Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

Department of Computer Science & Engineering

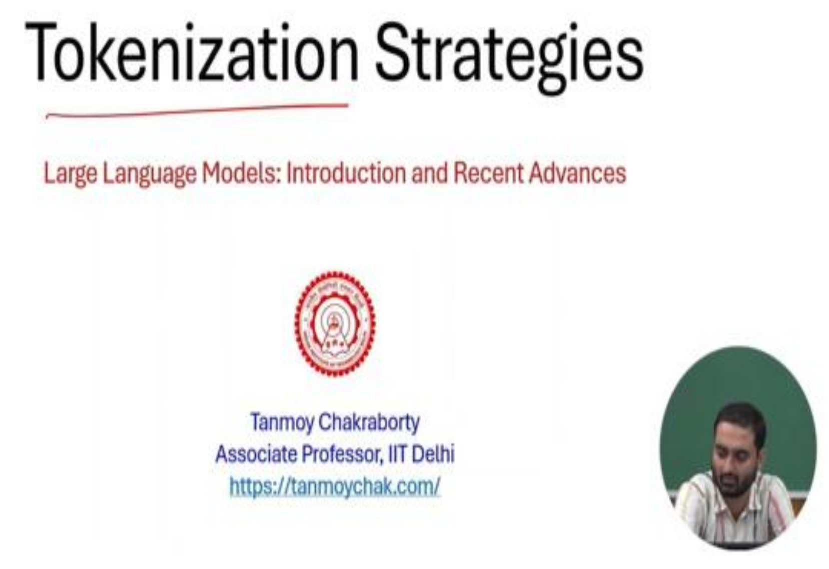
Indian Institute of Technology, Delhi

Lecture 9

Tokenization Strategies

Hello, everyone. We discussed different types of word embedding methods, right? Word2Veg, GloVe, FastText, and so on. So, in each of these techniques, right, if you notice it properly, the first and foremost step is to tokenize the streaming words, right? The tokenization is very important because it actually helps you determine which words are similar in a very high level. So it also helps you understand unknown words very very effectively. So in transformer or before transformer let's say RNN based methods, neural methods and also what to wake club kind of methods, the inputs that we give are basically a sequence of tokens. Now, before neural language models, what people used to do? People used to use different types of delimiters to tokenize a running text.

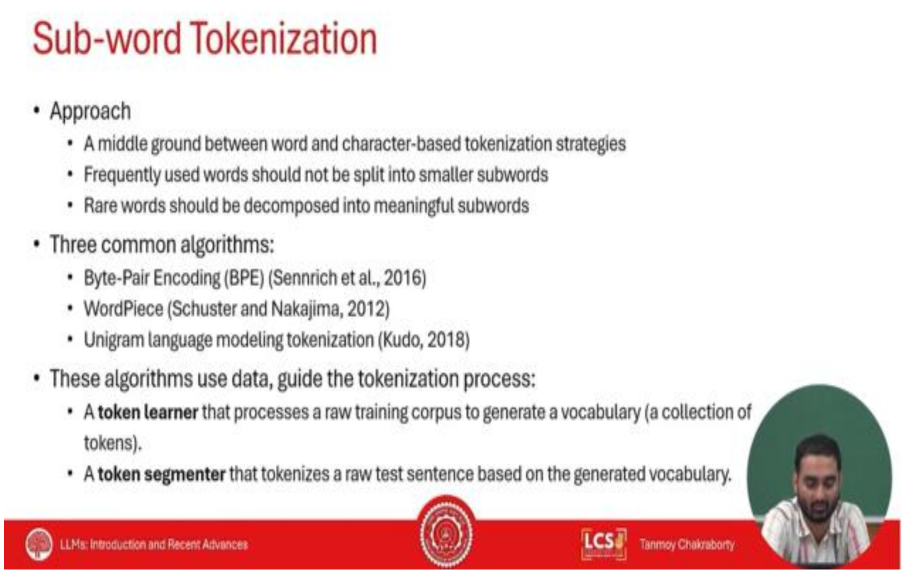
For example, you use space, you use tab, you use, say, full stop, you use new line. Using these delimiters, you can essentially tokenize a sequence of running text. So, but then people realize that you know these delimiters may not be a right approach because well in a normal text when it is okay to split running text into different words based on these delimiters but when it comes to the testing right, when you see an unknown token or you know a token which is difficult to decipher, you may wonder what kind of tokenization we should use.



Tokenization Strategies

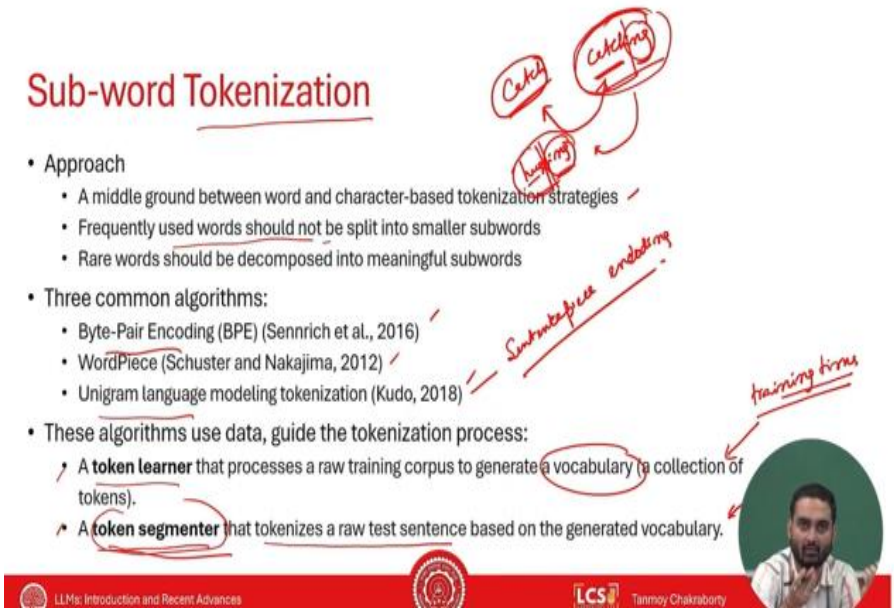
Large Language Models: Introduction and Recent Advances

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



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.



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.

The field of tokenization has been one of the prior areas in large language models in neural machine translation, neural language model in general.

So essentially here we look at three kinds of strategies for tokenization that have been used quite heavily in different kinds of large language models. So the word level tokenization, the strategy that was there before neural machine translation, neural language model, word level tokenization is not good enough because if you think of words, what is word? What is basically a combination of characters? Now you can think of exponential number of you know words possible right if you think of the fixed set of characters. Now with this fixed set of characters you can think of exponential number of words. It may be possible that some of the words are not meaningful words some of the words are meaningful words but you can think of you know a lot of such words right. So only words as a token may not be a right approach whereas A character as a token also may not be a right approach.

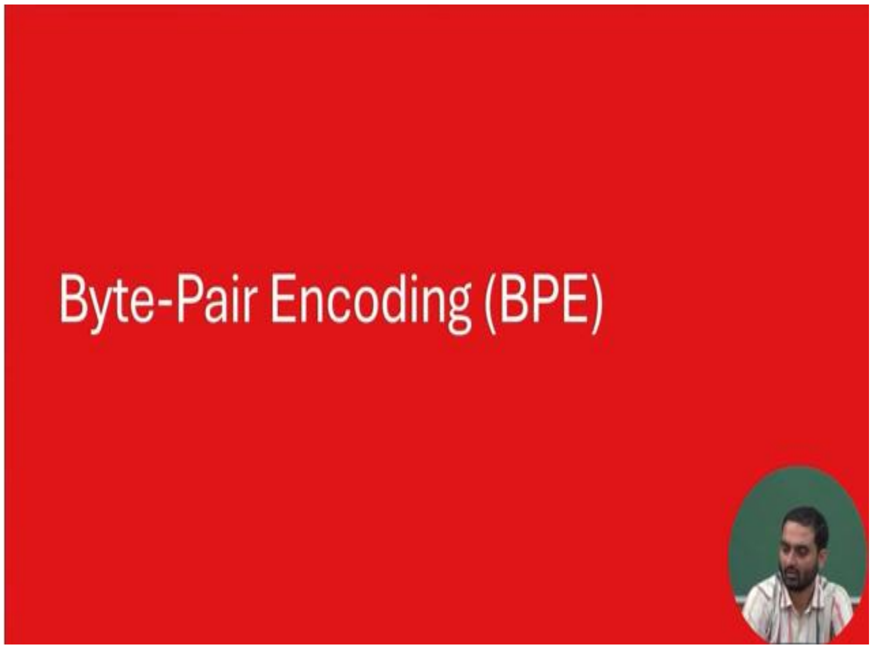
Because if you think of only a character, then there's a fixed set of characters that we can think of, right? 26 characters and maybe some special characters like hash, at the rate, like some unicorns and symbols like that. So that may be a limited set of vocabulary. Whereas if you only consider words, right, that is also a huge set of vocabulary. Now, I mean, you can also imagine new words which are not part of your training set, right? Remember, when you talk about tokenization, this tokenization is performed during the training phase, right? So during the training time, you basically go through the entire corpus and you identify the vocabulary, and that vocabulary will help you parse an unknown token during the test time, okay? So, in between character-level tokenization and word-level tokenization, we can think of sub-word-level tokenization, right? If you remember the word embedding method, we talked about, you know, word-to-weight glove and fast text, right? Fast text is essentially a sub-word-level tokenization. Sub-word-level embedding methods, which basically uses a sub-word-level tokenization strategy, okay? So, Now why this subword label tokenization is even more important? Now let's say if you have a verb, right? Let's say catch. And let's say catch, okay? And catching. Now let's say in your training set the word catch is present, but the word catching is not present. But in a test set, when you pass this word catching, you will not be able to split it properly because you know that in your training set this is not present. But let's say if you think of subword label embedding, and let's say one subword token is catch, right? Then you can say that, okay, in the test token, which is catching, you can split it into two parts, catch and ing.

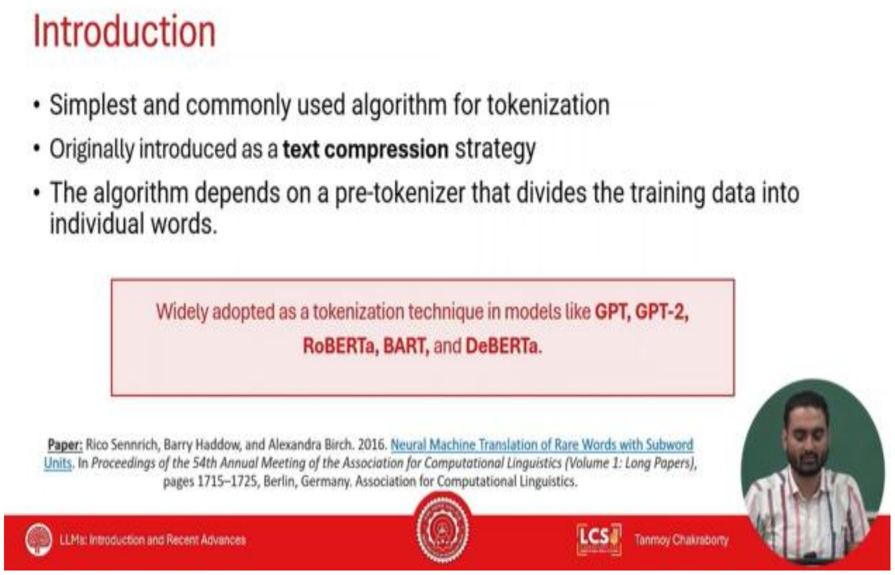
And since the word catch is also present here, maybe the word catch and catching, these are kind of similar, you know, appear in a similar context and so on and so forth, right? Similarly let us say another word let us say hugging right and let us say this is the split hug for example again it is a hypothetical split and ing okay. Now you know that in your test set ing is present. In your training cell let's say hugging is present where you know that ing is also part of hugging. So you can essentially say that okay maybe the word catching and hugging is similar, the word catching and hugging they are similar in some context maybe they are both of them are actually continuous terms right. So therefore, we will basically break a word into subwords, tokens into sub-tokens, and so on and so forth.

And then these subwords basically form tokens, and these tokens are basically fed to the neural model, to the transformer, to the RNN, to the Word2Vec, and any other models that you can think of. OK, so this is, by the way, tokenization is an active research area. So in this lecture we will look at three types of tokenization techniques, byte pair encoding, word piece encoding and unigram language model tokenization which is used in sentence piece encoding. which is used in sentence piece encoding. We will discuss these three kinds of techniques in this lecture and we will see that all modern LLMs like BERT, T5, write GPTs, they essentially use this, I mean, one of these methods or a combination of these methods, okay.

So, in each of these methods, we will see that there are two basic steps. The first step is basically a token learner. The task of this token learner is to learn the tokens which will basically form the vocabulary okay and this token learner is executed during the training time. So during the training time we run this token learner which will essentially uh learn or identify the tokens which are going to be a part of the vocabulary right then During the test time, we use another module called token segmenter. This is during the test time.

Remember in the training time you already identified the vocabulary in the test time based on the vocabulary and maybe a set of rules that you have identified right you can further you can tokenize a raw test sentence into different tokens okay and token segmented is responsible for this okay now For byte pair encoding, you have one type of tokenizer, one type of segmenter. For word piece, you have a different type of tokenizer and segmenter. For unigram language model segmentation, tokenization, you have, again, different types of tokenizer, learner, and segmenter.


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.**

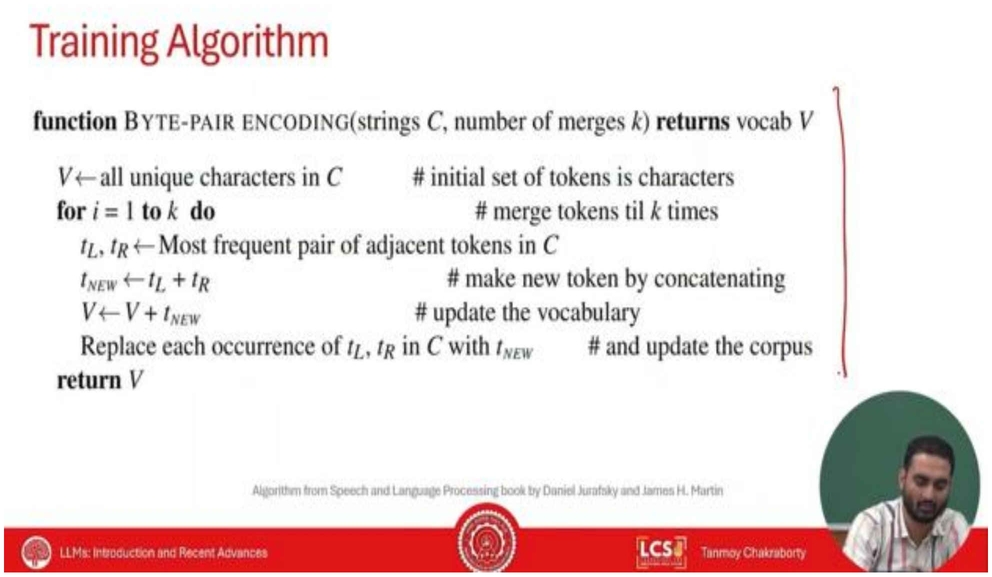
Paper: Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational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

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty

Let's start with byte pair encoding. In short, it is called BPE. Very simple idea. Byteware encoding was originally proposed for text compression techniques. For text compression strategy, it was proposed in long time back and then in 2016 in the ACL paper, they used this technique for neural machine translation, okay. And byte per encoding is used in methods like GPT, GPT-2, Robata, BART, B-A-R-T BART, right. This is a different BART by the way.

