Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

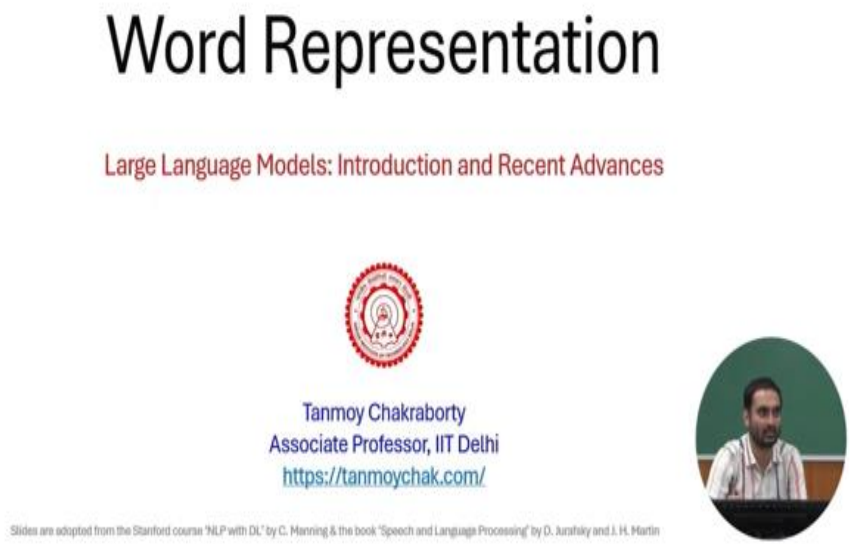
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 7

Word Representation: Word2Vec & fastText

Hello everyone. So in this lecture we will talk about word representation. We finished the language model, the language modeling part of it, right. We discussed statistical language models, different types of smoothing techniques and how to evaluate a language model. Perplexity we discussed in details, what is perplexity, how it is linked to entropy and how you can use perplexity as an intrinsic measure to evaluate the performance of a language model.

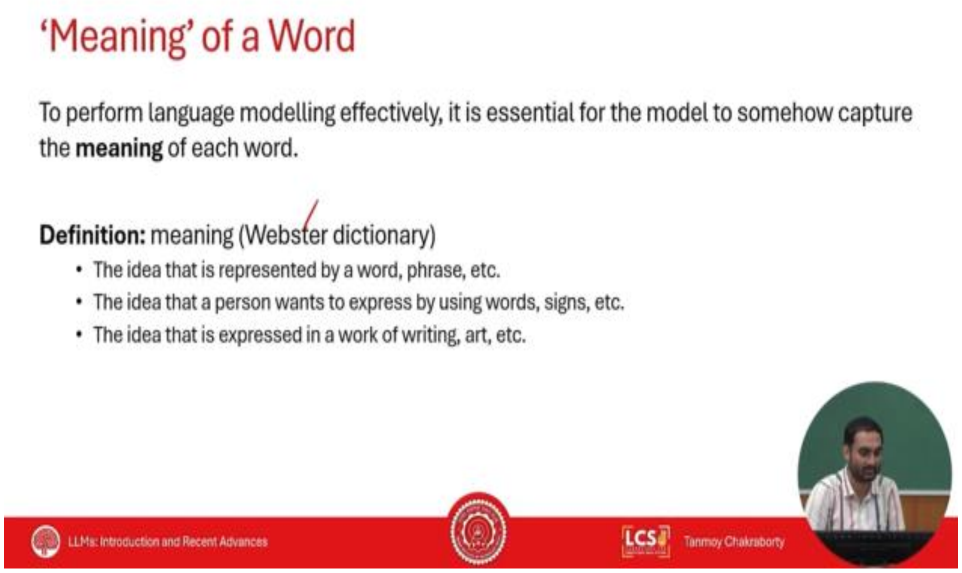


Word Representation

Large Language Models: Introduction and Recent Advances

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

Slides are adopted from the Stanford course 'NLP with DL' by C. Manning & the book 'Speech and Language Processing' by D. Jurafsky and J. H. Martin



'Meaning' of a Word

To perform language modelling effectively, it is essential for the model to somehow capture the **meaning** of each word.

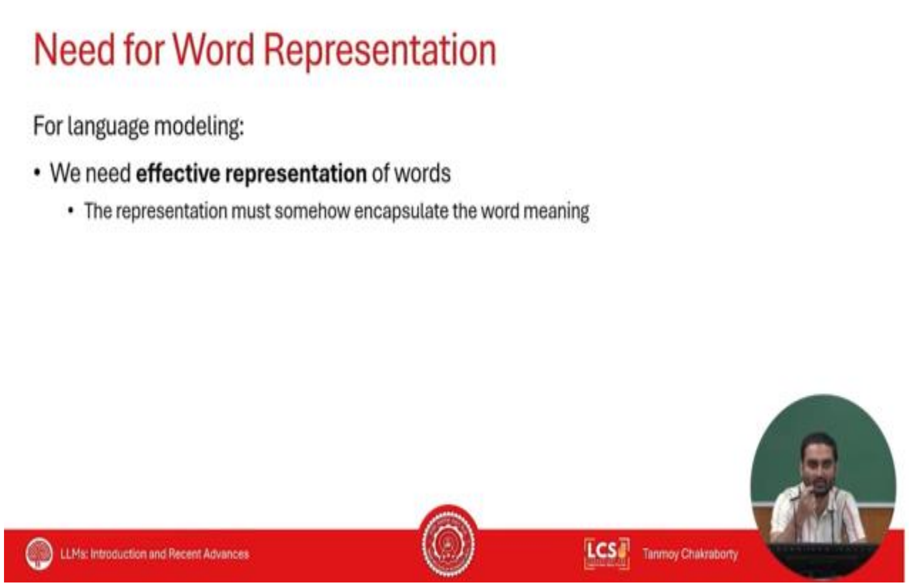
Definition: meaning (Webster dictionary)

- The idea that is represented by a word, phrase, etc.
- The idea that a person wants to express by using words, signs, etc.
- The idea that is expressed in a work of writing, art, etc.

So now we are moving towards the semantics.

Remember the different layers of NLP I mentioned, morphology, syntax, semantics. and so on and so forth. So here we start talking about semantics, right. The first thing in any semantic understanding of language, right, what we do is to represent a word or represent a sentence, represent a token, right.

And we will discuss in this class, we will discuss old school methods to represent a word and then we will see how modern days we represent a word using word vector, okay. Let's see what a dictionary says about a meaning of a word. If you look at the Mariam Webster dictionary, online dictionary, it says that a meaning is essentially the idea that is represented by a word or a phrase, right, etc. It can be an idea that a person wants to express by using different words, signs, and so on and so forth.

