n t'	1
'f r o m'	1
's o m e t h i n g .'	1
'a b o u t'	1
'e v e r y t h i n g'	1
"i s n ' t'	1
'b a d'	1

ignoring the prefix

Word	Frequency	Freq of 1st element	Freq of 2nd element	score
('k', 'n')	3	'k':3	'n':13	0.076
('n', 'o')	3	'n':13	'o':9	0.02
('o', 'w')	3	'o':9	'w':3	0.111
('w', 'i')	3			
('i', 'n')	7	'i':10	'n':13	0.05
('n', 'g')	7	'n':13	'g':7	0.076
('g', '.')	1			
('t', 'h')	5	't':8	'h':5	0.125
('h', 'e')	1			
('e', '</w>')	2			
⋮	⋮			
('a', 'd')	1	'a':3	'd':2	0.16

Now, merging is based on the score of each byte pair.

$$score = \frac{count(\alpha,\beta)}{count(\alpha)count(\beta)}$$

Merge the pair with highest score

Tokenization

- We are breaking ties where denominator is less.

- This signifies:
  - preferring pairs which occur more wrt to their individual occurrence, in comparison to others.
  - Also, ties will occur much less with WordPiece.

- Not only for tie-breaking, but preferring pairs which are relatively more likely to be seen.

Tokenization

- Tokenization in WordPiece differs from BPE.
- 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.

Tokenization

- For example, let’s take the word – fused.

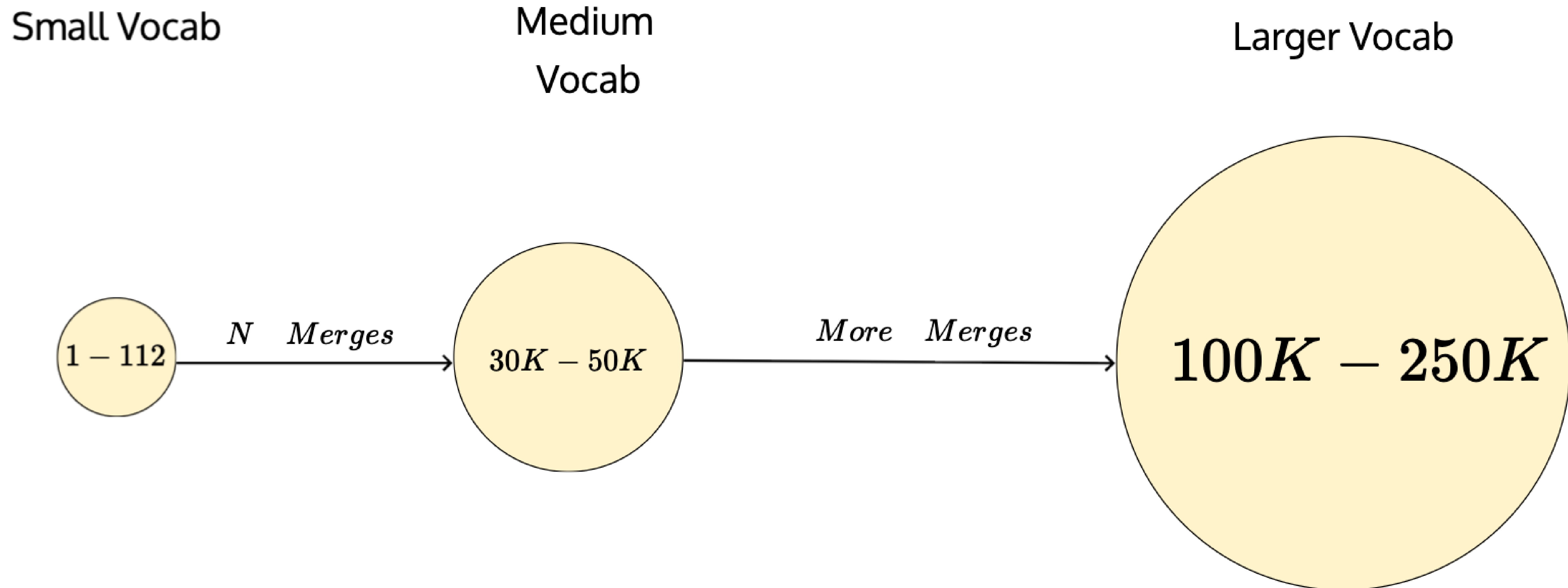
- Vocabulary :
  - ##d, ##e, ##f, ##g, ##i, ##l, ##n, ##o, ##r, ##s, ##u, ##w,
  - f, s, su,
  - ##er, ##ed

- fused ⟶ [f, ##u, ##s, ##e, ##d]