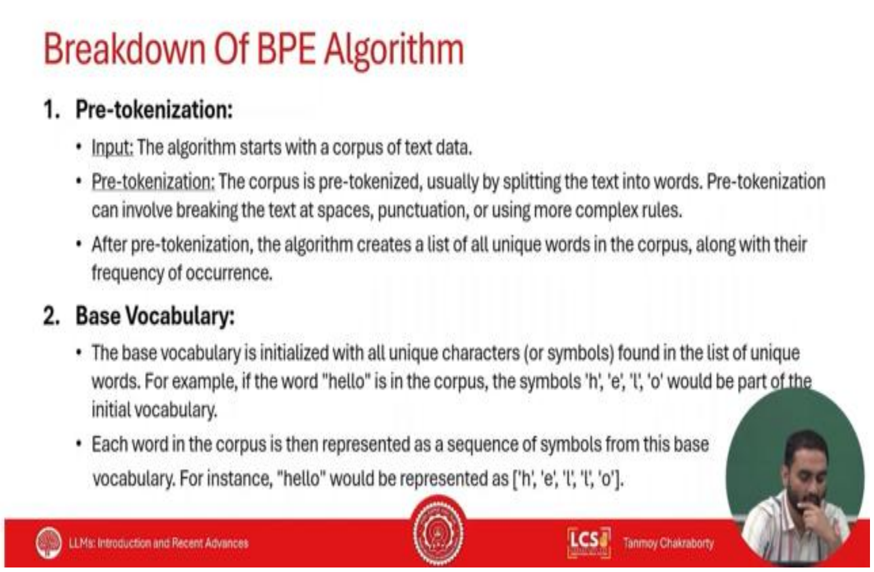
This is an encoder-decoder model, right. We will discuss later and then Robata and so on, okay.


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

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty


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'].

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

So let us look at the algorithm. So this is the algorithm. It is not very important.

You will understand the algorithm once we describe the method. Let us look at the method first. So essentially there are three steps. The first step is that now remember, as I mentioned earlier, you have basically two different modules, right? One is this token learner, other is this token segmenter. The task of the token learner is to identify tokens, okay? To identify tokens, what we do, during the pre-tokenization stage.

Now pre-tokenization stage is a stage which is executed before the token learner is executed. Now during the pre-tokenization stage what we do, we clean the corpus. We are given a training set. We clean the corpus, meaning maybe we remove punctuations. Maybe we remove some special characters.

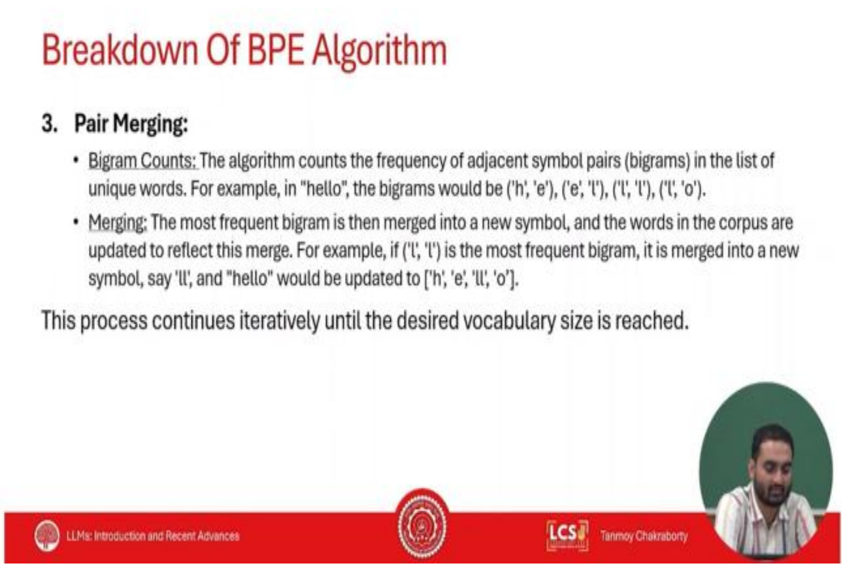
We normalize the text, meaning we convert all the lowercase to uppercase, uppercase to lowercase. Again, these are not very standard steps, by the way. All these things are dependent on the application. But during the pre-tokenization stage, you clean the corpus. you break words, let's say you break a sentence into words, right.

You break sentence into words based on space, punctuation, more complex rules and so on and so forth. You can use traditional NLP techniques for doing that, okay. Now once we do that, once your cleaning process is done, you essentially identify all the words. Right so all the unique words that are there in the corpus right. Now all these unique words basically are tokens types right.

All the types have been identified now okay. Now then what you do once it is done we create a base vocabulary What is a base vocabulary? A base vocabulary is essentially the starting point of your segmentation process, of your tokenization process. In the base vocabulary, what you do, you essentially add all the unique characters present in the tokens that you identified so far. So you have all these words. From the words you identify, you basically extract the characters.

And you add those characters into the vocabulary. For example, one word is hello, H-E-L-L-O. You add the character H, E, L, and O. You add L one time because you want to add only unique character. So H-E-L-O will be added to the vocabulary.

If there is a special symbol, let's say dollar or hash, this will also be added to the vocabulary, and so on and so forth. So now in the best case, what would happen? In the best case, you basically ended up adding all the unique characters present in a language, right? In English, let's say you ended up adding all 26 characters, assuming that all 26 characters appear in the corpus, in your training corpus at least once. Right, so now this is my base vocabulary and what I do, I keep on increasing this vocabulary size with more and more training, okay.



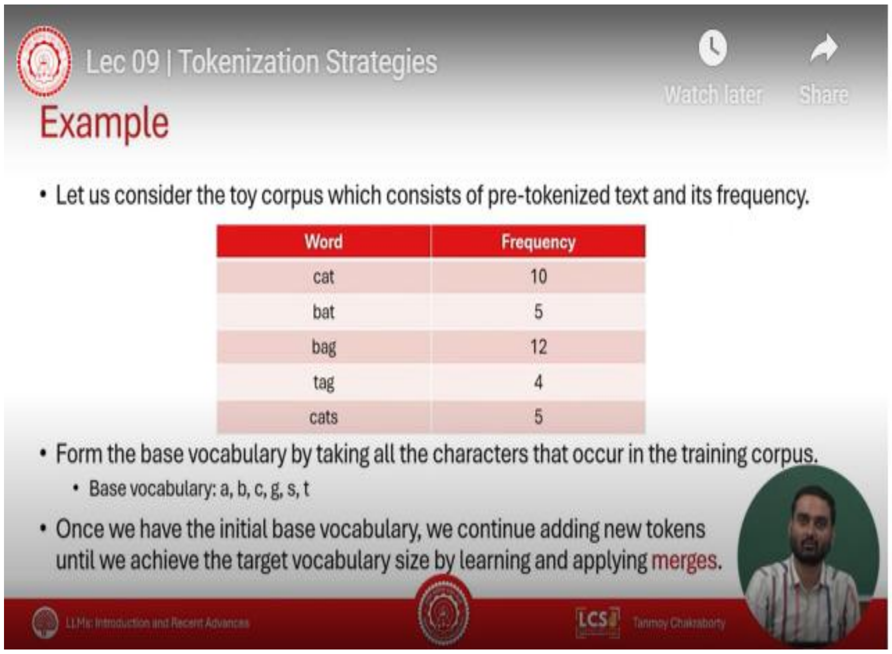
Breakdown Of BPE Algorithm

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.

Now what is the step 3? Step 3 is merging right Then we do once we identified the base vocabulary what we do we look at all combinations of these characters So far we have h-e-l-o these four characters then we look at all possible combination of characters. we start with bigram right bigram character let's say h e right e l h l h o l o and so on and so forth all possible characters all possible character combination and we choose that particular combination whose frequency is high in the training set.

I will explain what is the meaning of high frequency. Let's say for example, in our example let's say the combination LL right appears most frequently right in the bigram right. So LL is basically the most frequent bigram character in the training set, we merge L and L and that will become a vocabulary entry now.



Lec 09 | Tokenization Strategies

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

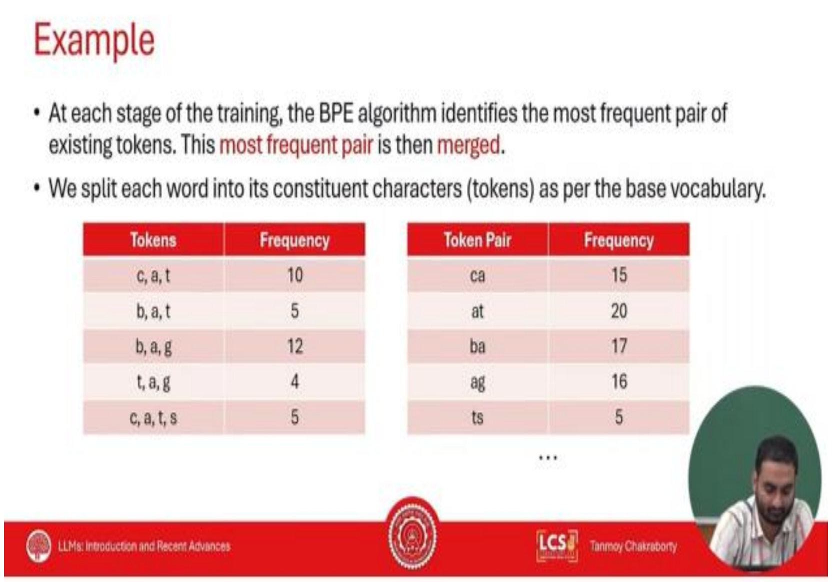
- Form the base vocabulary by taking all the characters that occur in the training corpus.
  - Base vocabulary: a, b, c, g, s, t
- Once we have the initial base vocabulary, we continue adding new tokens until we achieve the target vocabulary size by learning and applying merges.

Okay, so let us take an example and explain this. But before that, so we keep on merging this thing until unless a specific size of the vocabulary is reached, right.

Let's say we predefined the size of the vocabulary to be, let's say to be 1000, right. So we keep on merging bigram, trigram, fourgram and so on and so forth and we stop when we see that the specified size is achieved, okay. All right, let's say these are 5 words cat, bat, bag, tag and cats, okay. These words are present in the training set, right and their frequencies are as follows 10, 5, 12, 4 and 5. So the word cat appears 10 times, word bat appears 5 times and so on and so forth in the training set, right, okay.

So to start with what we do, we first identify the base vocabulary. Base vocabulary is identified from the characters, unique characters present in this corpus. So unique characters are A, B, C, G, S and T. So this is my base vocabulary.

Then we look at combinations. We look at the combination and the frequency and based on the highest frequency, we merge that particular combination, right? So how many combinations are possible? AA, AB, AC, AG, AS and AT, then again BB, BA, right? All possible combinations that you can think of, okay?



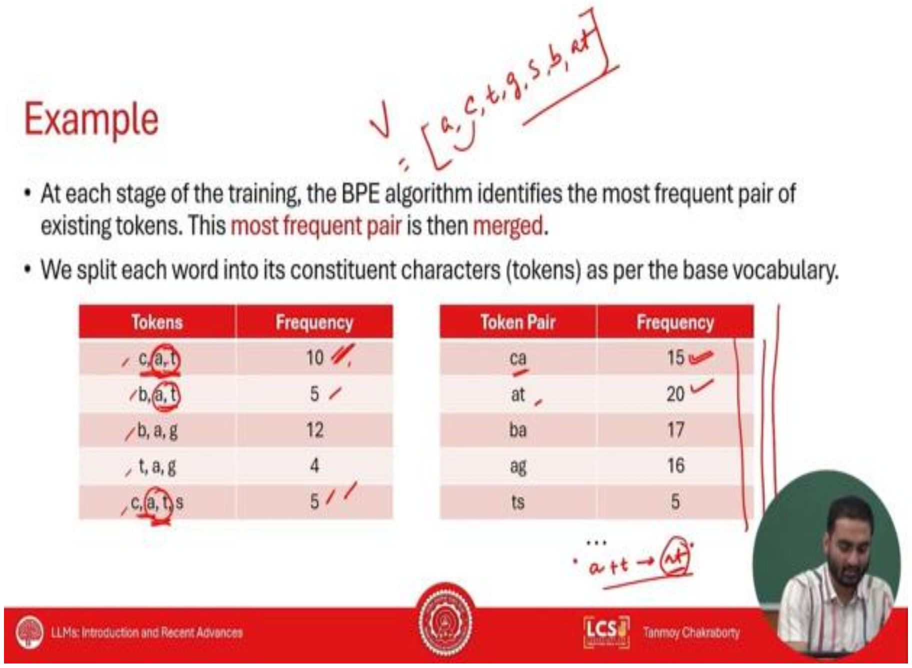
Example

- 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

$V = [a, c, t, g, s, b, at]$

- 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

...

$a + t \rightarrow at$

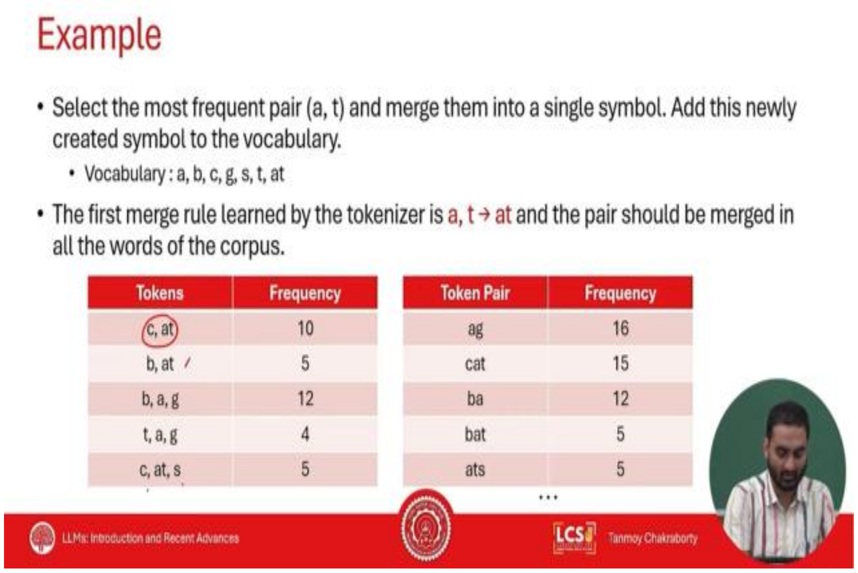
So let's look at it here. So the word cat, bat, cat, bat, bag, tag, and cats, these are the tokens present. So initially, as I mentioned, the unique characters are part of the vocabulary. So the character cat, C, A, T, G, S, these are part of the vocabulary now.

C, A, T, G, S, B also. Now we look at all combinations. Some of the combinations are listed here. Let's say CA by the CA and AC are different. CA this token pair, the frequency is 15.