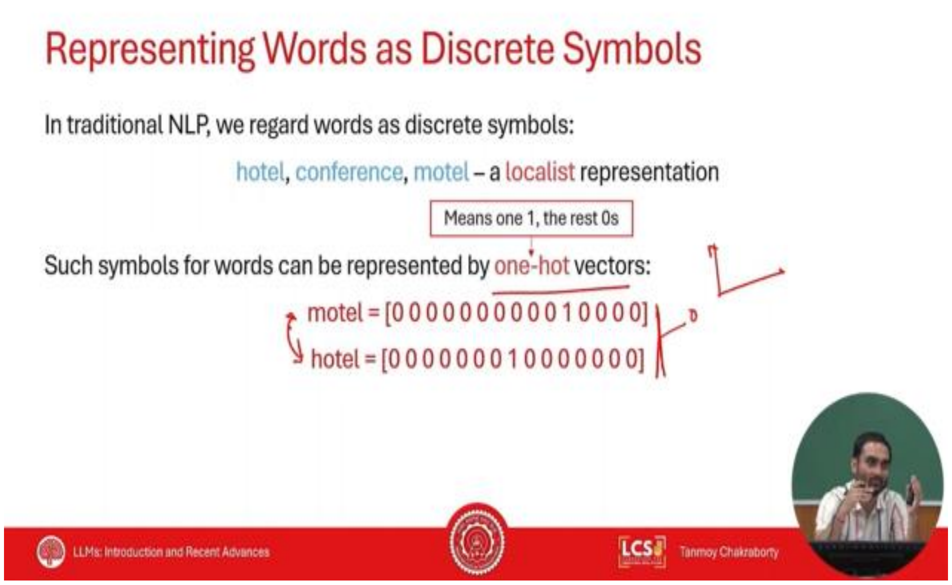


Need for Word Representation

For language modeling:

- We need **effective representation** of words
  - The representation must somehow encapsulate the word meaning

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty



Representing Words as Discrete Symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

Effective representation of words meaning has been one of the major challenges because we use a word in different contexts.

I mentioned about the word bank. For example, apple, mouse, these words are used in different contexts with different meanings and so on and so forth. So it is very tough to come up with a single representation or single meaning of a word. If you think that let me curate a dictionary where for every word I have a single meaning and whenever I need the meaning of that word I just retrieve that meaning from the dictionary and then I basically solve the problem. But this is not going to work because of the complicacy.

So far, if you look at the old school NLP techniques, the way people used to solve the semantics problem is that they used to look at documents and from documents, there were human annotators who were employed to read thousands of thousands of documents and then they used to create big ontology like a word net, right? where entities are nodes and entities are linked through different relations. One sense of an entity is linked to another sense of an entity and these relationships can be of different types, holonym, meronym, synonym, antonym, hyponym, hyponym and so on and so forth. And then whenever you are interested in understanding a meaning of a word, you look at the ontology, you retrieve let's say, the neighboring words of that given word and you try to guess the meaning of it. But then we will realize that this might not be a good approach. I will tell you why.

And then what is needed is that we essentially need some sort of vectorized representation of a word. Right, the computer does not know characters. Computer knows numbers right. So we all know ASCII values right, So you can say okay, why do we need a complicated vector representation let me take a word, and let me take the character, let me take the ASCII value of every character, right, and then we concatenate. The problem is that words have different lengths.

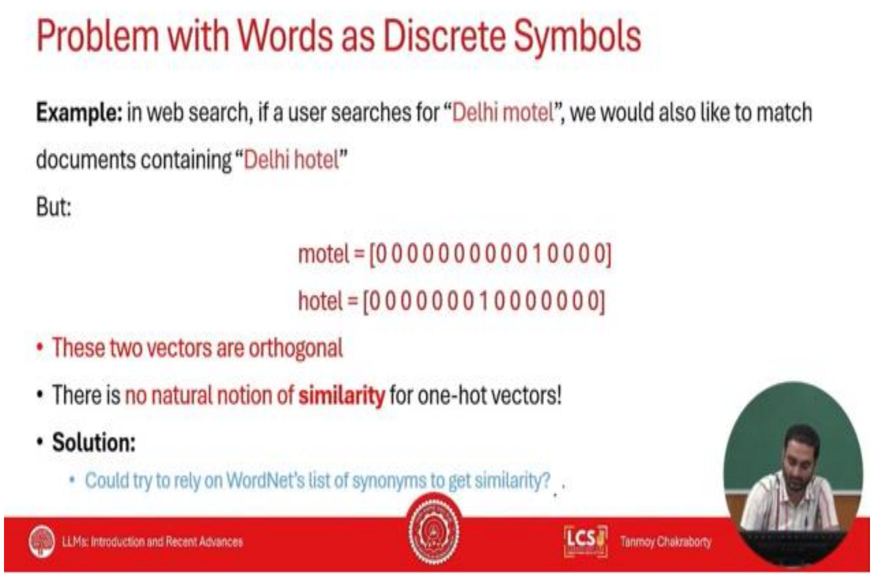
So if you just concatenate all the ASCII values, then the length of the vector would be different for different words. And there are also other problems. I'll tell you the other problems. So the simple way to represent a word using a vector is something called one-hot encoding. Right one hot vector.

What is a one hot vector? Very simple idea, let's say you already have a dictionary, where all the unique words are listed right and each word has an id. Right let's say the word a has an id, word n has an id, dot dot dot word zebra has an id, word zoo has an id and so on and so forth, id-1, id-2, id-3 and so on and so forth right and let's say the size of the dictionary is 1 million. So what we do in one hot encoding one hot vector is that for every word I create a vector of size 1 million right and let's say if the id of that word is 5 right what I do I will add 1 at the fifth position of the vector and the remaining positions will be marked as 0. So one position corresponding to the ID will be marked as 1 and the remaining position will be 0. Very simple idea.

And it also makes sure that different words have different representations. All the representations are unique. But there are many problems. There are multiple problems here. The first problem is that now all these vectors are orthogonal.

So if we take the dot product between two vectors, you will always get 0. Now if you think of if you take a synonymous two synonymous words like say motel and hotel motel and hotel they are kind of same right. If you take the dot product of these two vectors you see the dot product would be 0. It means that they are basically orthogonal to each other although their semantics is almost the same ok. So why not encoding is not a good approach.

The second problem here is that as the dictionary size increases, the vector size will also increase. In fact, you may need to recompute the vector again and again. And there are multiple problems.