Tokenization

- Start with first character, and keep on matching with the longest subword present in the vocabulary.
- fused ⟶ [f, ##u, ##s, ##e, ##d]
  - f is present
  - no bigram of ‘us’, however, unigrams of ##u, ##s, and bigram ##ed present. Match found.
- fuzed ⟶[f, ##u, ##z, ##e, ##d] (##z not present)
  - For BPE: [f, ##u, <UNK>, ##e, ##d]
  - For wordpiece: <UNK>

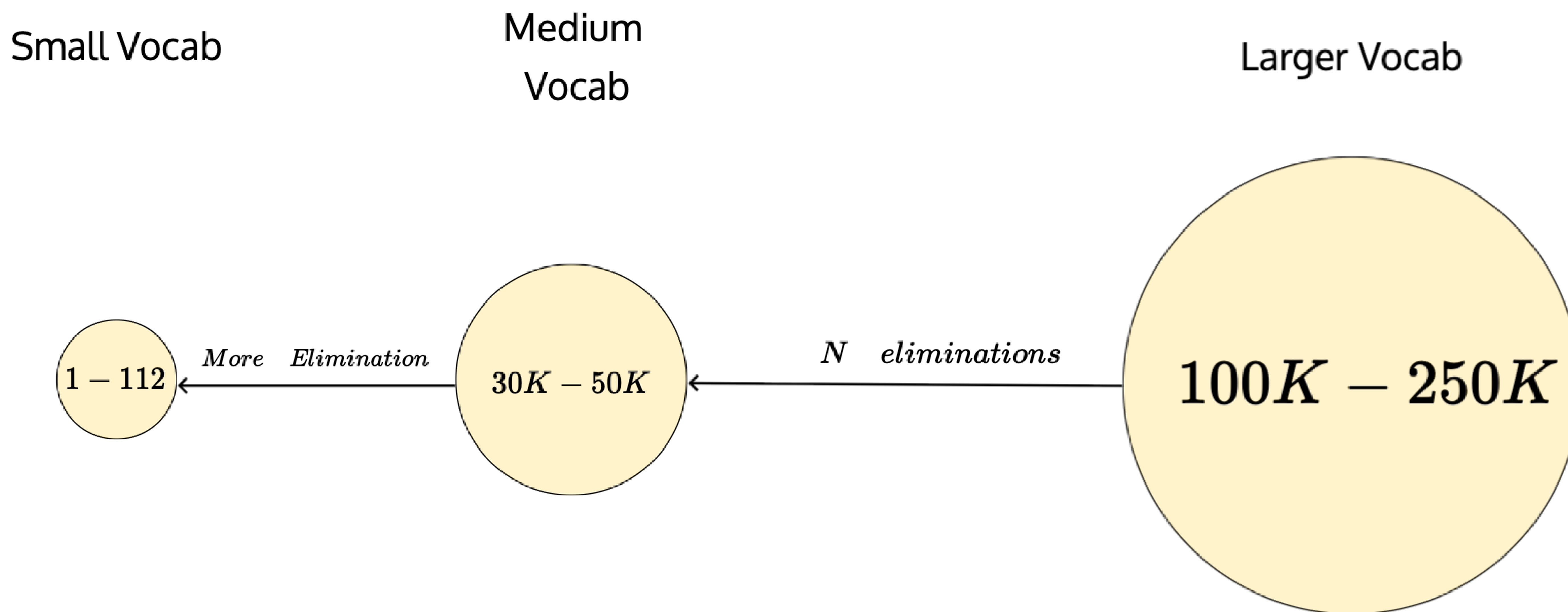
Appending the vocabulary



We start with a character level vocab and keep merging until a desired vocabulary size is reached

Shrinking the vocabulary

Well, we can do the reverse as well.



We start with word level vocab and keep eliminating words until a desired vocabulary size is reached

That's what we see next.

SentencePiece Algorithm

Senetencepiece / Unigram Algorithm

- Observation: A single word or sentence can be divided into different words using the same vocabulary.
- Problem: BPE operates using a deterministic and greedy approach.
- Solution: Introduce multiple subword candidates.
- The algorithm is based on Expectation-Maximization (EM) algorithm.

Senetencepiece / Unigram Algorithm

A given word can have numerous subwords.