Why this is 15? Look at it here. CA appears in the word cat and the word cat appears 10 times that means the word, that means the combination CA also appears 10 times in the words CAT right CA also appears here in the word cats right so the total appearance of CA is 10 plus 5 15. Similarly another combination is 80, 5 times here and 10 times here. So the frequency is 20 and so on and so forth. We calculate all bigram characters and their frequencies. which one is the highest right the highest is 80 right.

So therefore we combine A and T with this new split rule which is A plus T goes to 80. So this is the new split rule that we learned right from this table okay. So once we identified this what we do? All the appearances of A and T, A, T will be replaced by the new vocabulary word which is AT. Remember AT is new vocabulary word.

So AT will be added here. So this is a vocabulary V, right? So A and T will be added here and all these combinations will be replaced by the unique token AT.



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.

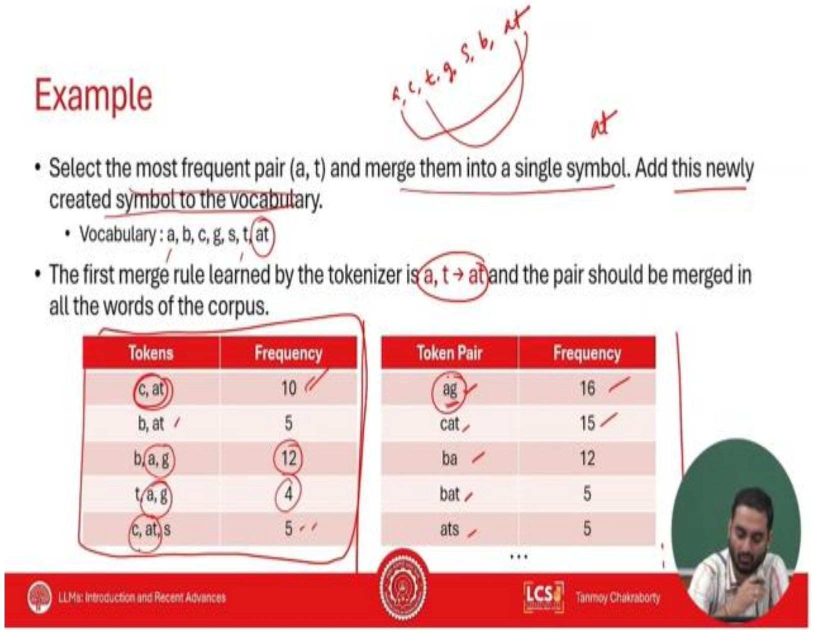
Tokens	Frequency
c, at	10
b, at	5
b, a, g	12
t, a, g	4
c, at, s	5

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

...

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty



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
...	

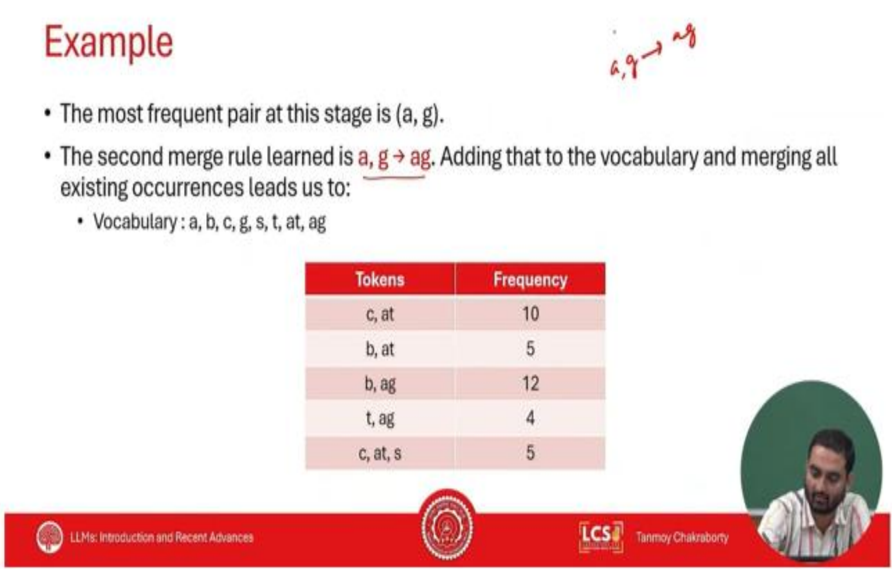
Okay, now look at here, C, AT, it is not C, A, T, this is C, AT, B, AT, right, C, AT, S and so on and so forth. Okay, so what we did, we select the most frequent pair A, T, merge them into a single symbol, a single symbol is now AT, add this newly created symbol to the vocabulary AT. But remember one thing, the earlier symbols which were used to create the new symbol A and T, they are also present in the vocabulary. It is not that we removed those two symbols A and T and we removed them by another, we replaced them by another new token A T.

No, those are also there and along with those you added a new token which is A T. Okay. All right. We learned a new rule, which is a plus t equals to at, a comma t, you know, leads to at. Now, these rules are important, right? We keep on collecting these rules, we keep on expanding the vocabulary, we keep on collecting these rules.

Okay now look at here now this is my modified table right. In the modified table again we keep on combining. Now here what are the combinations possible, again it depends on the vocabulary right my vocabulary is this one a c t g s b at . How many possibilities are there? Again, AC, AT, AG and so on and so forth.

Bigram possibilities are there. Trigram possibilities are also there, right? You can mark A with AT, AAT, right? C with AT, CAT and so on and so forth. Bigram and trigram possibilities are all there. As you see, this is a bigram possibility and these are trigram possibilities, right? This is again another bigram possibility and so on. This is a partial table by the way. If you merge Ag, look at the frequency, Ag appears here with the frequency 12, Ag is here with the frequency 4, so total is 16.

What about Cat? If we merge C and At, 10, C and At, 5, 15 and so on and so forth. Again, what is the highly frequent a token pair which is Ag in this case right.



Example

$a,g \rightarrow ag$

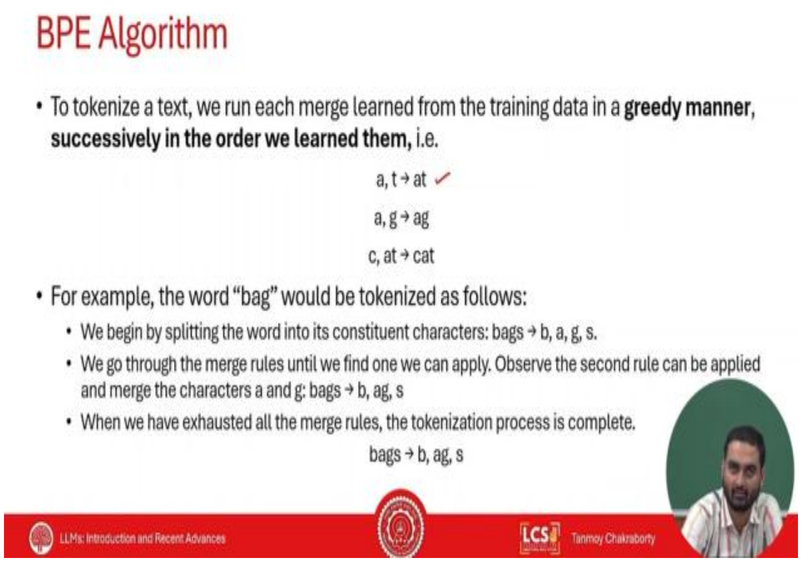
- The most frequent pair at this stage is (a, g).
- The second merge rule learned is $\underline{a, g \rightarrow ag}$. Adding that to the vocabulary and merging all existing occurrences leads us to:
  - Vocabulary : a, b, c, g, s, t, at, ag

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

So, we repeat the same thing, we learn a new rule a, g derives Ag initially we learned earlier we learned a, t derives At now a, g derives Ag. right and what else we do? We replace all the occurrences of a, g by ag. So this is going to be b, ag, this is going to be t, ag and so on and so forth, right.

This is my modified frequency table. This is the modified vocabulary where you see both these new tokens have been added and the two rules have also been added. Okay now again we do the same thing we keep on merging it can be 4 gram now right. As you see here cat plus s right trigram, trigram, trigram, trigram, 4 gram right and you see here the most frequent you know merging could be the most frequent token can be cat right with the rule c comma at derives cat. Okay, so the word cat will now be added to the vocabulary. Right, AT along with AT and AG, a new token which is added is CAT.

Okay, so all appearance of CAT will now be replaced. I mean, C comma AT will be replaced by CAT.



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

Okay, so we learned three rules A, T derives AT, A, G derives AG, C, A, T derives CAT right and what is my final vocabulary till this stage the final vocabulary is this one. a, b, c, g, s, t, then at, ag and cat. We keep on doing this until and unless the predefined threshold of this vocabulary is reached. Let us say the size of the vocabulary that we are expecting is 1000.

We keep on expanding and we stop when this predefined threshold is reached. Now let us see what happens during the test time. Right. So this is essentially, you remember again, I'm going back, back and forth, token learner.

So this is the task of the token learner. So whatever I discussed so far is a task of the token learner to learn the token and the vocabulary. Now we are executing token segmenter, right? What the token segmenter will do, whatever was learned, in the tokenization stage that rules that vocabulary will be used to segment unknown tokens. Okay so let's say the word is back BAG in your test set okay. Token segmenter will be executed here, so what it would do first? It would first split the word BAGS into…

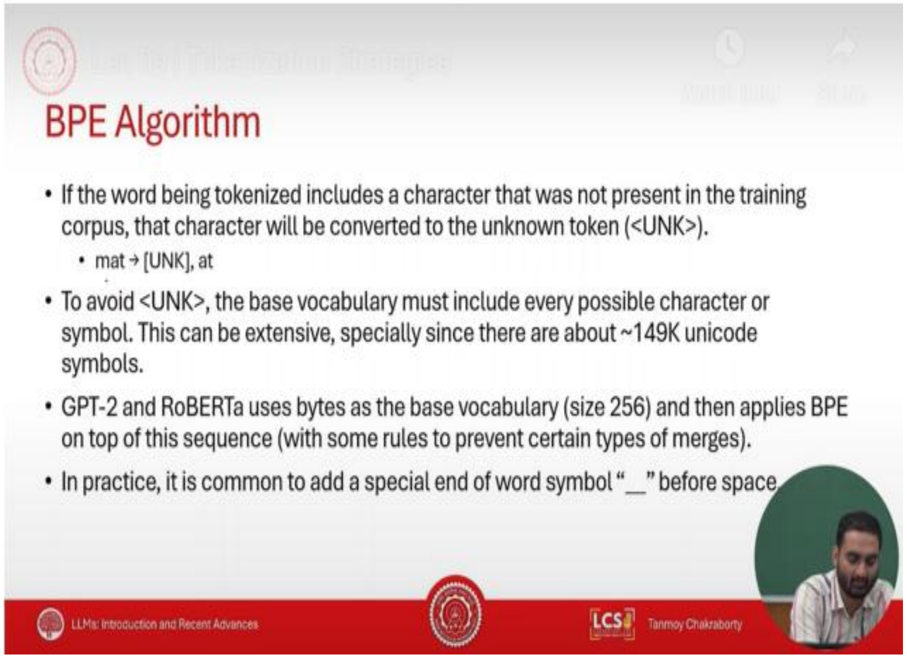
So it basically should be BAGS I think there is a typo here BAGS. So it first segment the entire word into characters B, A, G and S and then it will start applying the set of rules that we have learnt so far. But remember, the set of rules are applied in sequence, right? We'll first apply this, then this, and then this. The way we curated the rules, the same way we basically apply these rules, execute the rules.

It is not the case that we first apply C, A, T, and then A, T. The way we applied the rule, the way we curated the rule, the same sequence will also be used to execute those rules, OK? So we go through the merge rules right until we will find we basically we find one that we can apply for example right if I apply a, t derives at it is not there right if I apply a, g derives ag this can be applied here right so a, g will now be replaced by ag. So now the modified, the updated set of tokens will be B, AG, S. Now let's see whether I will be able to apply some other rules or not. Of course C, AT will not be applied. So the updated or the final tokenization, the final tokens for the word BAGS is going to be B, AG, S.

Okay and this will be fed to the transformer or to the RNN method right. Now what happens if a token is not present right in the vocabulary. Now remember here in this case all the tokens b, ag and s right b, ag and s they are part of the vocabulary Right? All three are part of the vocabulary. But let's say you have a word M U G.

Right? The word M, the character M is not present here. Right? The character U is also not present here. How do you deal with this? During the test time again.

Let's see. Okay.



BPE Algorithm

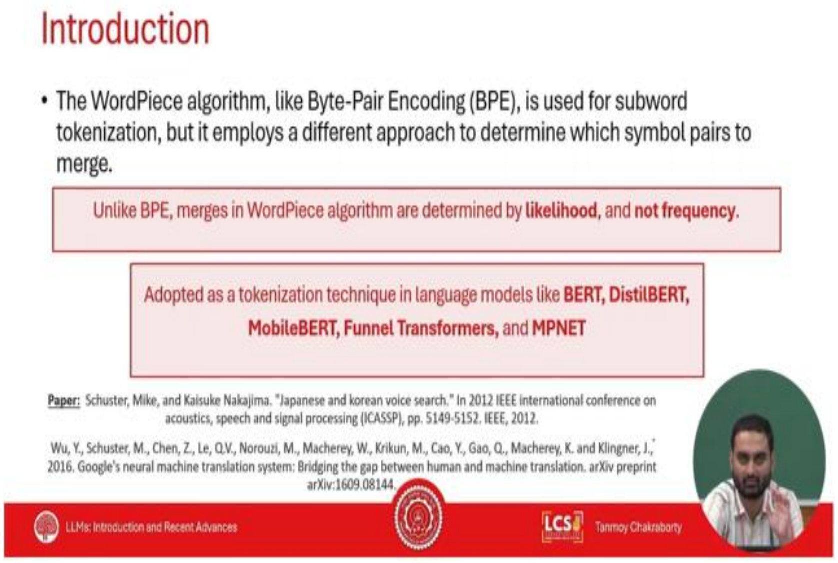
- If the word being tokenized includes a character that was not present in the training corpus, that character will be converted to the unknown token (<UNK>).
  - mat → [UNK], at
- To avoid <UNK>, the base vocabulary must include every possible character or symbol. This can be extensive, specially since 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.

Let's say the word is mat, M A T. Okay, and again we start applying the rule right First mat will be split into m comma a comma t right then we apply the first rule m comma a comma t right then we apply the first rule which we learn which is a comma t derives at. Right so a comma t will be replaced by at now i have two tokens m comma at now i can't apply other rules i can't apply other rules so the final token should be m comma at but the token m is not present in the vocabulary right. So if it is not present we also don't know its corresponding embedding right, think about what to wake, think about you know a transformer. In transformer when we feed all the tokens, we basically feed their corresponding embeddings, right. So M is not present in the training set, M is not present in the vocabulary, therefore M doesn't have an embedding, right.

What we do here, we replace all such unknown characters by a token called unk, u n k, okay. This unique token unk is always a part of the vocabulary that you have learned okay. It is always a part of the vocabulary that you have learned so far right. The base vocabulary must include every possible character or a symbol like unk. Okay so this will be replaced by unk and then You already know the embedding for AT because this is a known vocabulary entry.

So this is going to be my tokens for the word MAT. Clear? In this way, we deal with unknown. We'll see in case of WordPress, we have a different strategy. In fact, in GPT-2 and Robota, what they use, they use basically bytes, the bit-byte concept instead of characters, right? These bytes are basically parts of their base vocabulary, and then on top of this, they keep on applying these rules. Okay, so this is a simple version of BPE, byte pair encoding. Of course, you can think of modifications, further modifications on top of this.