Problem with Words as Discrete Symbols

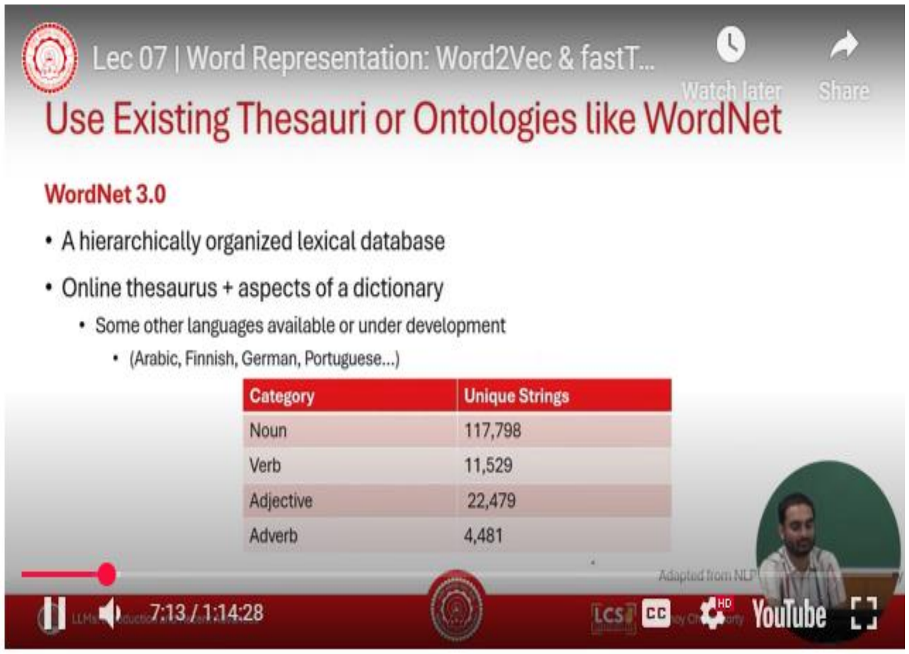
Example: in web search, if a user searches for "Delhi motel", we would also like to match documents containing "Delhi hotel"

But:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

- These two vectors are orthogonal
- There is no natural notion of **similarity** for one-hot vectors!
- **Solution:**
  - Could try to rely on WordNet's list of synonyms to get similarity?


Lec 07 | Word Representation: Word2Vec & fastT...

Watch later Share

Use Existing Thesauri or Ontologies like WordNet

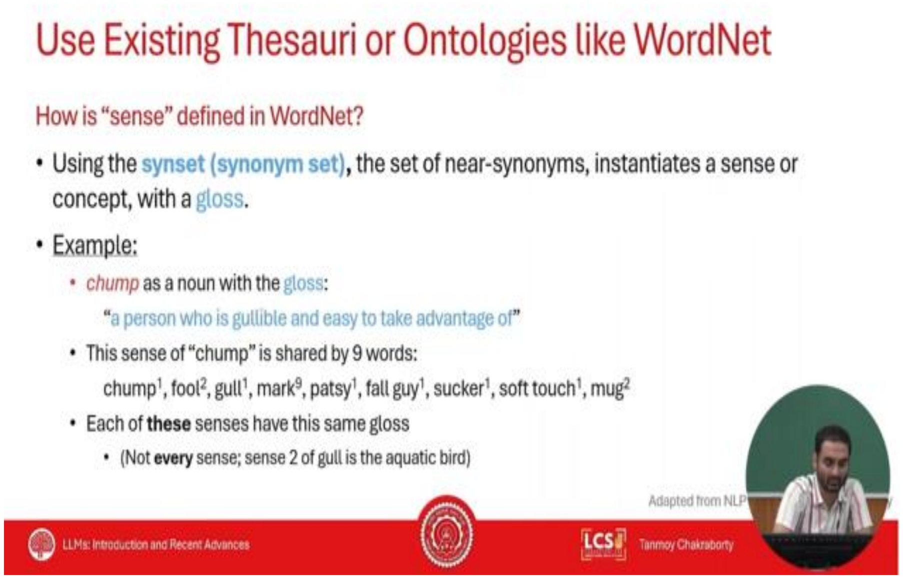
WordNet 3.0

- A hierarchically organized lexical database
- Online thesaurus + aspects of a dictionary
  - Some other languages available or under development
    - (Arabic, Finnish, German, Portuguese...)

Category	Unique Strings
Noun	117,798
Verb	11,529
Adjective	22,479
Adverb	4,481

Adapted from NLP

7:13 / 1:14:28



Use Existing Thesauri or Ontologies like WordNet

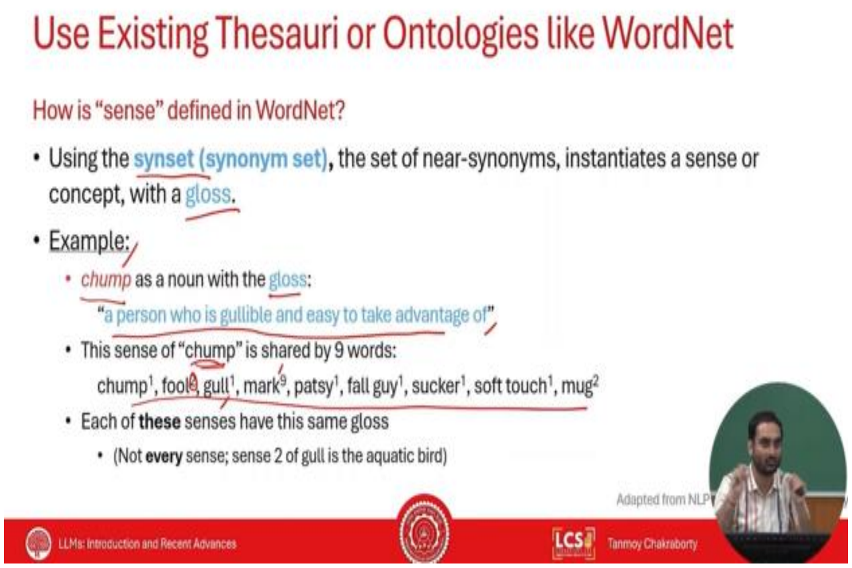
How is "sense" defined in WordNet?

- Using the **synset (synonym set)**, the set of near-synonyms, instantiates a sense or concept, with a gloss.
- Example:
  - *chump* as a noun with the gloss:
    "a person who is gullible and easy to take advantage of"
  - This sense of "chump" is shared by 9 words:
    chump^1, fool^2, gull^1, mark^9, patsy^1, fall guy^1, sucker^1, soft touch^1, mug^2
  - Each of **these** senses have this same gloss
    - (Not **every** sense; sense 2 of gull is the aquatic bird)

Adapted from NLP

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty


Use Existing Thesauri or Ontologies like WordNet

How is "sense" defined in WordNet?