For instance, the word "X=hello" can be segmented in multiple ways (by BPE) even with the **same vocabulary**

$\mathcal{V}$={h,e,l,l,o,he,el,lo,ll,hell}

$$\mathbf{x_1} = he, ll, o \quad \mathbf{x_2} = h, el, lo$$

$$\mathbf{x_3} = he, l, lo \quad \mathbf{x_4} = hell, o$$

however, following the merge rule, BPE outputs : he, l, lo

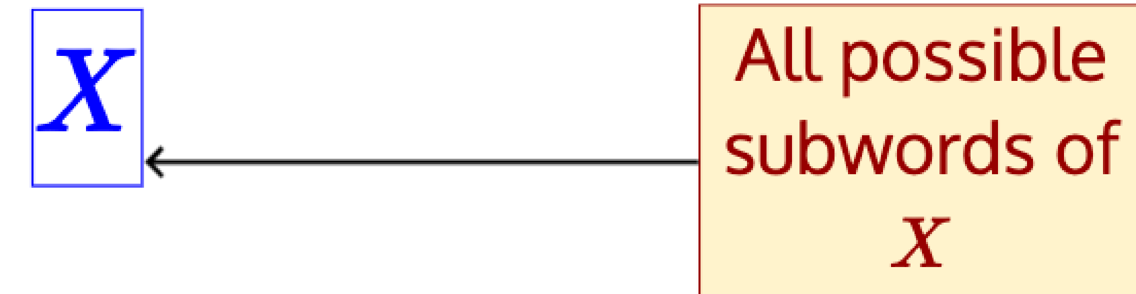
On the other hand, if

$\mathcal{V}$={h,e,l,l,o,el,he,lo,ll,hell}

then BPE outputs: h, el, lo

Therefoe, we say BPE is greedy and deterministic (we can use BPE-Dropout [Ref] to make it stochastic)

The probabilistic approach is to find the subword sequence $\mathbf{x}^* \in \{\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_k}\}$ that maximizes the likelihood of the word X



observed

hidden

The word X in sentencepiece means a sequence of characters or words (without spaces)

Therefore, it can be applied to languages (like Chinese and Japanese) that do not use any word delimiters in a sentence.

Senetencepiece / Unigram Algorithm

Let $\mathbf{x}$ denote a subword sequence of length n.

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

then the probability of the subword sequence (with unigram LM) is simply

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i)$$

$$\sum_{x \in \mathcal{V}} p(x) = 1$$

the objective is to find the subword sequence for the input sequence X (from all possible segmentation candidates of $S(X)$) that maximizes the (log) likelihood of the sequence

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in S(X)} P(\mathbf{x})$$

We can use Viterbi decoding to find $\mathbf{x}^*$.

Then, for all the sequences in the dataset D, we define the likelihood function as

$$\mathcal{L} = \sum_{s=1}^{|D|} \log(P(X^s))$$

$$= \sum_{s=1}^{|D|} \log \Big(\sum_{\mathbf{x} \in S(X^s)} P(\mathbf{x}) \Big)$$

Recall that the subwords $p(x_i)$ are hidden (latent) variables.

Therefore, given the vocabulary $\mathcal{V}$, Expectation-Maximization (EM) algorithm could be used to maximize the likelihood

Senetencepiece / Unigram Algorithm

Let X = "knowing" and a few segmentation candidates be $S(X) = \{‘k, now, ing‘, ‘know, ing‘, ‘knowing‘\}$

Given the unigram language model we can calculate the probabilities of the segments as follows

$$p(\mathbf{x_1} = k, now, ing) = p(k)p(now)p(ing)$$

$$= \frac{3}{16} \times \frac{3}{16} \times \frac{7}{16} = \frac{63}{4096}$$

$$p(\mathbf{x_2} = know, ing) = p(know)p(ing) = \frac{21}{256} = \frac{336}{4096}$$

$$p(\mathbf{x_3} = knowing) = p(knowing) = \frac{3}{16} = \frac{768}{4096}$$

$$\mathbf{x}^* = \arg\max_{\mathbf{x} \in S(X)} P(\mathbf{x}) = \mathbf{x_3}$$

In practice, we use Viterbi decoding to find $\mathbf{x}^*$ instead of enumerating all possible segments

Word	Frequency
'knowing'	3
'the'	1
'name'	1
'of'	1
'something'	2
'is'	1
'different'	1
'from'	1
'something.'	1
'about'	1
'everything'	1
"isn't"	1
'bad'	1