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**.

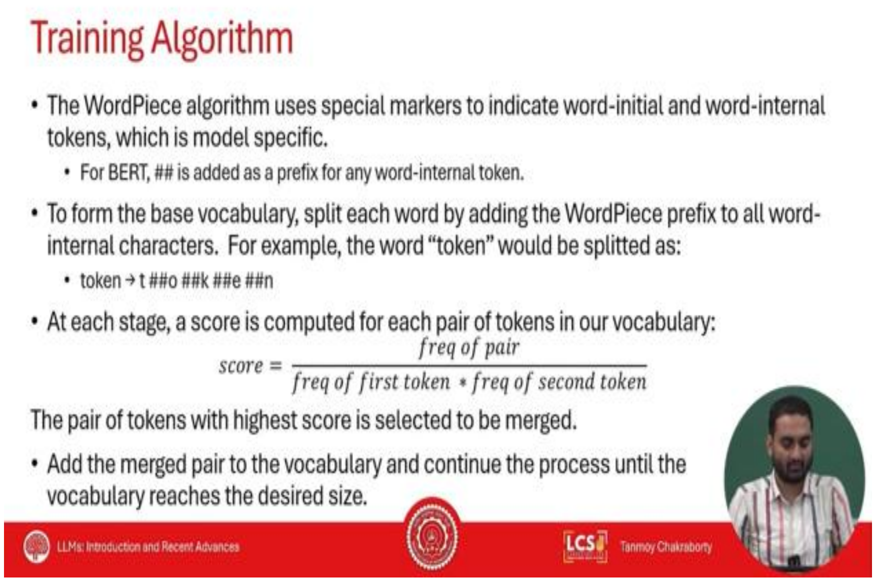
Adopted as a tokenization technique in language models like **BERT, DistilBERT, MobileBERT, Funnel Transformers,** and **MPNET**

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.

Let's move to the next part, which is word piece tokenization. The idea behind WordPiece tokenization is very same as BPE. The only difference is that here the merging operation, merging of two characters, merging of two tokens into sub-tokens is not performed based on the frequency. It is performed based on another property, another formula that we'll discuss here. So in WordPiece we merge based on something called the likelihood, not frequency.

How do we define likelihood? We will discuss. What this tokenization is used in BERT, the Google's BERT paper, DistilBERT, MobileBERT, FunnelBERT and MPNet and so on and so forth. This was proposed way back 2012 and then I think later on it was modified for the task of language processing.



Training Algorithm

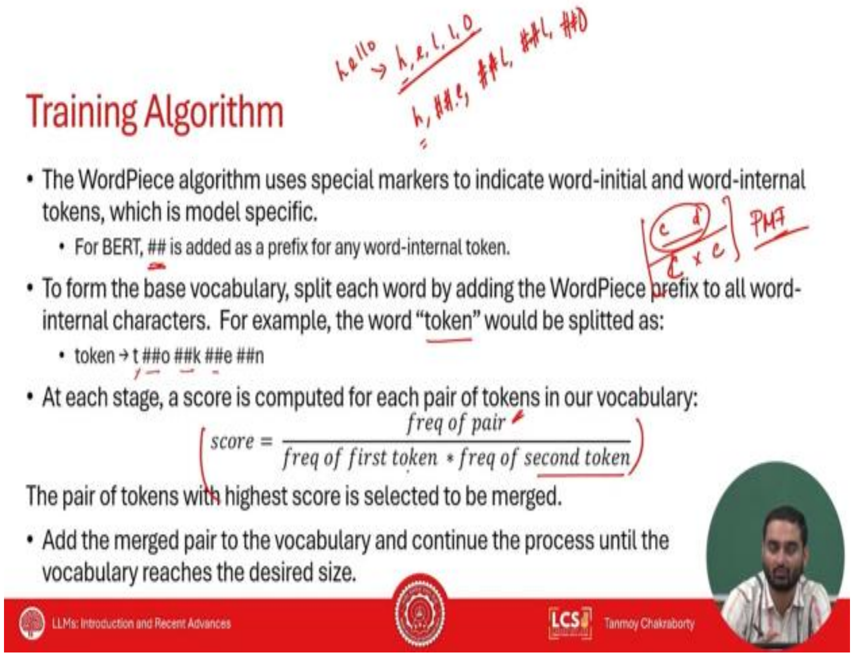
- The WordPiece algorithm uses special markers to indicate word-initial and word-internal tokens, which is model specific.
  - For BERT, ## is added as a prefix for any word-internal token.
- To form the base vocabulary, split each word by adding the WordPiece prefix to all word-internal characters. 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.

LLMs: Introduction and Recent Advances   LCS   Tanmoy Chakraborty



hello → h, e, l, l, o

h, ##e, ##l, ##l, ##o

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:

$$\left[\frac{c\ d}{c \times c}\right] \text{PMI}$$

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

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.

LLMs: Introduction and Recent Advances   LCS   Tanmoy Chakraborty

Okay, so let's look at the steps here. It again has two steps. In the first step, during the training time, we have the training corpus training document.

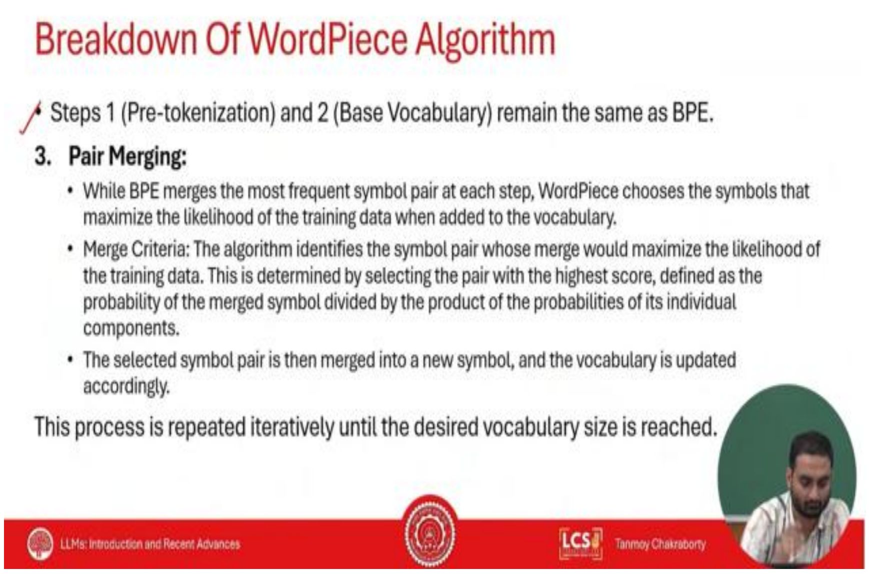
And from the training document, we identify the vocabulary, we identify the rules, and so on and so forth. And then during the test time, we use this vocabulary and rules for splitting a word into subwords. So here what we do. Remember in the BPE, what I mentioned, let's say a word hello, H-E-L-L-O, right? It was split into H-E-L-L-O at the beginning, right? Character level split. But here, what we do, we use a special character double hash, right to indicate to differentiate between the character which appears at the beginning of the word and the characters which appear you know in the intermediate part of the word right.

So here the word HELLO will be split in this way H comma double hash E comma double hash L comma double hash L comma double hash O. Double hash is a prefix this indicates that the token is an internal token not the beginning of a token. Okay, so similarly the word TOKEN will be split into T, double hash O, double hash K, double hash E, double hash N. Right, now for every, so when we think of merging, right, we use, let us say, let us say we want to merge two characters C and D.

In case of BPE, what we did? We looked at the frequency of CD. Here what we do, we use this core function. This is very simple. Frequency of CD divided by frequency of C times frequency of D. This is a kind of mutual information, PMI, pointwise mutual information that we generally discuss in the semantics of NLP.

Okay, so this is kind of my likelihood. Okay, so frequency of the pair divided by frequency of the first token times frequency of the second token. What is the beauty of this one? The beauty of this score is that let's say even if a pair is highly frequent, a pair is highly frequent, if their corresponding constituent characters are also highly frequent, that specific pair will be penalized. right versus let's say there is a case where the pair is you know not that frequent kind of medium frequent right but the tokens are not the corresponding tokens are not that frequent that particular pair will be given high priority.

We will discuss with an example okay.



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.

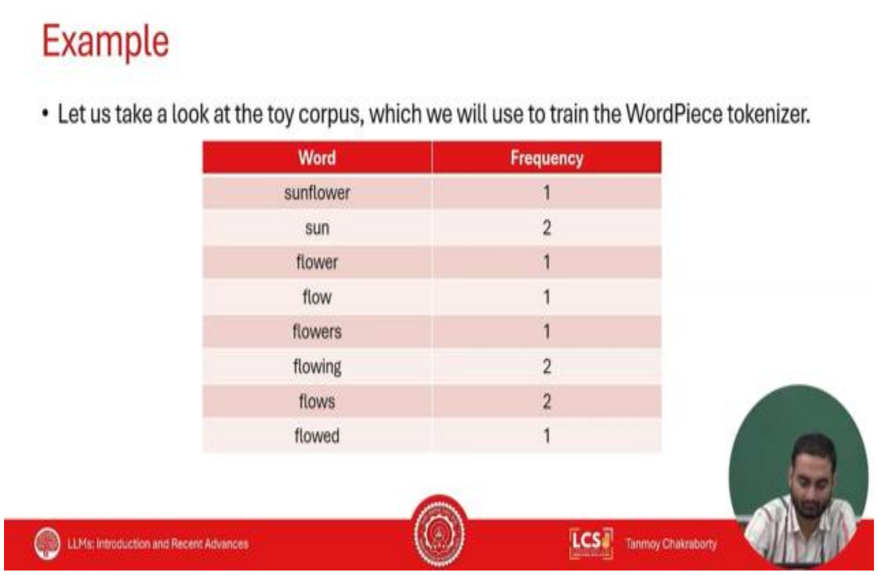
This process is repeated iteratively until the desired vocabulary size is reached.

LLMs: Introduction and Recent Advances LCS2 Tanmoy Chakraborty

So step 1. Step one and step two are the same as we discussed in case of BPE. Step one is basically a pre-tokenization where we clean the corpus, we split the sentence into words based on space characters, based on space tabs, some other unicorns and so on and so forth. Step two, we identify the base vocabulary. Base vocabulary in our case again the base vocabulary constitutes all this tokens, all these characters.

But remember here, characters are differentiated based on the positions. If a character appears at the beginning, it will be added to the vocabulary without hash. If the character appears at the middle or the end, it will be added to the vocabulary with a prefix of double hash. Now, step three is the merging. What we do here? We choose the symbols that maximize the likelihood of the training data when added to the vocabulary. Okay? What is the merge criteria? The algorithm identifies the symbol pair whose merge would maximize the likelihood of the training data.

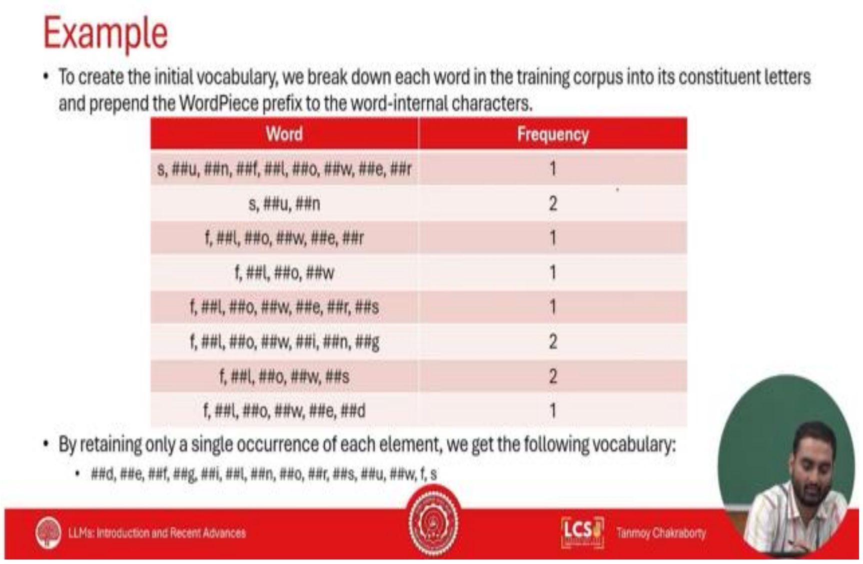
This is determined by selecting the pair with the highest score, the score that I discussed earlier, defined as the probability of merge symbols divided by the product of the probabilities of its individual components. Okay? And again, similar to BPE, we stop the method when a particular size of the vocabulary 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

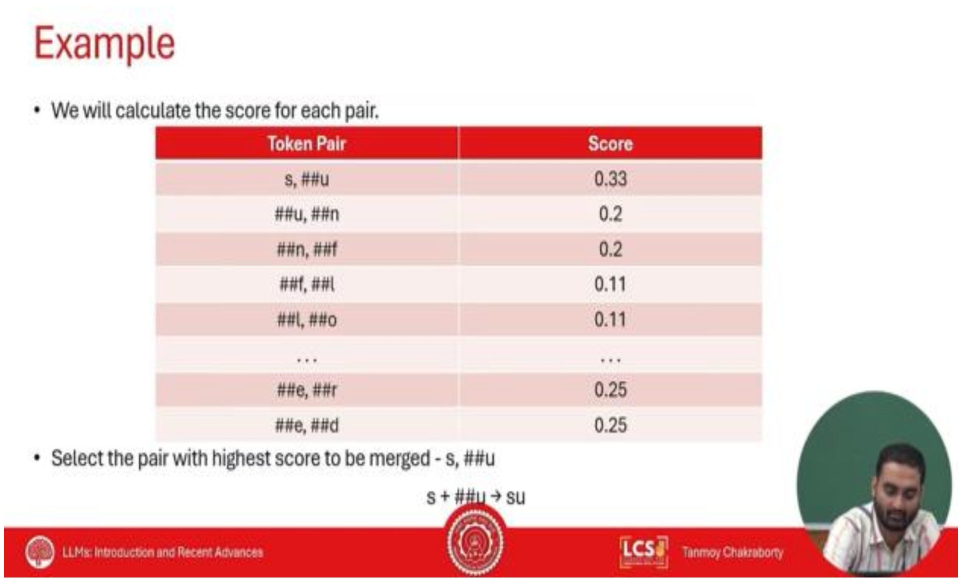


Example

- 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

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

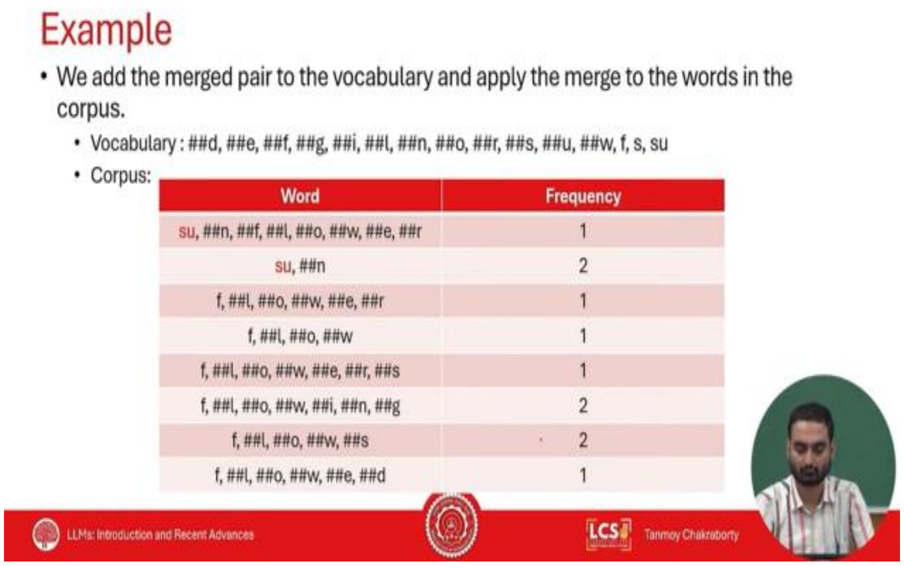
LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Let's again take an example. Okay, so let us assume that these are the training words, present in the training corpus, sunflower, sun flower, right, flow, flowers, and so on and so forth, and the corresponding frequencies. Right? Now, when we segment, Right? As you see here, sunflower, sun, hashtags u, hashtags n, hash, hash, hash, hash, right? Sun, hash u, hash n, and so on and so forth.