- Using the synset (synonym set), the set of near-synonyms, instantiates a sense or concept, with a gloss.
- Example:
  - *chump* as a noun with the gloss:
    "a person who is gullible and easy to take advantage of"
  - This sense of "chump" is shared by 9 words:
    chump[1], fool[2], gull[1], mark[9], patsy[1], fall guy[1], sucker[1], soft touch[1], mug[2]
  - Each of **these** senses have this same gloss
    - (Not **every** sense; sense 2 of gull is the aquatic bird)

Adapted from NLP

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty

So if two words share mutually similar semantics, this kind of 1-0 encoding may not be a good solution for that. So what is the next solution? The next solution is to look at the WordNet.

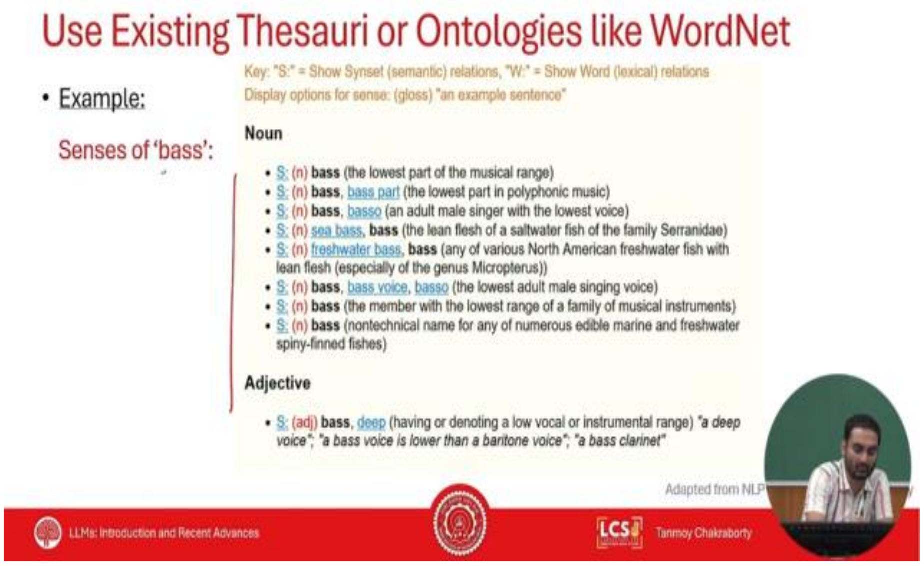
As I mentioned earlier, WordNet is a... semi-automatically generated ontology where you see many nouns, verbs, mostly nouns, verbs, adjectives and adverbs. They are linked through different relations.

Every word in the word net is represented by something called a synset. Every word has a synset. What is synset? Synset is a synonymous set or synonym set. So each word, each sense of a word, not every word. So remember every word has multiple senses.

A word can have multiple senses. So every sense of a word is represented by a synonym set or synset. And the description of the synset is called the gloss. Okay for example let's look at the word chump okay the gloss or the description of this word is a person who is gullible and easy to take advantage of. So this is the description of the sense one sense of the word chump okay and this description is written manually by the way okay it's a huge manual effort and What are the different senses of the word chump? Now if you look at word net, word net will say that chump has these many senses.

One sense of chump is same as the second sense of fool. Now remember fool is also an entry in the word net. The word fool also has multiple senses. Sense 1, sense 2, sense 3, sense 4 and every sense has a gloss. So, the particular sense of the word charm is linked to the second sense of the word fools, to the first sense of the word gull, to the ninth sense of the word mark and so on and so forth.

Now, imagine the amount of effort. Every word has multiple senses. The senses are linked, not the words. The senses are linked.



Use Existing Thesauri or Ontologies like WordNet

- Example:

Senses of 'bass':

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **bass** (the lowest part of the musical range)
- S: (n) **bass**, bass part (the lowest part in polyphonic music)
- S: (n) **bass**, basso (an adult male singer with the lowest voice)
- S: (n) sea bass, **bass** (the lean flesh of a saltwater fish of the family Serranidae)
- S: (n) freshwater bass, **bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- S: (n) **bass**, bass voice, basso (the lowest adult male singing voice)
- S: (n) **bass** (the member with the lowest range of a family of musical instruments)
- S: (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Adjective

- S: (adj) **bass**, deep (having or denoting a low vocal or instrumental range) *"a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"*

Adapted from NLP

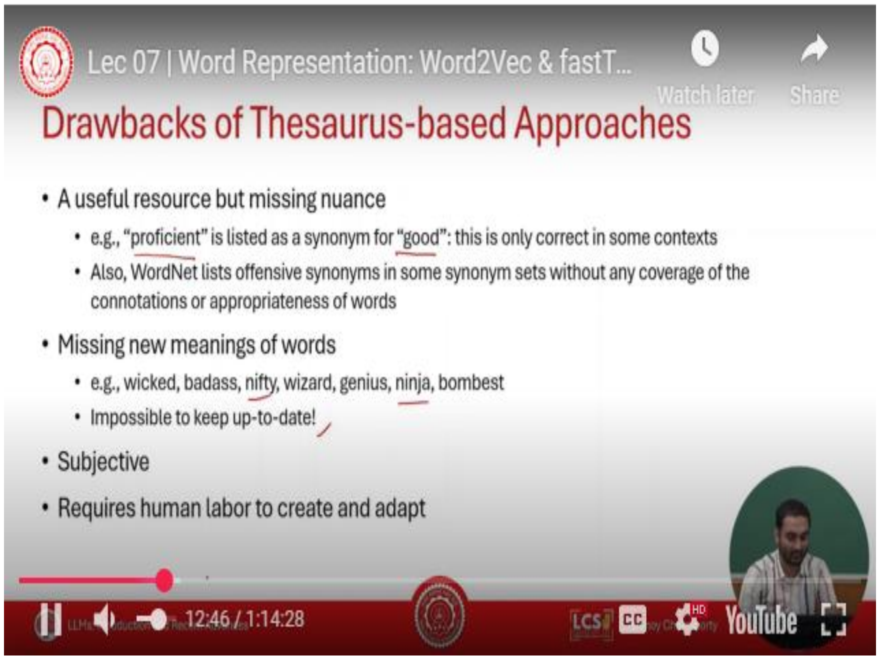
LLMs: Introduction and Recent Advances

LCS

Tanmoy Chakraborty

Okay, so now this is one entry in the WordNet, right? The word BASS, B-A-S-S.

If you look at the word net ontology, you see that the word bass has multiple senses. The first sense is bass is a musical, some sort of a musical range, right? Bass can also be a fish, right? Bass can also be some sort of singing, male singing voice, bass guitar, right? So, it has multiple senses, of course the pronunciations are different, but the surface word is the same. So, this is the first sense of bass, this is the second sense of bass, third sense of bass and so on and so forth. So, and if you look at every sense there is this gloss, these are different glosses.