Senetencepiece / Unigram Algorithm

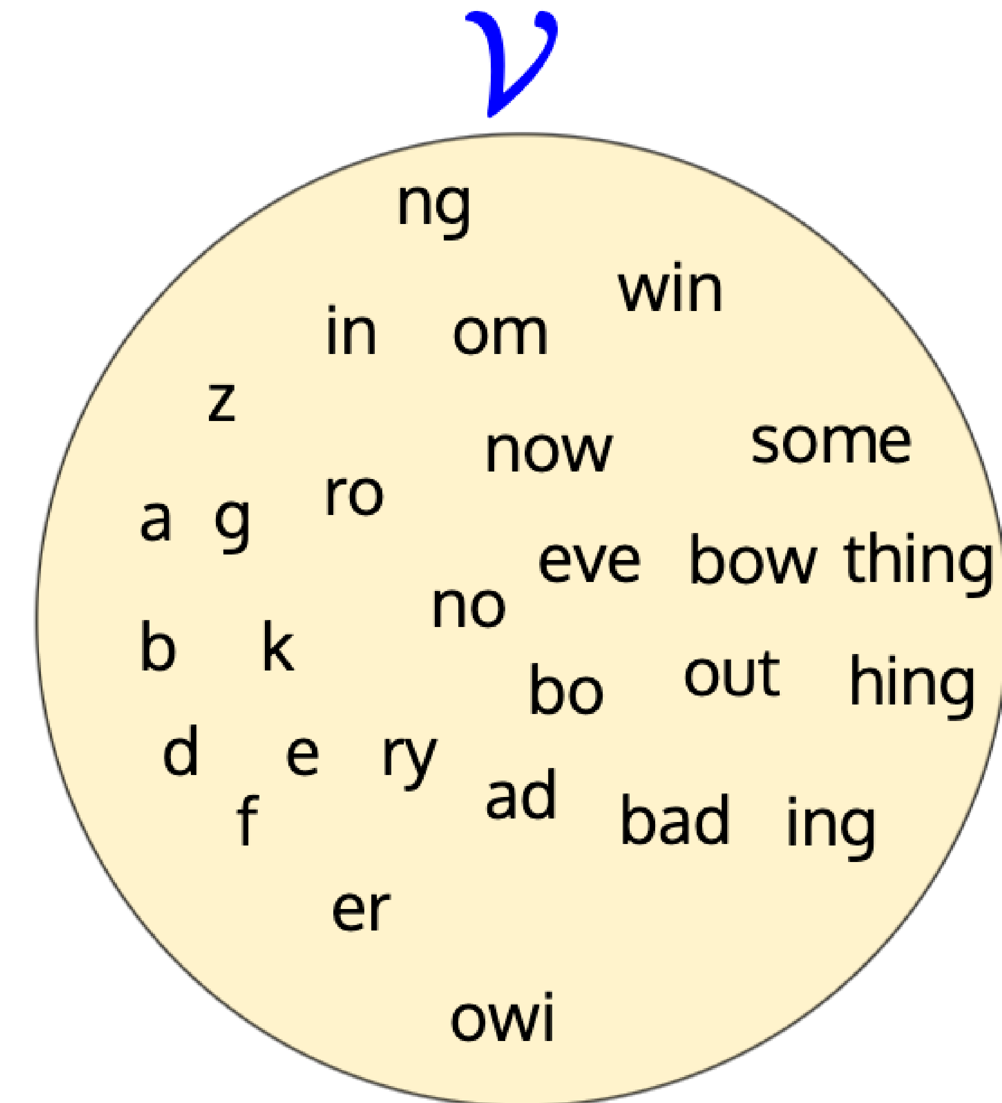
Algorithm

Set the desired vocabulary size

1. Construct a reasonably large seed vocabulary using BPE or Extended Suffix Array algorithm.
2. **E-Step:**

   Estimate the probability for every token in the given vocabulary using frequency counts in the training corpus
3. **M-Step:**