As I mentioned earlier, right, all the initial characters will be there without hash, and all the intermediate characters will be there with hash. And what is the vocabulary now? The vocabulary is this one. You see here, there are two unique starting characters, s and f, s and f. right and all the other characters are intermediate characters.

Therefore, they are preceded by a prefix, a prefix is double hash here. Similar to bp, here also I start with merging a pair of characters right for example, s and u let's look at s and u so s and u right so s and u. So the bigram SU right, what is the frequency of this bigram? The frequency is 1 plus 2, so 3 and then is there any other S and U? No right, so the frequency is 3 and what is the frequency of S? The frequency of S is 1 plus 2, 3 plus 2 right, so 3 3, 4, 4 and then 6, right? So 6 times what is the frequency of U? Frequency of U is 1 plus 2, 3. So 3 and then that's all, right? So 3, okay? So the score for SU is basically 1 by 6, right? right you look the calculation may be wrong, but I hope you understood the concept. So similarly we merge u n, u f, f l, l o and so on and so forth right and we obtain the corresponding scores. Here again, you basically identify that pair whose code is the maximum, in this case S and U, S and U will be merged and the new token will be formed which is SU.

Now here, important thing to note is that when you merge S and hash U, the resultant token is not hash SU, this is only SU, why? Because S is basically the initial character. But let us say if you had merged this one, the resultant character would be un because both un are the intermediate characters not the starting character. So in this case, you identified S and U, you merge them, and the new vocabulary word is 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

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty

So SU is a new vocabulary word, along with the previous words, previous tokens present.

By the way, I'm misusing the term word. I'm basically using the term word and tokens interchangeably, but I hope you understand. So once this is identified then what we do again similar to the BPE, I replace all s hash u will be replace this by su, all appearances of su will be replaced by su and so on. okay. So this is my modified table.

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

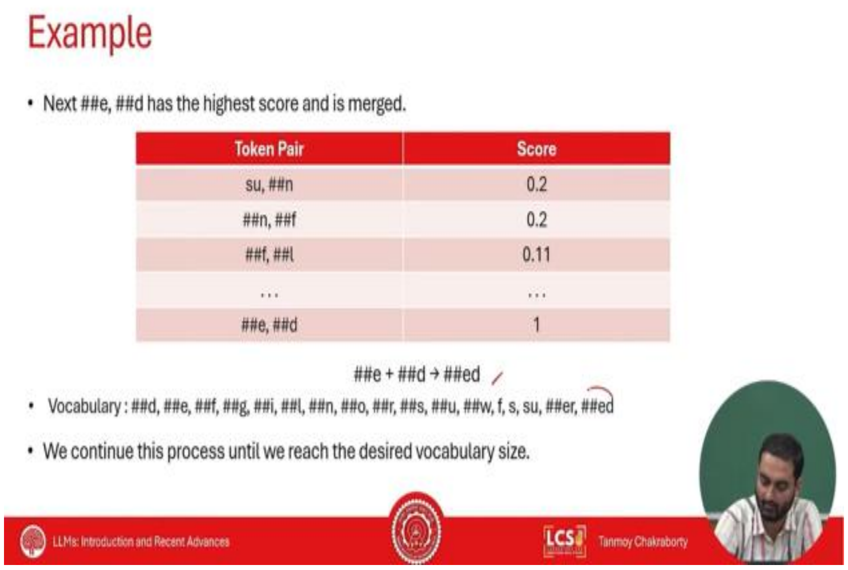
- The best score is shared by ##e, ##r and ##e, ##d. Let's say, we select ##e, ##r as the best pair and merge them.

##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

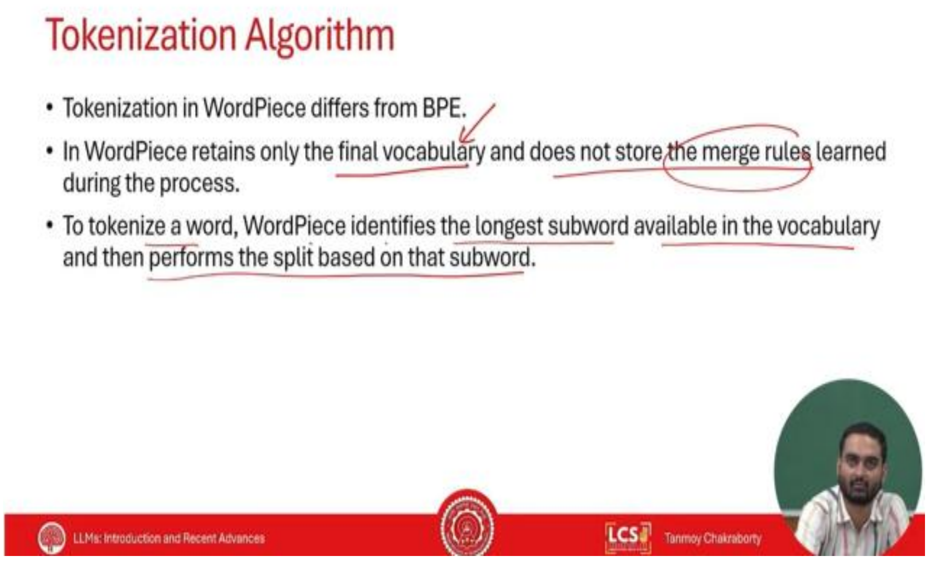
##e + ##d → ##ed

- 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.

Again, on top of this I apply the same set of rules, right, bigram, bigram combination, trigram combination.

You see here these are trigram combination, these are bigram combinations, right. And Let's say in this case, there's a tie, right? So we can break the tie by choosing one of them. So let's say in this case, we chose E and R to be merged. So E and R will be merged, and the new token hash ER, which will be a part of the vocabulary. Okay, hash er is a part of the vocabulary now and we replace all the appearances of e, r by hash er, right and so on and so forth.

So your vocabulary size gets increased, right, the table gets modified and so on and so forth. The next stage you again merge ed, right, it will be added here, vocabulary size increases and so on and so forth.



Tokenization Algorithm

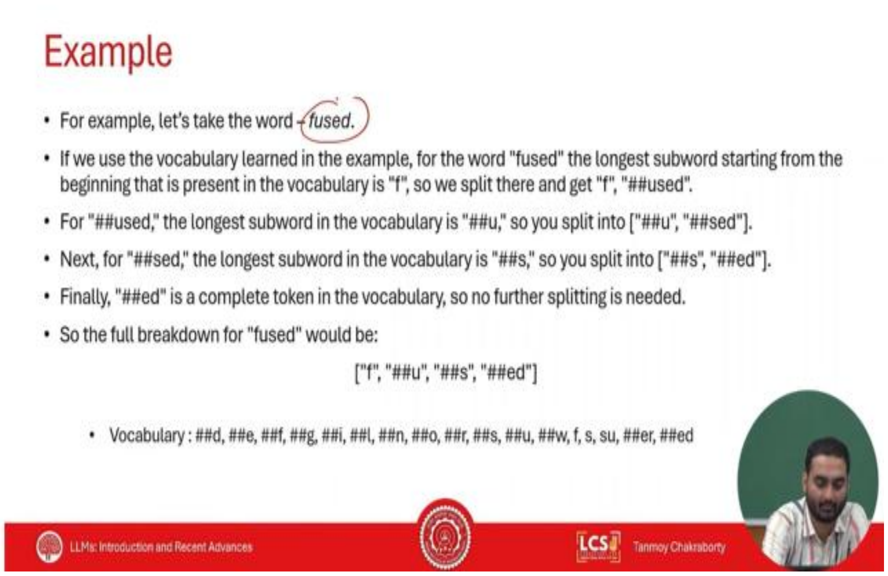
- Tokenization in WordPiece differs from BPE.
- In WordPiece retains only the final vocabulary and does not store the merge rules learned during the process.
- To tokenize a word, WordPiece identifies the longest subword available in the vocabulary and then performs the split based on that subword.

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

Alright, so this is the tokenization step for WordPiece? Now, so this is the training time. Now what happens in the test time? This is very important. During the test time, remember in case of BPE, what we did? We first split a word into characters, right? And then we applied all these rules one by one in sequence, right? And then we kept on combining characters based on these rules, right? And through that process, we ended up basically splitting the word into tokens.

We use the tools in case of BPE, right? Here, in case of WordPiece, we will not use any rule. We will not store the merge rules. We will only store the vocabulary. That's the difference here. We will not store the rules at all. We will only store the vocabulary, right? Now, if we have only the vocabulary, the updated vocabulary that we obtained from the training set, How do we split an unknown token into subtokens or a test token into subtokens? Let's see.

So for that, wordpiece identifies the longest subword available in the vocabulary and then performs the split based on the subword?

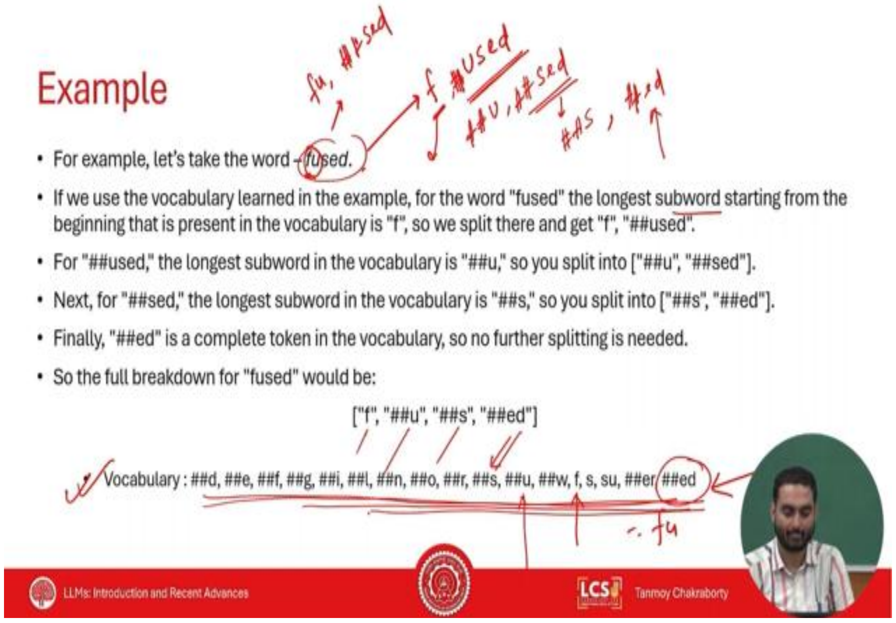


Example

- 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

- 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

LLMs: Introduction and Recent Advances | LCS | Tanmoy Chakraborty

Let's see. Let's say the word is fused.

F-U-S-E-D. And let's say this is my vocabulary that we have already learned from the training set. Okay. We don't have the rules. We haven't stored the rules. So here if you start from the beginning again, you start from the beginning, right? If you look at the vocabulary here, what is the longest sub-word or what is the longest sub-word which is present in the vocabulary? So if, let's say you start from f, right? f is present here, of course, f is present, but f may not be the longest.

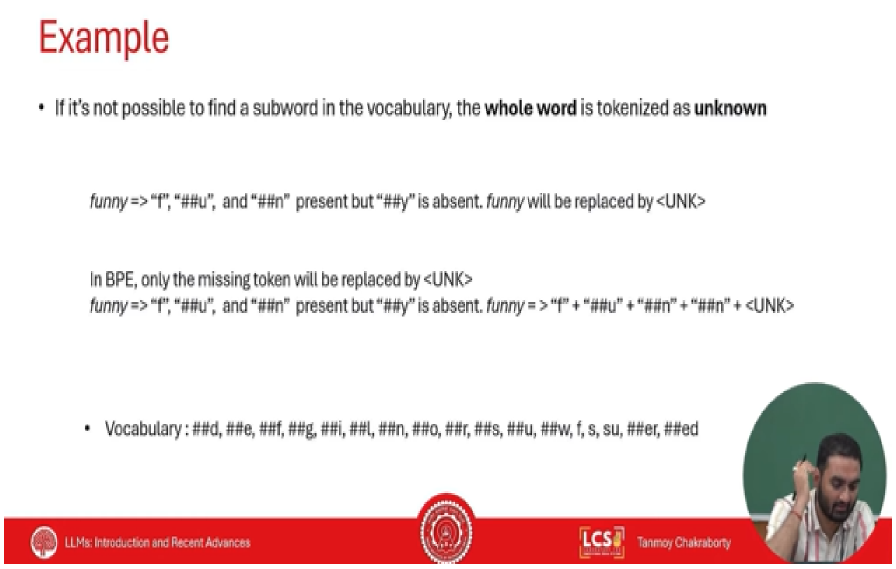
Let's look at if u. If u is present here, does f u present here? f u is, so f u is not present here. right. So it means that the longest sub word here is f, right, fu is not present here. Of course if fu is not present, fus will also not be present. So we split first fused by f, so this will be split into f, used, of course this hash, okay.

Now I know that this is the longest one that is possible so I keep this thing aside. Let us look at this one. Can we split this further? What is the longest sequence here? USED. So of course if we consider only hash U, hash U is present here. What about hash US? Hash US is also present here? No. That means from here also the longest sub word is hash u and the remaining part will be separated.

Now let us look at this one hash sed of course double hash. What about hash sed? Hash s is present here, yes hash s is present, What about hash se, hash se is not present here, That means again the longest word which is possible here is hash s comma hash ed, let us look at this one again, okay hash ed is present, yes hash ed is present here. Okay, so the final tokens that are derived from fused is f, hash u, hash s, hash ed. Okay, we haven't used any merging rules, right.

What we did here, very simple idea. Given a word, right, we start from the beginning. We look at what is the longest subword present in the vocabulary. Let's say, again hypothetically assume that the token fu was present here. The token fu was also present in the vocabulary, right? If it was the case, then the tokens would have been fu comma hash sed. But this is not the case here.

So we start from the beginning, we look at the longest subword which is present in the vocabulary or not. If it is present, we segment that part and again process the remaining part of the word similarly. We don't need to store the merging rule which is useful in our case because we can basically save a lot of memory.