Drawbacks of Thesaurus-based Approaches

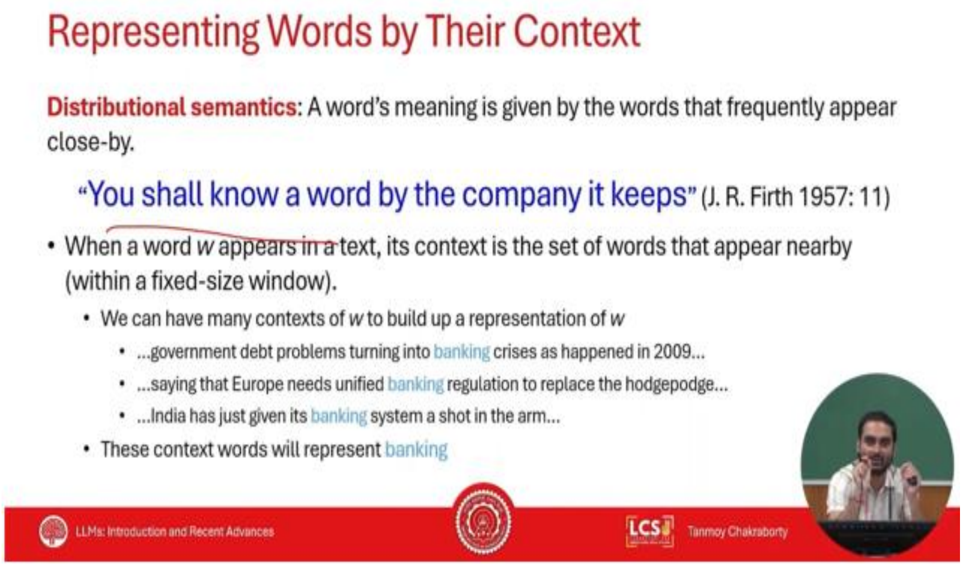
- A useful resource but missing nuance
  - e.g., "proficient" is listed as a synonym for "good": this is only correct in some contexts
  - Also, WordNet lists offensive synonyms in some synonym sets without any coverage of the connotations or appropriateness of words
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
  - Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt

So now one can say that you know we do not need 1-0 encoding, I will just look at the word net, fetch the corresponding sense and that will be sufficient.

So what are the problems with this kind of thesaurus based approach? Thesaurus means resource, thesaurus based approach or ontology based approach. The first problem is that there may be other meanings which are not captured into the WordNet version. WordNet has multiple versions by the way. For example, the word proficient and the word good, they are synonymous in certain contexts, but they are not synonymous in all the contexts. Secondly, it's very difficult to add newly coined words You need manual effort.

For example, ninja, nifty and so on and so forth. These kind of newly coined words, if you want to add them into the ontology, you need to actually study a lot about their usage. What are the senses, where they are used and so on and so forth. And then you also have to remember what are the other senses that can be linked to this sense of nifty for example. So, it is actually very complicated.

Up to date information is difficult to obtain. The other problem is that WordNet mostly considers noun, adjective, adverb and verb. The other part of speech words are not covered appropriately. This is also subjective because if you ask one annotator they may feel that, okay, this is not the sense, if you ask another annotator, they may feel that no, this is the sense and so on and so forth. It depends on who is annotating, right, it requires a lot of human labor.



Representing Words by Their Context

Distributional semantics: A word's meaning is given by the words that frequently appear close-by.

"You shall know a word by the company it keeps" (J. R. Firth 1957: 11)

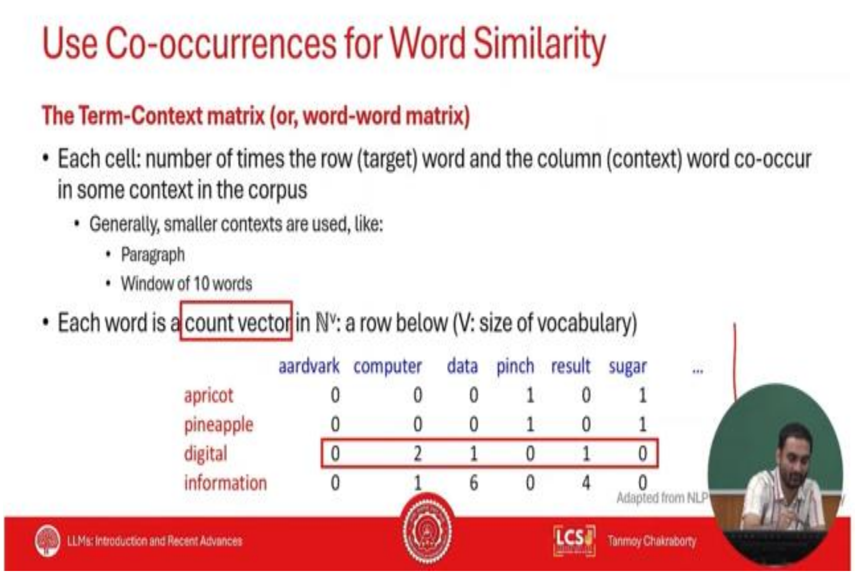
- When a word *w* appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
  - We can have many contexts of *w* to build up a representation of *w*
    - ...government debt problems turning into banking crises as happened in 2009...
    - ...saying that Europe needs unified banking regulation to replace the hodgepodge...
    - ...India has just given its banking system a shot in the arm...
  - These context words will represent banking

To address this we will use the notion of distributional semantics. If you remember the second lecture where we talked about NLP, I mentioned three ways to understand semantics of a word. What are those three ways? Ontological semantics, then decomposition semantics, distributional semantics. So distributional semantics is the premise based on which most of the existing what representation techniques are proposed. And what is the philosophy behind the distributional semantics? J. R. Forth in 1957, he said this famous quote that you shall keep a word by the company, you shall know the word by the company it keeps. Okay, so now based on this philosophy, again if you go back to old school NLP, right, people came up with many, many approaches, right. One simple approach is to look at the documents, right? You read the sentence in the document and you look at the neighboring words of every word and from the neighboring words, from the neighborhood, you try to imply or you try to guess the meaning of that word, right? For example, the word banking. If you look at this three sentences here, the first sentence government debt problems turning into banking crisis as happened in 2009. Right here essentially you are talking some sort of crisis, right? Where the next one saying that Europe needs unified banking regulation.

Here you are talking about some sort of regulation, rules and regulations. Third one, India has just given its banking system a shot in the arm. So this is more of a banking system. The literal meaning of banking system. So you look at the neighbors and from the neighbors you guess the meaning of that particular word.