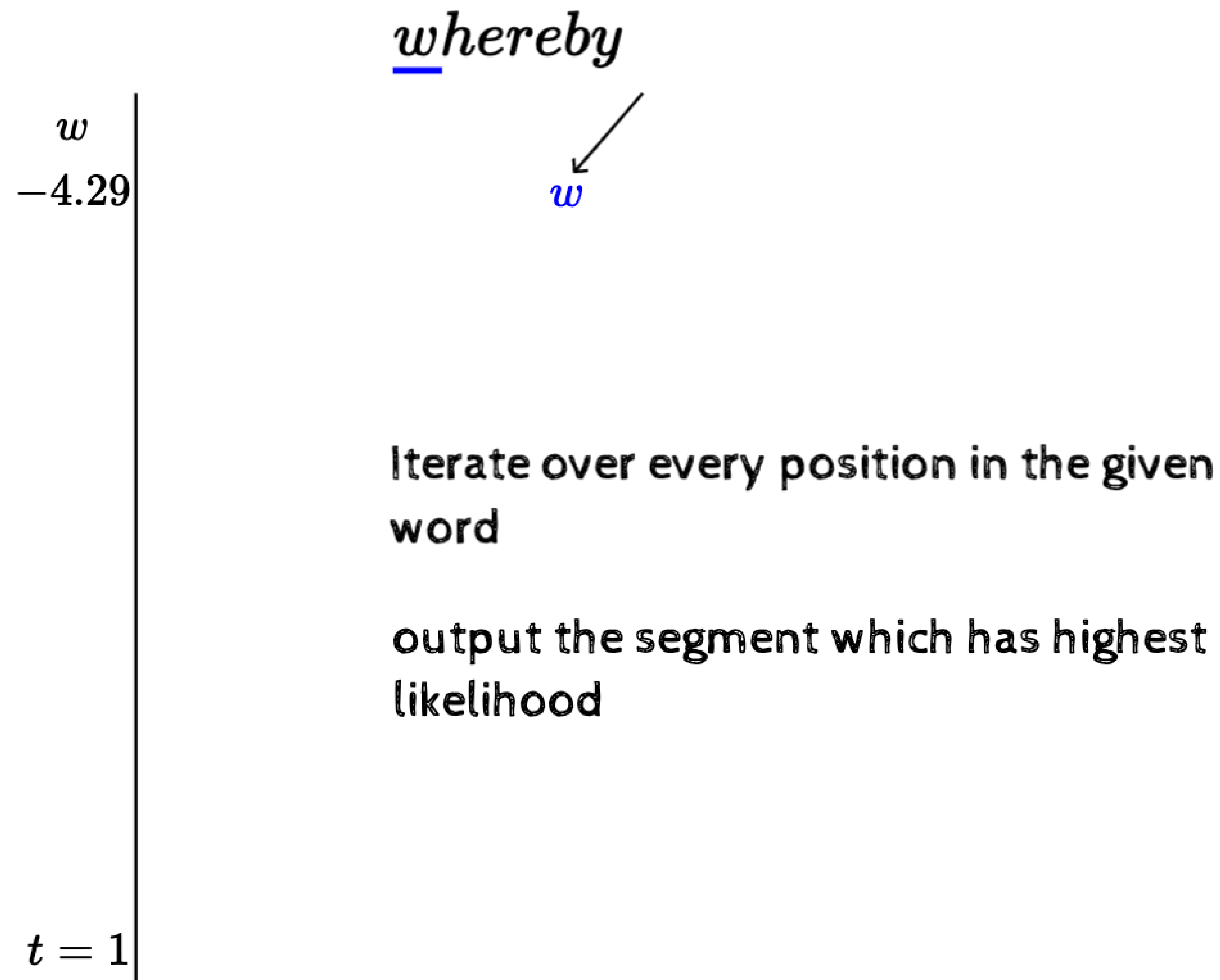
   Use Viterbi algorithm to segment the corpus and return optimal segments that maximizes the (log) likelihood.
4. Compute the likelihood for each new subword from optimal segments
5. Shrink the vocabulary size by removing top $x\%$ of subwords that have the smallest likelihood.
6. Repeat step 2 to 5 until desired vocabulary size is reached