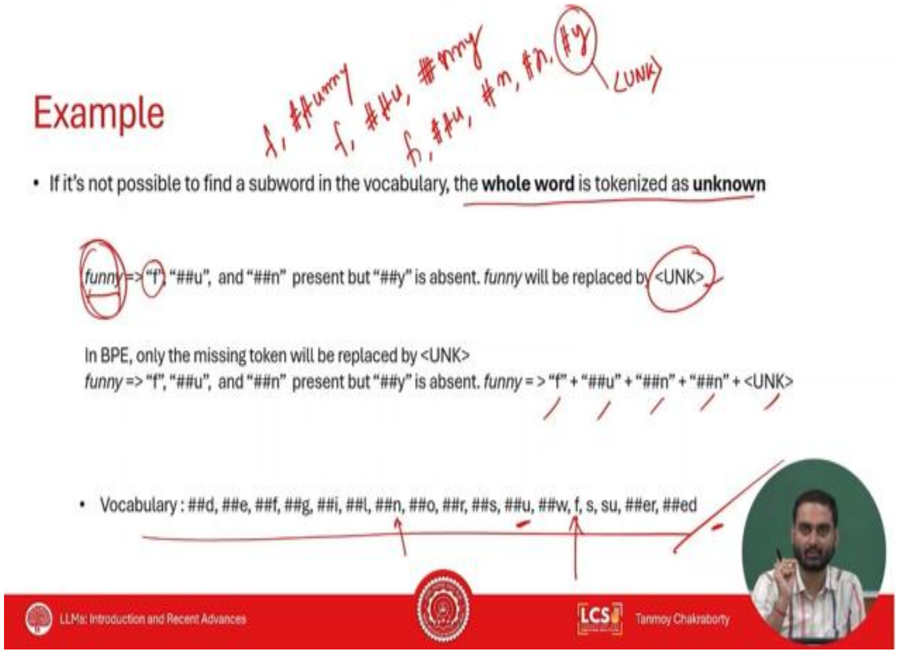
Example

- 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



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

What about the unknown word? Let's say the unknown word is funny, F-U-N-N-Y. Again, this is my vocabulary. I start with f. So the word f is present here. The word f is present here. But the word f u is not present here.

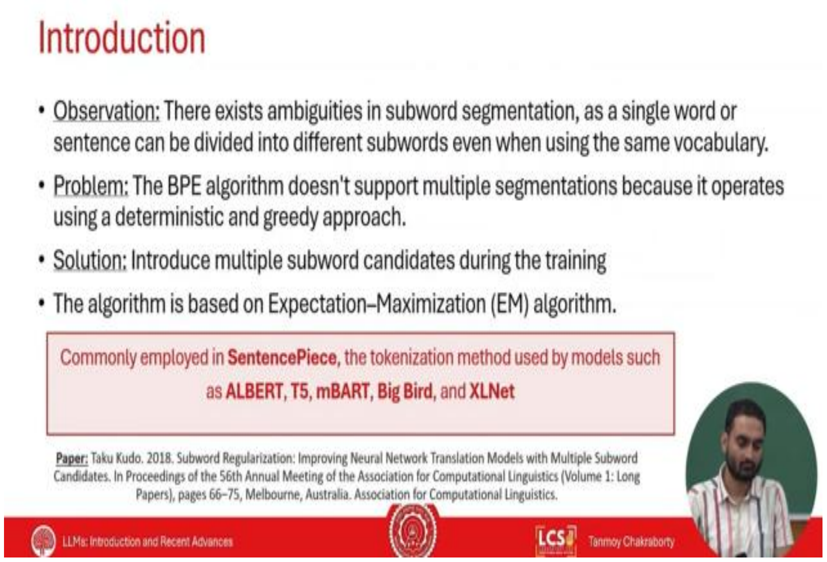
So the longest word is f, right? Then let's say f, right? You split it into tokens. So f then hashtag hash u n n y right. Again, maybe the unique token is the longest token is hash u. So f then hash u and then hash the remaining part is hash nny right.

Hash n n is not present but hash n is present. so f hash u, hash n, hash n, and the remaining is hash y. Let's look at hash y. Hash y is not present here. So hash y is not present here. It is not present. What we do here, what we did in case of BPE? In case of BPE, we replaced this by the token unk.

In case of BPE, the tokens were f, u, n, n, unknown. But in case of WordPiece, what we do, the whole word is tokenized as unk. I replace the whole word by the token unk, not the subword. The entire token, the entire word will be replaced by unk. If a specific token, a sub-token is not present in the vocabulary, the entire word will be replaced by unk.

That's it. That is about word piece.

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.

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

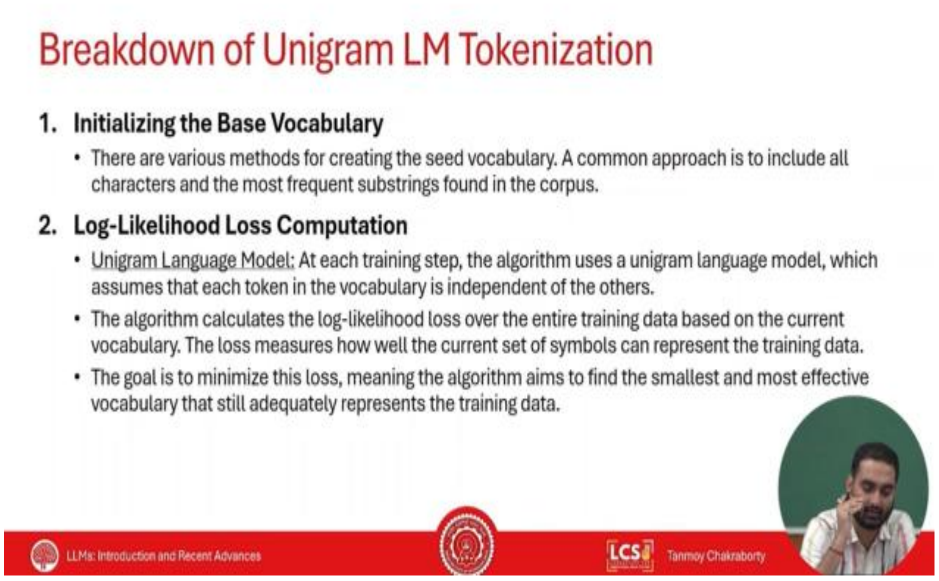
Now let's move to the next part, which is unigram tokenization. So in Unicram tokenization, it's a bit complicated, okay, but it's a very interesting idea. So in this tokenization, so this is used in a specific embedding encoding method called sentence space, right, which is heavily used in recent methods like T5, Ember, ExcelNet, and so on. So the idea here is that we will go back to the concept of unigram language model. I hope you remember what is a unigram language model.

We discussed in the language model introduction chapter, given a sequence of tokens w1, w2, dot, dot, dot, wn. When we measure the probability of the sequence, this token, this is basically a multiplication of all individual tokens, p of w1 times p of w2, and so on and so forth. probability of all individual tokens. We assume that all these tokens are independent of each other. So we use the same unigram language model concept here while segmenting a word into subwords. Right? What is the motivation here? The motivation here is, if you remember the discussion that we had today, right, BPE and WordPiece, both these algorithms produce a single type of segmentation for a specific word.

For a given word, you have only one type of segmentation. But in real world, it may not be the case. A word can be segmented in multiple ways. There can be multiple ways through which you can segment a specific word. So in this unigram tokenization what we do? We introduce multiple subword candidates for a particular token or for a particular word during training, right.

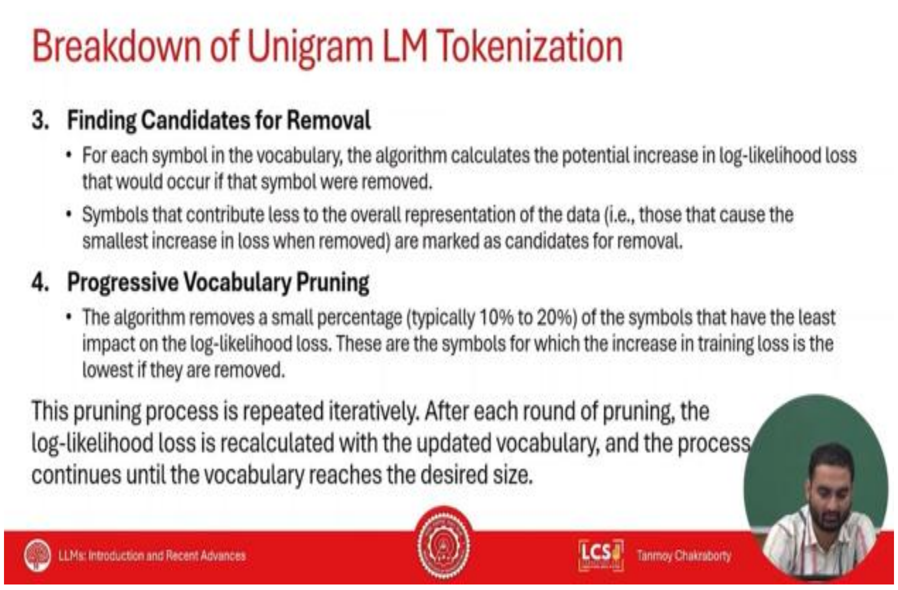
And here we will use the concept of expectation maximization, EM algorithm. So EM as we know it's a very interesting idea. We generally use in non-parametric estimation and so on and so forth. If you are not aware of EM, you go back and check the machine learning codes. For example, the algorithm K-means, this algorithm uses EM algorithm for defining the centroids and so on.

So expectation maximization is used during tokenization. Let's see.



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

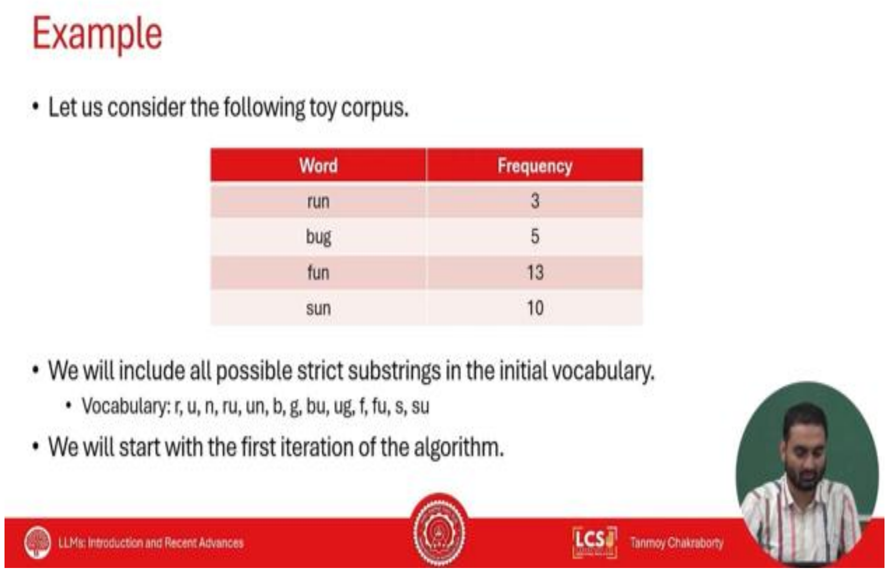
3. **Finding Candidates for Removal**
   - For each symbol in the vocabulary, the algorithm calculates the potential increase in log-likelihood loss that would occur if that symbol were removed.
   - Symbols that contribute less to the overall representation of the data (i.e., those that cause the smallest 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.

So again similar to the previous methods, the first step is to identify the base vocabulary. Now here this Unigram language model tokenization is different from the other tokenization that I discussed today. How? So in both byte pair and word piece, our goal was to increase the vocabulary, right? And so we started with the base vocabulary and we kept on increasing the vocabulary until and unless we reach a certain threshold size of the vocabulary, right? Here, this is just opposite. We have a massive base vocabulary and we keep on reducing the vocabulary.

We keep on shrinking the vocabulary until unless a specific threshold is reached, right? There we increased, here we decrease. So to start with, during the initialization stage, we can use different types of methods to basically get all possible combinations of vocabulary tokens that are possible. For example, you add all the characters, you add all the subwords, possible substrings, and so on and so forth into the vocabulary. And then we later decide that out of these tokens present in the vocabulary, which tokens are useful, which tokens are useless. Okay? So, here what we can do? We include all the characters, all possible, all frequent, you know, medium frequent substrings into the vocabulary.

We can also, here you can also run BPE, for example. You can run BPE exhaustively, right, to create the base vocabulary. Okay? So once we did this, the next stage is essentially to compute the log likelihood loss. So what is this log likelihood loss? The log likelihood loss is computed based on this unigram language model.

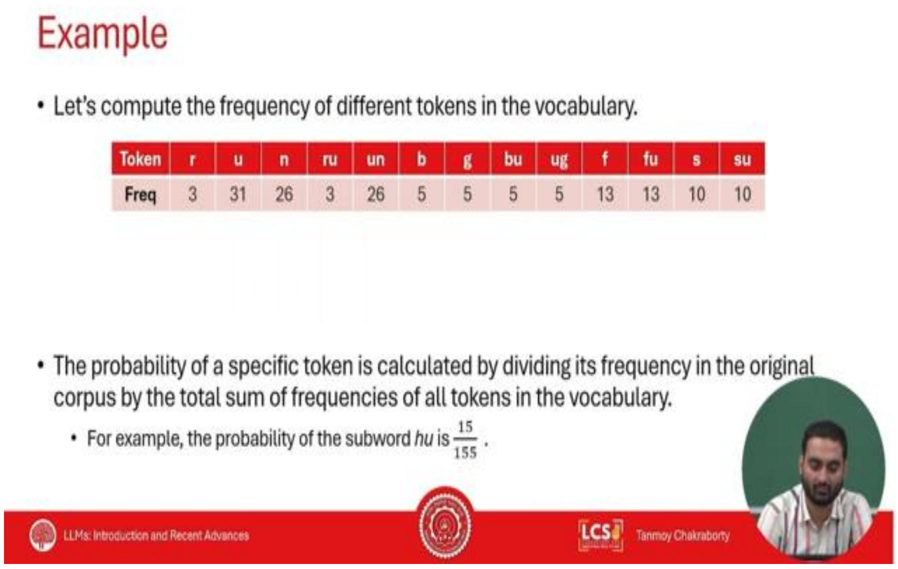


Example

- 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.