Count-based Methods

LLMs: Introduction and Recent Advances | LCS | Tanmoy Chakraborty



Use Co-occurrences for Word Similarity

The Term-Context matrix (or, word-word matrix)

- Each cell: number of times the row (target) word and the column (context) word co-occur in some context in the corpus
  - Generally, smaller contexts are used, like:
    - Paragraph
    - Window of 10 words
- Each word is a count vector in $\mathbb{N}^V$: a row below (V: size of vocabulary)

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Adapted from NLP

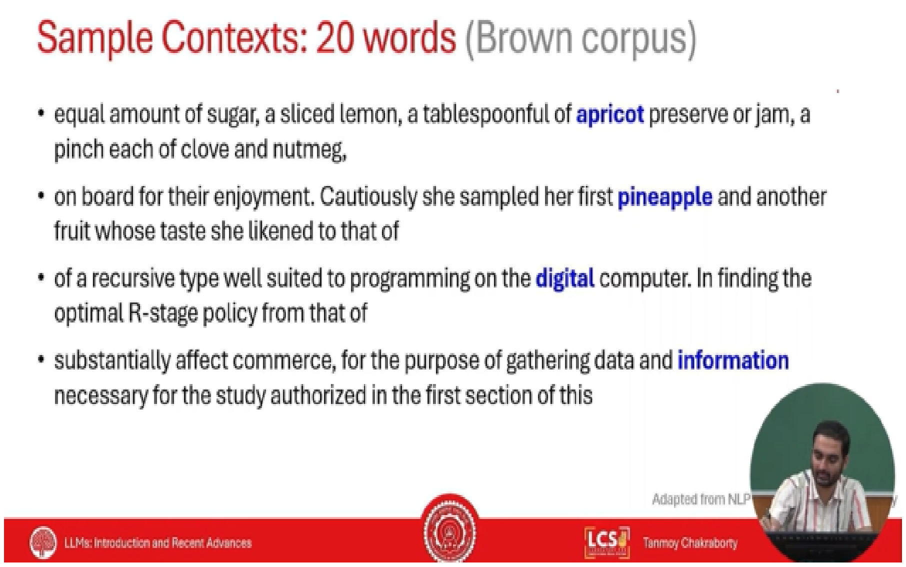
LLMs: Introduction and Recent Advances | LCS | Tanmoy Chakraborty

Now how do we systematically do it? We systematically do it using a count based approach. A very simple count based approach. But the idea is that, let's say you have multiple documents. Okay, you have a bunch of documents, you have corpora and you see for how many times this particular word appears in a document, how many times this word appears in other documents and so on and so forth. It's just a count, a bag of word kind of approach.

Okay, let's look at this example. This is a simple example where we will basically create this kind of matrix. This matrix is called term context matrix. What is a term context matrix? Here rows are individual terms, individual words, individual tokens and columns are also individual tokens. And you see how many times, let's see here, let's say this entry. Says how many times the word computer appears as a neighbor of the word apricot, okay.

Some sort of bigram count table, very similar to bigram count table, but what is the difference here? In case of bigram, these two words should appear side by side, wi minus one wi, right? But here, when I look at the context, I first decide the size of the context. Okay let's say I have a context size of 10 I look at 10 words at a time right and I'll see how many times the word apricot and computer appear together right in the same context right and that's going to be the count here. So term context and this is a square matrix right So from the term context matrix, now if you look at every row, okay, now you can think of every row as a vector, right? Let's say first row is a vector for the word apricot, right? When apricot is used as a term, right? You can also... And an apricot will also appear in a row, right? Here also you will see an entry for apricot, right? And there will be a column corresponding to the word apricot, right? So that column can be treated as a context vector for the word apricot. So row can be treated as a term vector for the word apricot. Column can be treated as a context vector for the word apricot. Now it is up to you to decide which vector you want to consider. One option is that you take both the vectors and you sum them.

Other option is that you try with context vector as well as a term vector and see which one is useful. Very simple idea.

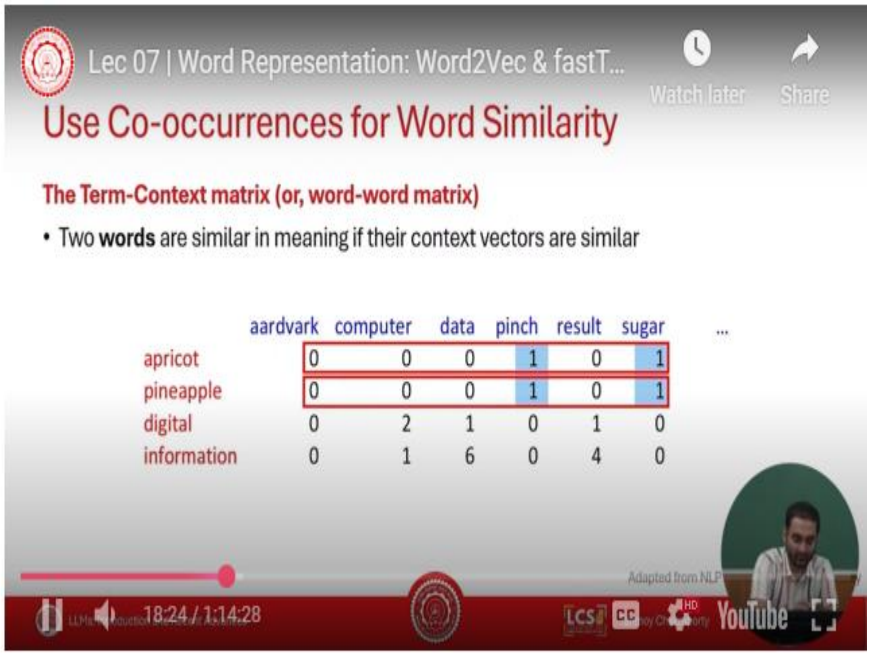


Sample Contexts: 20 words (Brown corpus)

- equal amount of sugar, a sliced lemon, a tablespoonful of **apricot** preserve or jam, a pinch each of clove and nutmeg,
- on board for their enjoyment. Cautiously she sampled her first **pineapple** and another fruit whose taste she likened to that of
- of a recursive type well suited to programming on the **digital** computer. In finding the optimal R-stage policy from that of
- substantially affect commerce, for the purpose of gathering data and **information** necessary for the study authorized in the first section of this