Let us consider segmenting the word "whereby" using Viterbi decoding

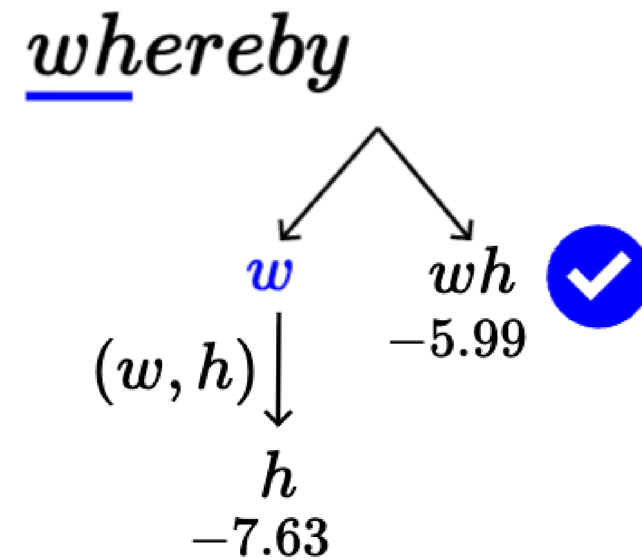
Senetencepiece / Unigram Algorithm

Forward algorithm



Token	log(p(x))
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

Senetencepiece / Unigram Algorithm

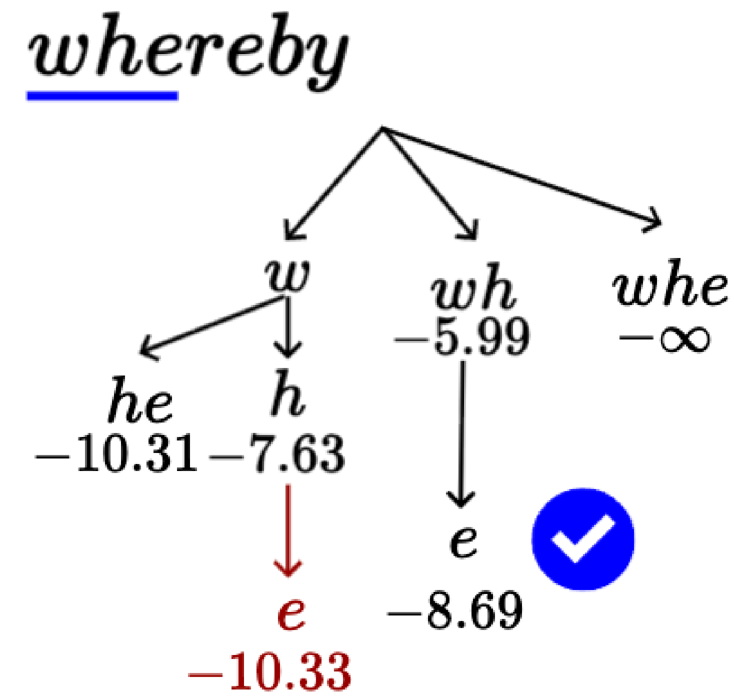


At this position, the posible segmentations of the slice "wh" are (w,h) and (wh)

Compute the log-likelihood for both and output the best one.

Token	log(p(x))
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

Senetencepiece / Unigram Algorithm

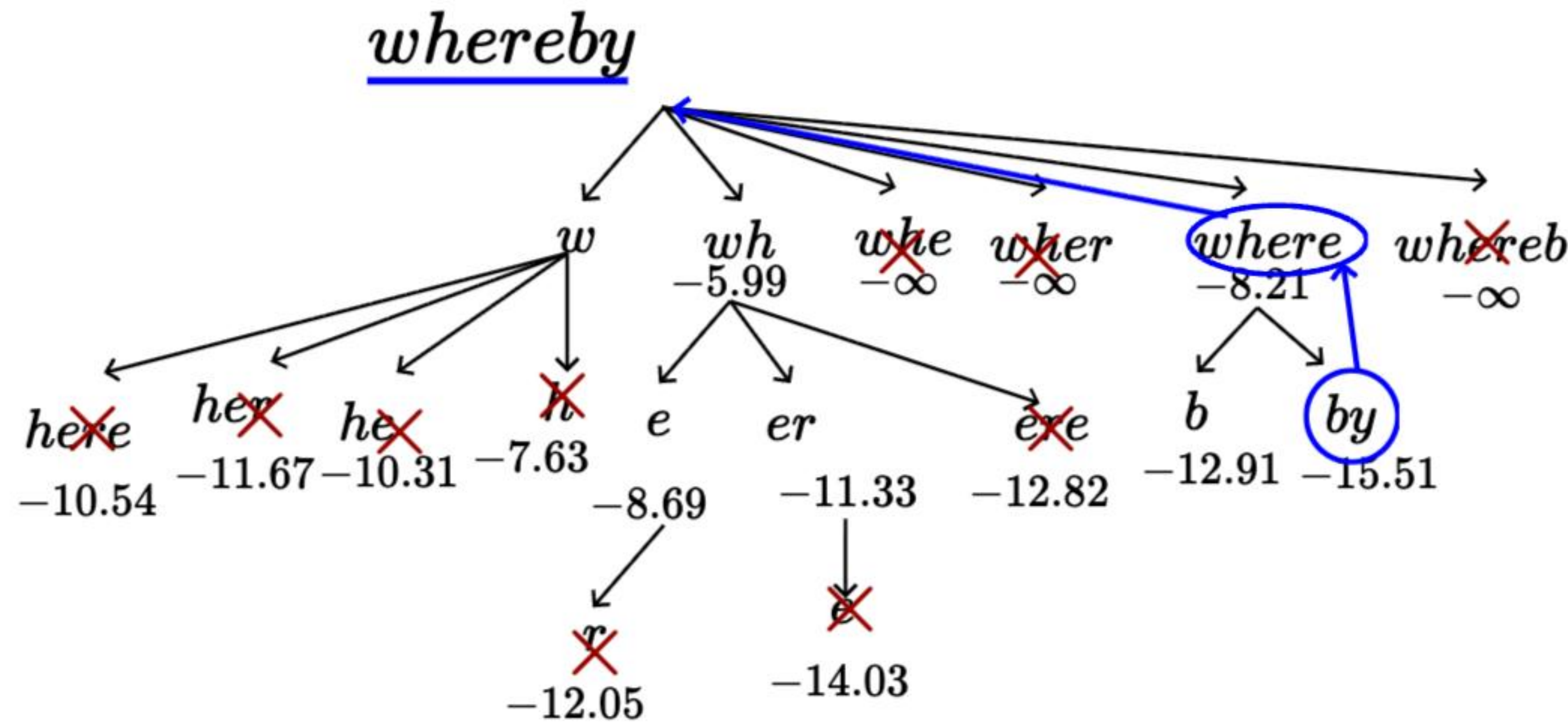


We do not need to compute the likelihood of (w,h,e) as we already ruled out (w,h) to (wh). We display it for completeness

Of these, (wh,e) is the best segmentation that maximizes the likelihood.