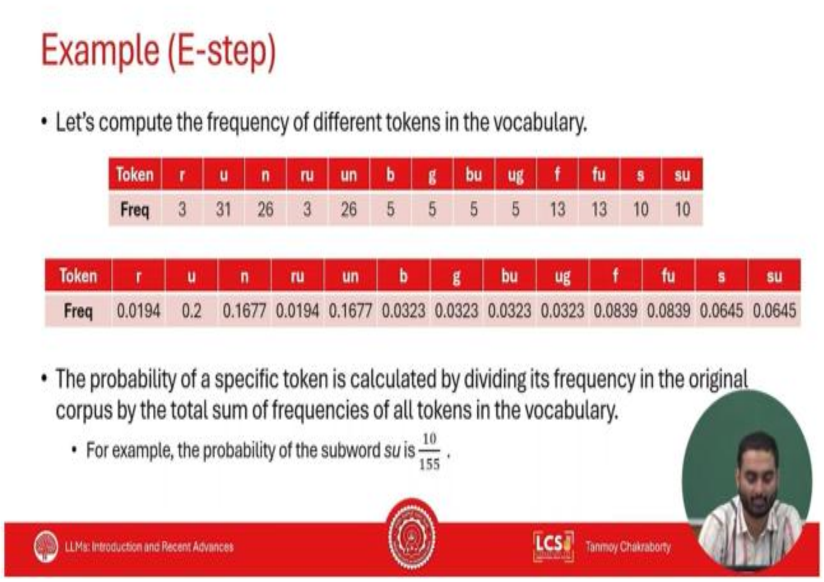


Example

- 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

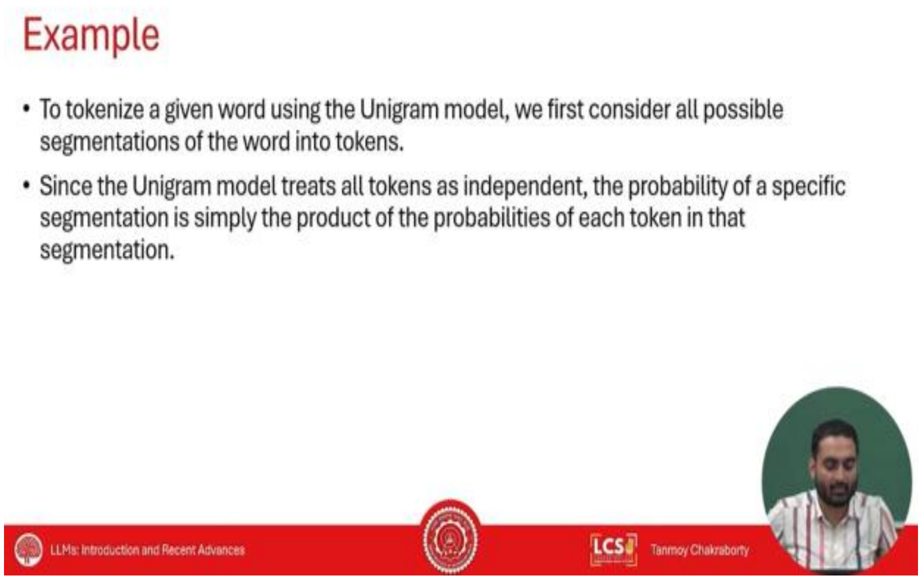
- The probability of a specific token is calculated by dividing its frequency in the original corpus by the total sum of frequencies of all tokens in the vocabulary.
  - For example, the probability of the subword *su* is $\frac{10}{155}$.

It's better to first, I think, let's look at example, and then we go back to the algorithm.

Let's assume that these are the training words, run, bug, fun, and sun. Okay, and these are the frequencies. Let's say we start with this vocabulary, right? Where R, U, N, RU, UN, BG, these are present. There is no hash use here by the way. Okay, these are kind of vocabulary tokens used in BPE, right? Therefore, I said that we can just run BPE initially at the beginning to curate the initial vocabulary for Unigram language model tokenization. Okay then what we do? Now these are my vocabulary words r u n ru un b g and so on and so forth initial vocabulary tokens right and these are the frequencies.

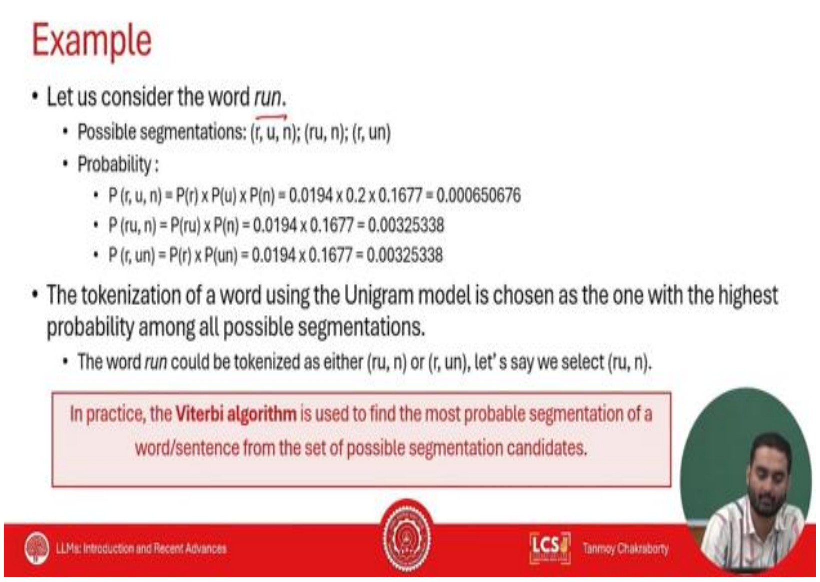
For r this is 3 for u 31 and so on and so forth. We will not use this raw frequency; we will use unigram frequency. What is the unigram frequency? What is the unigram probability here? Unigram probability for R is 3 by the sum of this. What is the sum of all the token? Let's say the number is 155.

The sum of all these frequencies is 155. So the probability here, unigram probability is 3 by 155. This is 31 by 155 and so on and so forth. For example, for the token SU, this is 10 by 155. So we started off with the base vocabulary. Now for each vocabulary entry, we calculated the unigram probability.



Example

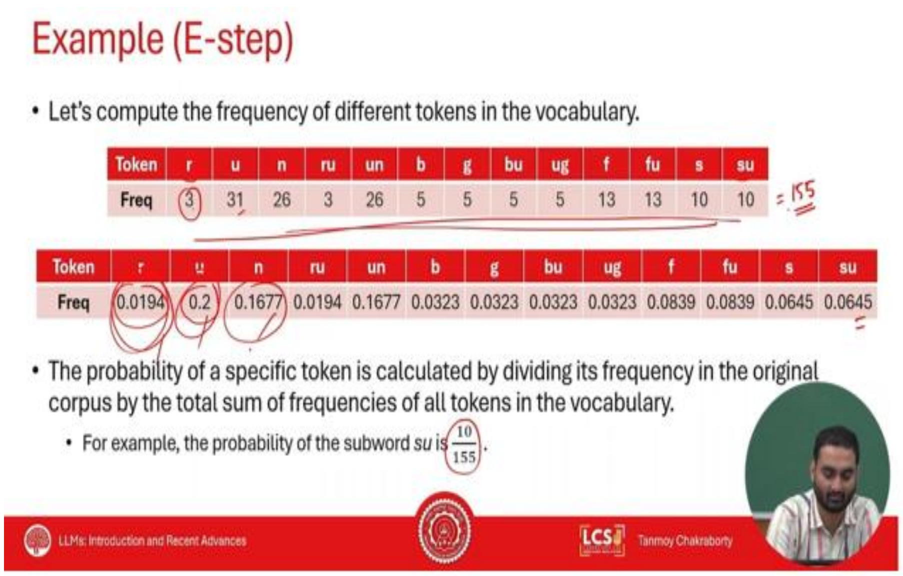
- To tokenize a given word using the Unigram model, we first consider all possible segmentations of the word into tokens.
- Since the Unigram model treats all tokens as independent, the probability of a specific segmentation is simply the product of the probabilities of each token in that segmentation.



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 (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

= 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

- 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 $\frac{10}{155}$.

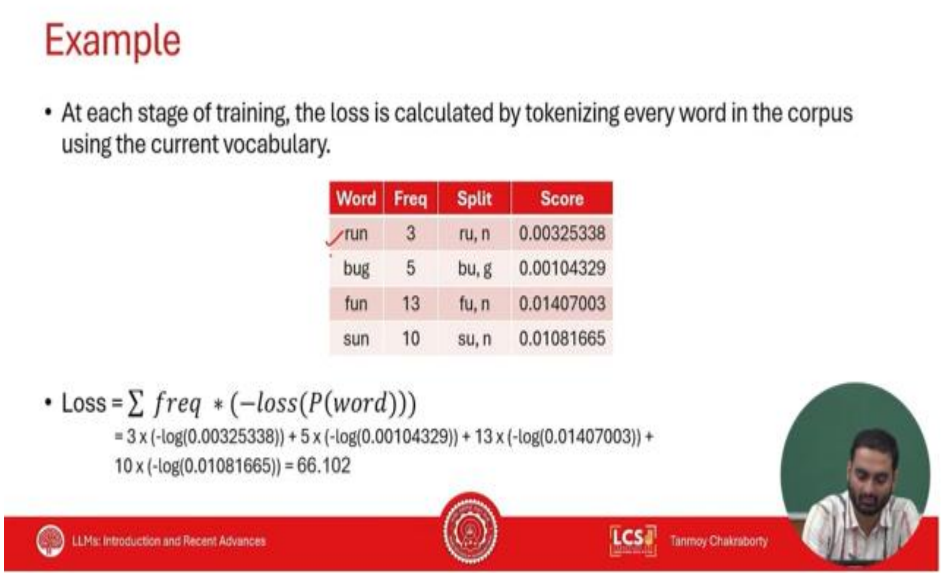
What we do then we now for every word present in the training set We measure the probability of a specific segmentation as I mentioned a particular word can be split in different ways right For example, the word run can be split into r, u, n or ru, n or r, un.

Remember r, u, n, ru, un these are all vocabulary entries already present in the initial vocabulary. So the word r, u, n can be split in three different ways. Now let us look at the probability of each of these individual splits. The probability of this split is as I mentioned this is unigram probability meaning all the unigrams are independent of each other. So probability of R, U, N is probability of R times probability of U times probability of N.

And how do we get this PR, PU, PN? We will get PR, PU, PN from here. This is PR, this is PU, this is PN. So this times this times this. This is going to be the probability for this split. For the signal split which is RU, N, RU and N, the probability will be PRU times PN. PRU will come from here. PU will come from here. Right so this is the probability for this split. For the last split which is r, un the probability will be pr times pun again we go back to the table and we will see the probability is this one. For every word I look at all possible splits with respect to the vocabulary and their corresponding probabilities. Now from this, we identify the best split, right, according to this probability. So the best split here can be, so look at this one, these and these both are highest and the same, right.

So let's say we randomly choose this one, this one, R U, N, right, as the best split. So for the word R U, N, so far the best split is RU, N. Now think about it. This is very exhaustive by the way. It's not a very computationally efficient method, right? For every word coming up with all possible splits based on the vocabulary, it's quite tedious. So what we use, we use this Viterbi algorithm.

I will not discuss Viterbi here, but you go back and check Viterbi is a very standard method that we generally used in sequence modeling and many other places. So we use Viterbi algorithm to make this the entire computation efficient. But at the end of the day, the end goal is to measure the probability for every split.



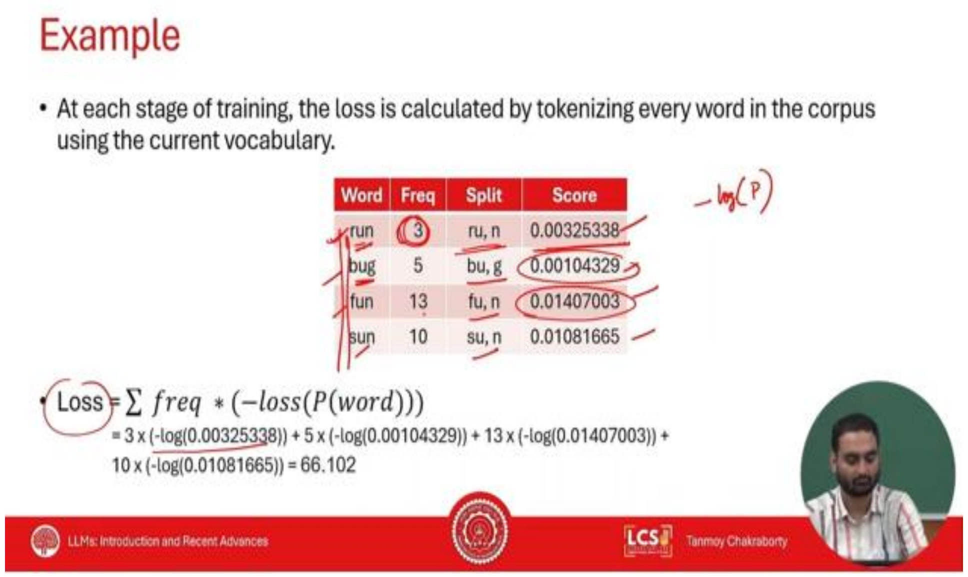
Example

- 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

LLMs: Introduction and Recent Advances — Tanmoy Chakraborty



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

$-\log(P)$

- 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

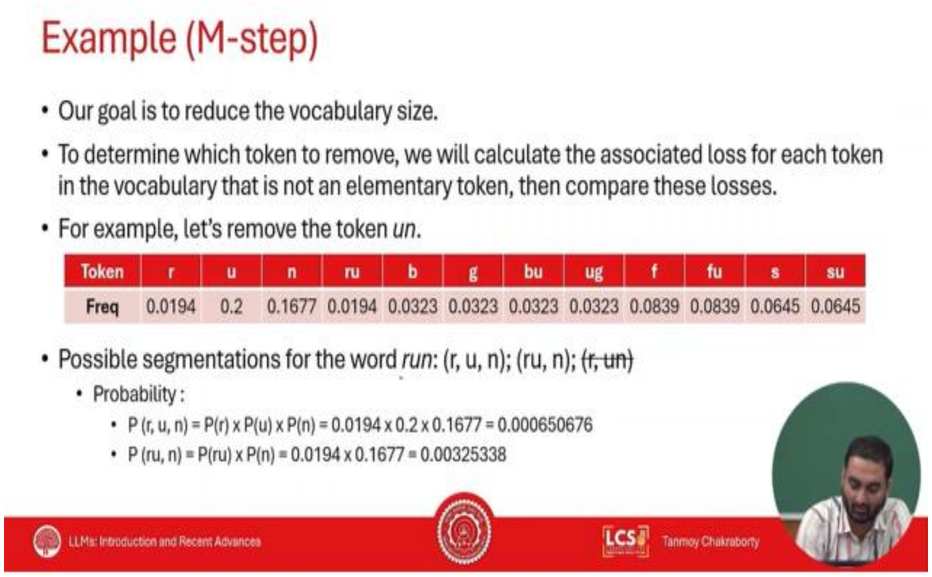
LLMs: Introduction and Recent Advances — Tanmoy Chakraborty

So for the word run, right for the word run whose frequency is 3 the base split that we identified is ru, n and the corresponding probability is this one. Similarly for the word bug which is present in the training set let us assume that the base split that we identified is bu, g and this is the score for fun fu, n this is the score for sun su, n this is the score.

Okay so for every word in the trading set we identify the best split and the corresponding probability. Then we compute a loss. Okay what is the loss? Here loss is the log loss very simple log loss for every word I know the probability right I basically take the log of the probability right and minus so minus log minus log of this probability Why minus? Because this probability ranges between 0 to 1. Log of that probability will always return minus. Therefore, minus and minus will cancel out. So for every word I compute minus log loss and since the first word run is present 3 times, so this will be 3 times the corresponding minus log loss.

Similarly for the word bug, log minus log of this times 5. For the word fun, minus log of this times 13 and so on and so forth. So this log loss essentially is the loss for the entire corpus of the training set. Because these words are taken from the training corpus.

Essentially when we are combining everything in the log loss, it is basically the log loss of the entire training corpus. Now the current log loss is 66.102.