Adapted from NLP

LLMs: Introduction and Recent Advances   LCS2   Tanmoy Chakraborty



Lec 07 | Word Representation: Word2Vec & fastT...   Watch later   Share

Use Co-occurrences for Word Similarity

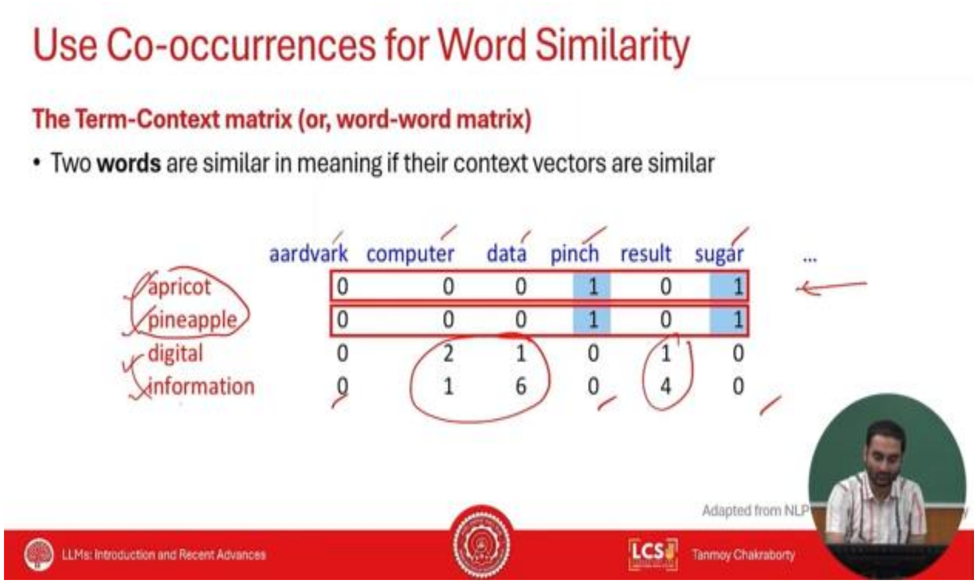
The Term-Context matrix (or, word-word matrix)

- Two **words** are similar in meaning if their context vectors are similar

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Adapted from NLP

18:24 / 1:14:28   YouTube



Use Co-occurrences for Word Similarity

The Term-Context matrix (or, word-word matrix)

- Two **words** are similar in meaning if their context vectors are similar

	aardvark	computer	data	pinch	result	sugar	...
apricot	0	0	0	1	0	1	
pineapple	0	0	0	1	0	1	
digital	0	2	1	0	1	0	
information	0	1	6	0	4	0	

Adapted from NLP

LLMs: Introduction and Recent Advances

LCS

Tanmoy Chakraborty

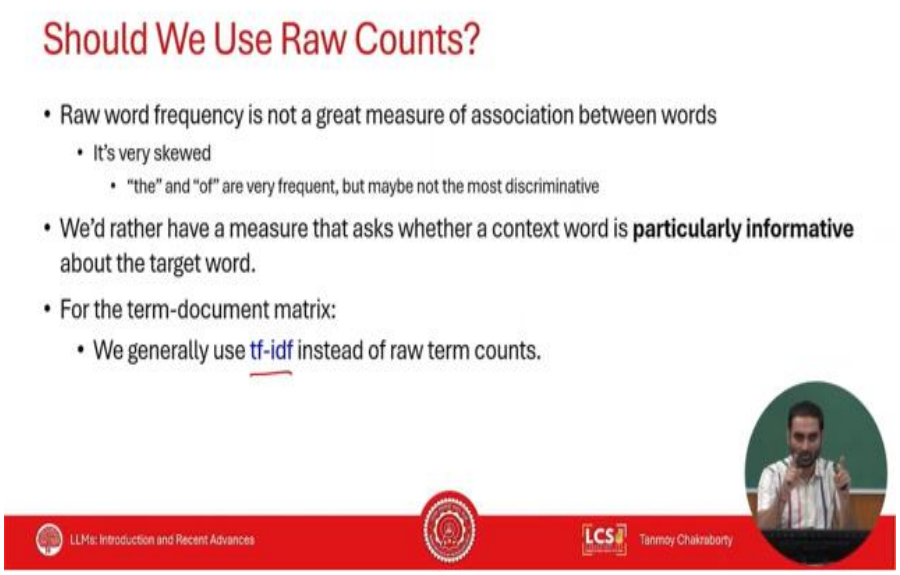
So let's say you are given this kind of documents. It's a stylized example.

There are multiple sentences. You scan through the documents and you create this kind of matrix. So now if you look at two entries here, let's say apricot and pineapple. You see both these words co-appear with similar context words pinch and sugar. Right. But they did not appear, co-appear with the context was artwork or computer data.

Right. So you can say that, OK, maybe these two words are similar. OK. Similarly, if you look at digital and information, you'll see both these words co-appear with the word computer and data. right also results, but they did not co-appear with the word adverb pinch and sugar. So, maybe these two words are similar right simple idea you look at you take every column you take a pair of columns you do some sort of dot product cosine similarity and then you figure out the similarity right.

So, here what are the problems? The first problem is that, again, this metric size can increase with increasing number of vocabulary terms. Second problem is that vector size will also be very large. If the vocabulary has, let's say, millions of words, then the size of the vector should be very, very large. And also, this matrix is sparse.

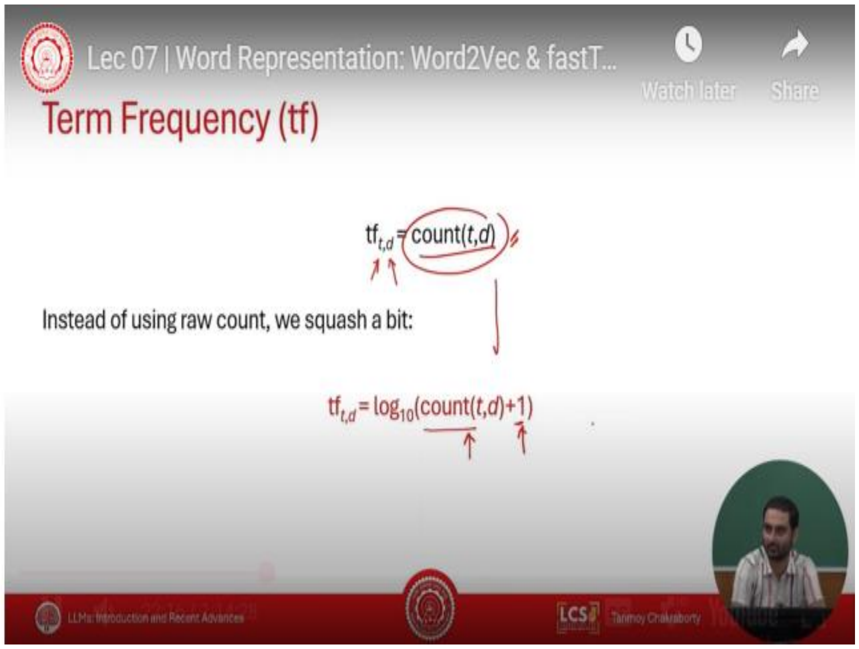
Most of the entries are zero. A sparse matrix. The fourth problem is that, We see stop words, frequently appearing words, particle, preposition, their corresponding entries will be very very high, obviously.