Token	log(p(x))
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

Senetencepiece / Unigram Algorithm



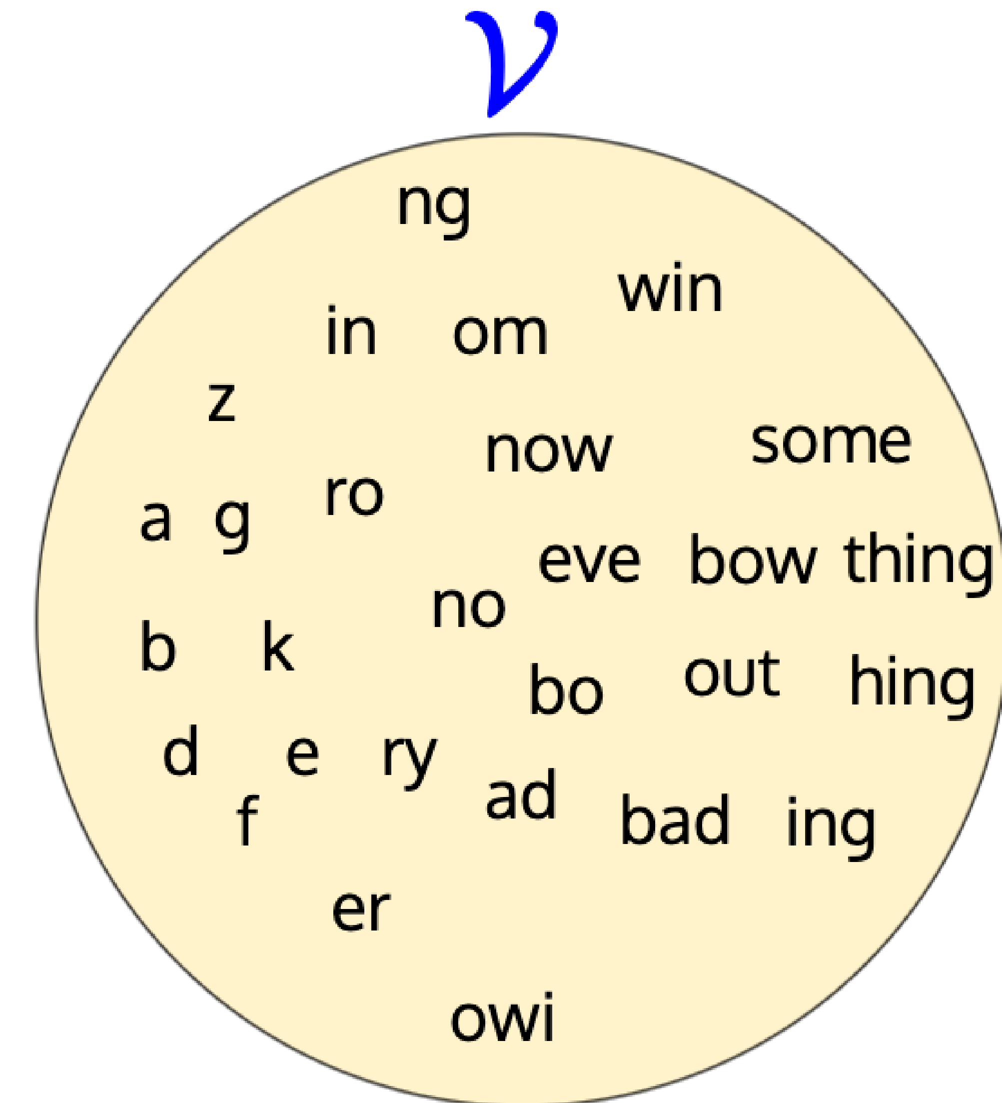
Token	log(p(x))
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

Senetencepiece / Unigram Algorithm

Algorithm

Set the desired vocabulary size

1. Construct a reasonably large seed vocabulary using BPE or Extended Suffix Array algorithm.
2. **E-Step:**
   Estimate the probability for every token in the given vocabulary using frequency counts in the training corpus
3. **M-Step:**
   Use Viterbi algorithm to segment the corpus and return optimal segments that maximizes the (log) likelihood.
4. Compute the likelihood for each new subword from optimal segments
5. Shrink the vocabulary size by removing top $x\%$ of subwords that have the smallest likelihood.
6. Repeat step 2 to 5 until desired vocabulary size is reached



Let us consider segmenting the word "whereby" using Viterbi decoding

Does creating vocabulary and tokenizers really help for domain-specific modelling?