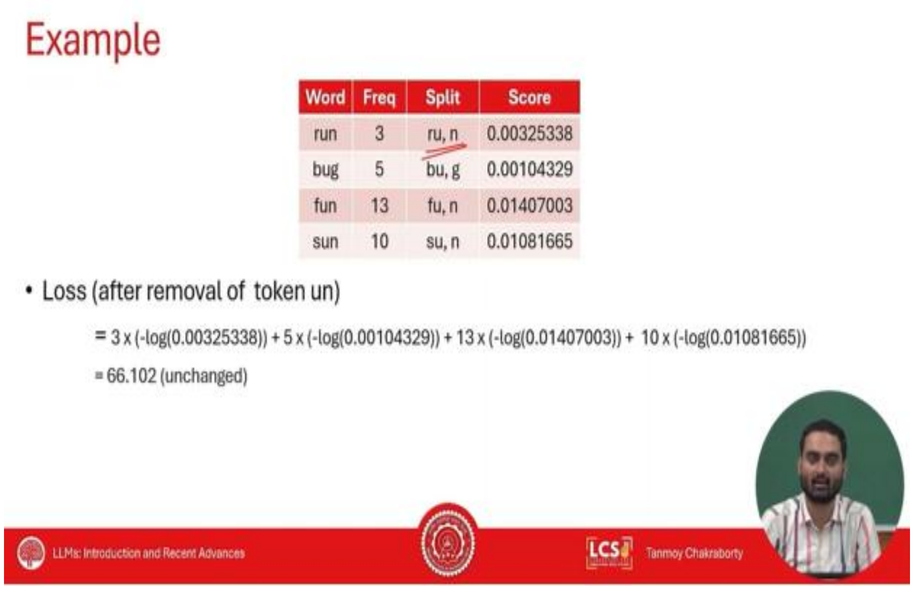


Example (M-step)

- 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)~~
  - 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

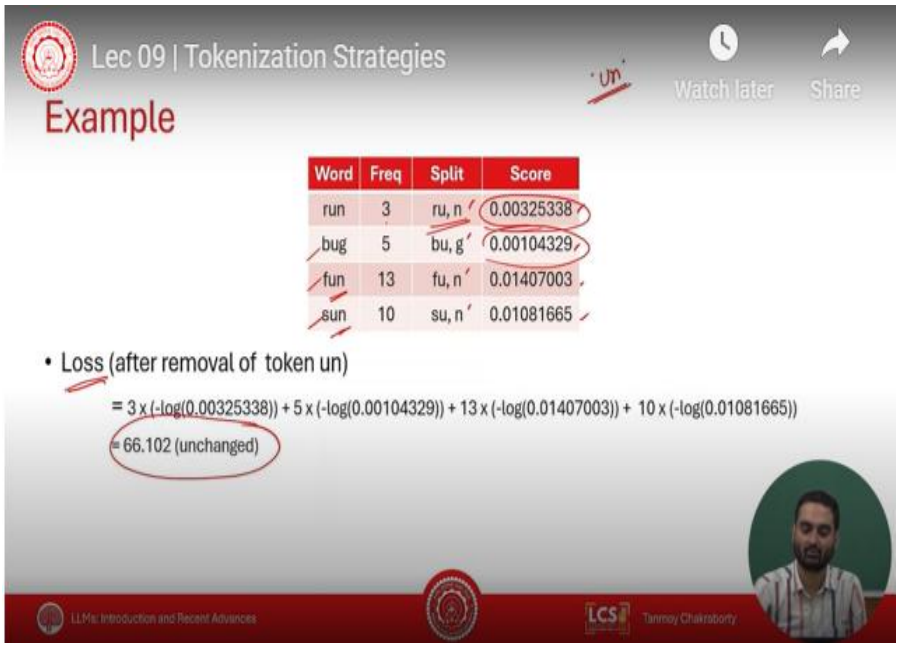
- Loss (after removal of token un)

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

  = 66.102 (unchanged)

LLMs: Introduction and Recent Advances

LCS2 Tanmoy Chakraborty



Lec 09 | Tokenization Strategies

Watch later Share

'un'

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)

LLMs: Introduction and Recent Advances

LCS2 Tanmoy Chakraborty

As I mentioned we will use EM algorithm right, if you do not know what is EM let's forget about EM, let's assume that there are two steps M step and E step okay. So what is the goal now? My goal is to remove a vocabulary entry which is not useful.

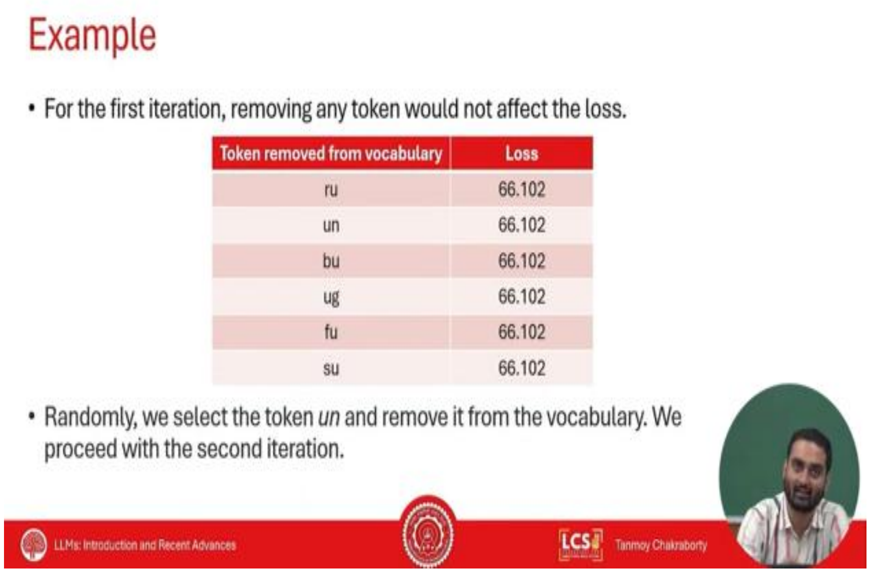
Already the vocabulary is significantly large. We started with a large base vocabulary. Then at every stage we identify the token in the vocabulary which is not useful. How do we do that? Let's see. So far, this is my token frequency table based on the vocabulary. In the vocabulary, I have R, U, N, RU, blah, blah, blah.

And these are the corresponding probabilities, right? Unigram probabilities that I mentioned earlier. This table is something I already showed earlier. Let us assume that I want to remove UN. I want to remove the token UN from the vocabulary. If I remove the token UN, so for the word RUN, remember initially I mentioned that there were three splits, possible splits. One was R,U,N, the other was RU, N, the other was R, UN, right? So this split will not be possible then because UN, if we remove UN, this split will not be possible.

Right so there are only two splits that are possible r u n and ru and n, right. Then for these two splits i measure the probability in the same way right. Probability of r, probability of u, probability of n based on this table, right. Probability of ru, probability of n Now out of these two splits which one is the best? This is the best right. So for r u n, Now, so for RUN, the split that is the best at this moment is RU, N if we remove UN. Okay similarly for bug, for fun, right for sun, look at here, for fun, one split could have been f comma un right.

But since we removed u n we just need to discard that option. So then maybe the base split will be fu comma n right similarly here s u comma n and so on. So un is a possible token from the vocabulary that I want to remove right. If I remove these are the resultant base splits and these are the corresponding probabilities right Again what is the total loss? The total loss of the corpus with respect the corpus is the frequency times minus log loss so this log of this minus log of this times 3 minus log of this times 5 and so on and so forth.

So this is the total log loss now after removing un. Now remember this number 66.102. Earlier it was also 66.102 when the token un was present in the book of law. After removing it, it is still 66.102 meaning removal of the token un doesn't affect the log loss at all. So I can safely remove this token from the vocabulary.



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

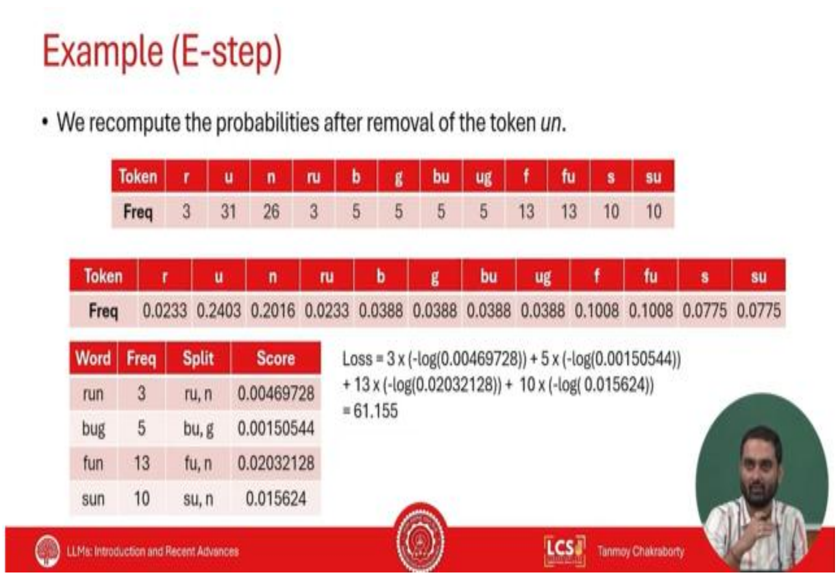
- Randomly, we select the token *un* and remove it from the vocabulary. We proceed with the second iteration.

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

In fact, if you do this calculation, you will see that there are many other candidate tokens that can be removed. UN was one of the possibilities. RU can also be a possibility. BU can also be a possibility. UG can also be a possibility. In fact, removal of each of these tokens from the vocabulary will not change the loss at all.

Right. So now out of this, which one I should remove? I can choose one randomly, one of these randomly. Right. In fact, what the algorithm does in practice, it removes 5 to 10% or 10 to 20% of vocabulary entries at one go.

Right. If you see all these tokens are useless, you remove them at a time. Okay. So but let's assume that in our case we remove this one by one right. So let's say we select un and we remove un right and then we basically modify the table.



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

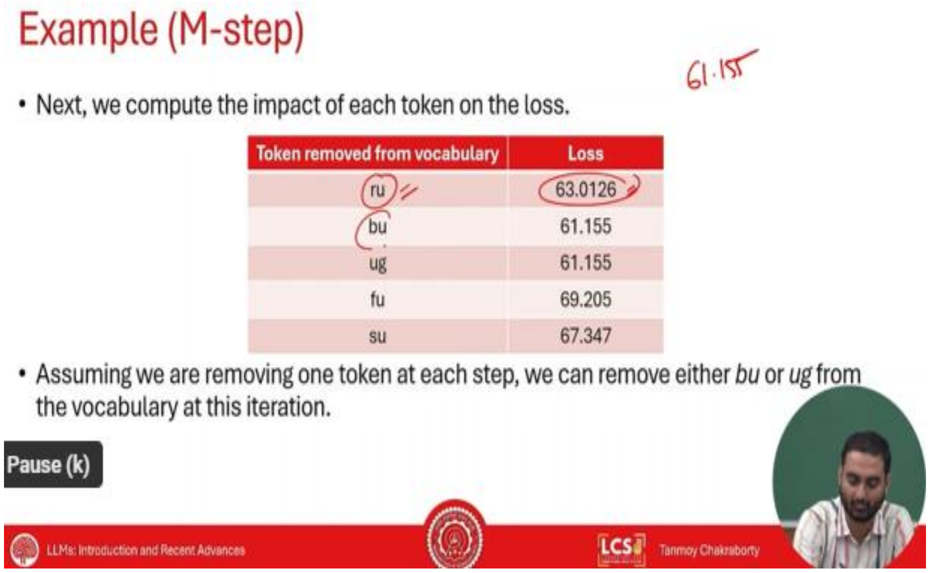
LLMs: Introduction and Recent Advances

Tanmoy Chakraborty

So in the maximization stage, what we did, we maximized instead, we removed basically a token. We aim to remove a token whose removal will not affect the loss or whose removal will affect the loss, least affect the loss.

Now, once we removed UN, this is the modified table now. This is the vocabulary R U R U and R U look at here though the token un is not present at all and the corresponding frequency. Since you already removed the token the corresponding bigram probability will also change right, the corresponding bigram probability will also change okay. So now the resultant bigram probability is this one Now let us look at the split again for these words RUN, RUN, BUG, FUN and SUN.

Let us assume that the base split for RUN is RU, N. For BUG, BU, G, For FUN, FU, N, For SUN, SU, N. And these are the corresponding probabilities based on this table, okay. Again I measure the loss, the total loss, the total loss is the frequency times minus log loss and now the current log loss is 61.155, okay.



Example (M-step)

61.155

- 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.

Now let's see which one i want to remove if i remove ru right so among these tokens this is the vocabulary if i remove ru the loss will be 63.0126 remember this number 61.6155 If I remove RU, this will be the loss. If I remove BU, this will be the loss. If I remove UG, this is the loss, and so on and so forth. Now look at here. The token BU, if we remove this token BU or UG, it will not affect the loss at all. So I can remove either both BU and UG, or I can remove either BU or UG. So I keep on removing, you know, vocabulary entries in this way.

And then I stop when I see a certain, you know, a certain vocabulary size each list. So let me recap what we discussed here in this specific algorithm, Unigram language model tokenization. So I am given the training set. From the training set, I did all this pre-tokenization.

I identified all the words. From the words, I then identified the base vocabulary. The base vocabulary in this case is the longer one compared to BPE and WordPiece. Because here, the aim is different. Here, the aim is to basically remove the vocabulary entities which are not useful. So what we can do? We can either, you know, you can either add all possible characters, you know, subwords that you can think of out of the words present in the training set, right, and make a larger kind of vocabulary. Or you can run BP or WordPiece, not WordPiece, maybe BP, right, to identify the initial vocabulary.

Once it is ready, then what we do? Then for every word present in the training set, I identify the base split for that word. How do we identify the base split? Base split, so for a word there can be multiple possible splits. For every split, why measure the probability? The probability is essentially the multiplication of all the individual probabilities of tokens present in the specific split. So for every word, I identify the base split.

For every word in the training set, I identify the base split. Now for the entire corpus, meaning for all the words present in the training set, I basically accumulate all the loss together. So what is the loss? Loss is a log loss. So minus log of the probability of the split times the frequency of the word. So I measure the total loss with respect to the corpus and that's my standard now. Then from the vocabulary, I identify that token whose removal will not impact the loss or least impact the loss.

I can remove one such token from the vocabulary at a time or multiple tokens from the vocabulary at a time. Once I remove this, again I modify the frequency table, right, meaning the unigram table, right, based on the modified count and so on and so forth. Then again based on the modified count for every word present in the trading set, I again identify the base split, right, again cumulative loss, right, and again we choose the best candidate token whose removal will not impact the overall loss. And we keep on doing this thing. We keep on removing tokens from the vocabulary, unless it basically gets synced into a certain size, predefined size, that you already determined at the beginning.

So this is a unigram language model tokenization, which is used in sentence-based embedding. These are the standard tokenization methods, BP, WordPiece and SentencePiece, Unigram language model. But there are lots of recent advances in tokenization strategies. Tokenization is very important to handle low resource languages, for example, you know, those languages which are not at all present in the training time, right? The model is completely unaware of those languages, for example. Let's say you don't have the resource to compute, to pretend the model from scratch for a specific language, right? We can use this kind of tokenization techniques to deal with low resource languages. With this I stop here and next day we basically move to the next topic of this course. Thank you.