Yes, it does.

NLP Model	Named Entity Recognition	Relation Classification	Event Argument Extraction	Paragraph Classification	Synthesis Action Retrieval	Sentence Classification	Slot Filling	Overall (All Tasks)
MatSciBERT (Gupta et al., 2022)	$0.707_{\pm 0.076}$ $0.470_{\pm 0.092}$	$0.791_{\pm 0.046}$ $0.507_{\pm 0.073}$	$0.436_{\pm 0.066}$ $0.251_{\pm 0.075}$	$0.719_{\pm 0.116}$ $0.623_{\pm 0.183}$	$0.692_{\pm 0.179}$ $0.484_{\pm 0.254}$	$0.914_{\pm 0.008}$ $0.660_{\pm 0.079}$	$0.436_{\pm 0.142}$ $0.194_{\pm 0.062}$	$0.671_{\pm 0.060}$ $0.456_{\pm 0.042}$
MatBERT (Walker et al., 2021)	$0.875_{\pm 0.015}$ $0.630_{\pm 0.047}$	$0.804_{\pm 0.071}$ $0.513_{\pm 0.138}$	$0.451_{\pm 0.091}$ $0.288_{\pm 0.066}$	$0.756_{\pm 0.073}$ $0.691_{\pm 0.188}$	$0.717_{\pm 0.040}$ $0.549_{\pm 0.091}$	$0.909_{\pm 0.009}$ $0.614_{\pm 0.134}$	$0.548_{\pm 0.058}$ $0.273_{\pm 0.051}$	$0.722_{\pm 0.023}$ $0.517_{\pm 0.041}$
BatteryBERT (Huang and Cole, 2022)	$0.786_{\pm 0.113}$ $0.472_{\pm 0.150}$	$0.801_{\pm 0.081}$ $0.466_{\pm 0.111}$	$0.457_{\pm 0.024}$ $0.277_{\pm 0.034}$	$0.633_{\pm 0.075}$ $0.610_{\pm 0.046}$	$0.614_{\pm 0.128}$ $0.419_{\pm 0.149}$	$0.912_{\pm 0.015}$ $0.684_{\pm 0.095}$	$0.520_{\pm 0.057}$ $0.224_{\pm 0.073}$	$0.663_{\pm 0.038}$ $0.456_{\pm 0.048}$
SciBERT (Beltagy et al., 2019)	$0.734_{\pm 0.079}$ $0.497_{\pm 0.091}$	$0.819_{\pm 0.067}$ $0.545_{\pm 0.119}$	$0.451_{\pm 0.077}$ $0.276_{\pm 0.080}$	$0.696_{\pm 0.094}$ $0.546_{\pm 0.243}$	$0.701_{\pm 0.138}$ $0.516_{\pm 0.217}$	$0.911_{\pm 0.017}$ $0.617_{\pm 0.143}$	$0.481_{\pm 0.144}$ $0.224_{\pm 0.010}$	$0.685_{\pm 0.056}$ $0.460_{\pm 0.044}$
BERT (Devlin et al., 2018)	$0.657_{\pm 0.077}$ $0.461_{\pm 0.058}$	$0.782_{\pm 0.056}$ $0.494_{\pm 0.061}$	$0.418_{\pm 0.053}$ $0.225_{\pm 0.091}$	$0.665_{\pm 0.057}$ $0.532_{\pm 0.194}$	$0.656_{\pm 0.099}$ $0.515_{\pm 0.067}$	$0.910_{\pm 0.017}$ $0.633_{\pm 0.133}$	$0.520_{\pm 0.019}$ $0.257_{\pm 0.022}$	$0.658_{\pm 0.030}$ $0.439_{\pm 0.021}$

MatSciNLP - Song et. al. 2023 - evaluated on various Material Science (MatSci) tasks.

1. SciBERT & MatSciBERT (tokenized on scientific corpus) beats BERT (tokenized on general corpus).
2. MatBERT (tok on MatSci corpus) beats all.

Tokenizer is one of the essential factors among others.

Thank you

References:

- CS336: Language Modeling from Scratch by Prof. Percy Liang
- Advanced Large Language Models by Prof. Tanmoy Chakraborty
- Introduction to large language models by Prof. Mitesh Khapra
- Stanford CME295 Transformers & LLMs