Should We Use Raw Counts?

- Raw word frequency is not a great measure of association between words
  - It's very skewed
    - "the" and "of" are very frequent, but maybe not the most discriminative
- We'd rather have a measure that asks whether a context word is **particularly informative** about the target word.
- For the term-document matrix:
  - We generally use tf-idf instead of raw term counts.

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty



Lec 07 | Word Representation: Word2Vec & fastT...
Watch later
Share

Term Frequency (tf)

$$\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$$

Instead of using raw count, we squash a bit:

$$\mathrm{tf}_{t,d} = \log_{10}(\mathrm{count}(t,d)+1)$$

LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

So we need some sort of normalization after this. So what we do here? So from a raw count of terms and context, now we move towards a slightly advanced concept called TF-IDF. So what is TF-IDF? In TF-IDF also we are going to create this matrix, remember this. TF stands for term frequency and IDF stands for inverse document frequency.

TF IDF okay, let's first discuss TF, so what is TF? Term frequency of a term T right with respect to a document D is the number of times that particular term appears in the document. So remember this is a tuple, this is a function of T, D You can't say that the term frequency of T is this. You have to say the term frequency of T with respect to the document D is this. Okay. So by the way, when I say document, a document can be a Wikipedia document.

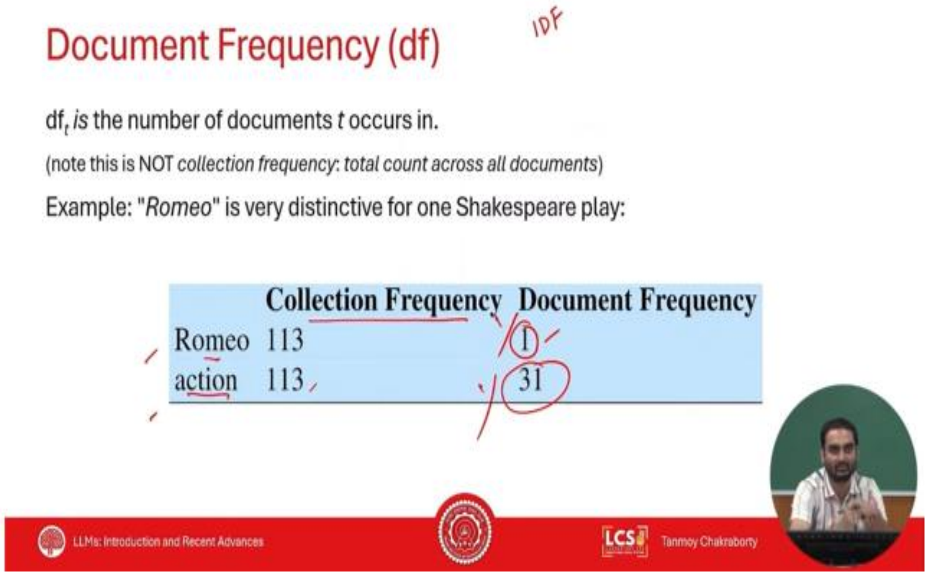
A document can be a novel. A document can also be a paragraph. A document can also be a sentence. It depends on how you define a document, right? Because if you have only one document, then how do you compute TFIDF? You consider every paragraph as a document and then you measure this. So count of this, a raw count of this. Now the raw count will be very, very high for again frequently appearing words, stop words, right.

To just dampen the effect of this, we squash this a bit. Instead of taking the raw count, I take the log of the raw count, right. Log of the raw count and if the count is 0, then log 0 will be undefined, right. Then therefore we just add 1. That's a smoothing technique.

$$tf_{t,d} = count(t,d)$$

$$tf_{t,d} = \log_{10}(count(t,d) + 1)$$

So log count plus 1 is the TF measure. What is the problem in TF?



Document Frequency (df)

IDF

df_t *is* the number of documents *t* occurs in.

(note this is NOT *collection frequency: total count across all documents*)

Example: "*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

The problem with TF is that again frequently appearing words, stop words will have very high TF count okay. To address this we will use another component of TF-IDF called inverse document frequency IDF okay IDF. So what is inverse document frequency? Let's first see what is document frequency. A document frequency is the number of documents where the particular word appears. It is not the number of counts of all the documents where this particular entry appears.

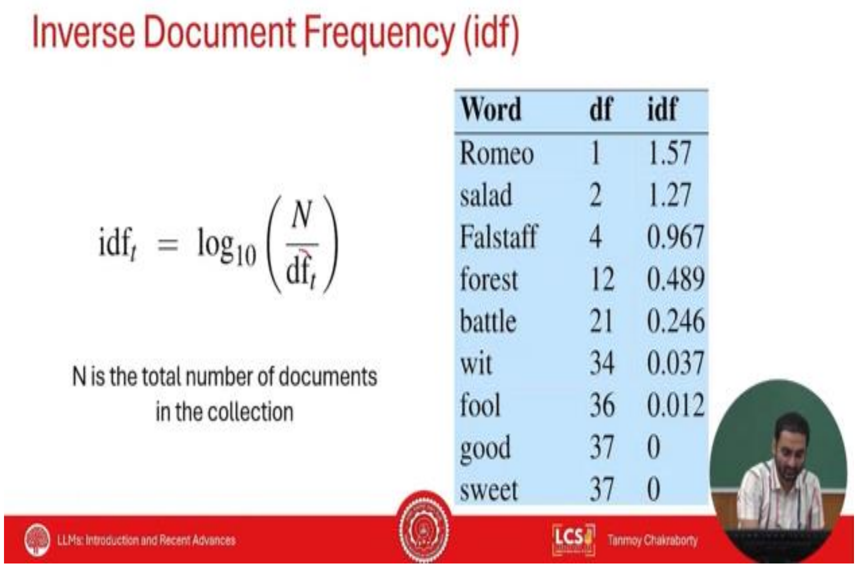
Only the number of documents where the particular entry appears. It's called document frequency. Let's say there are two words Romeo and action. The word Romeo appears only one document, not in the other documents of Shakespeare, for example. Okay. So the term frequency will be very high for that document, but the document frequency will be very low.

Why? Out of let's say thousand documents, only one document has this word Romeo. Right? So document frequency will be let's say one. By the way, what is this collection frequency? Collection frequency is the sum of the count of this word across all the documents. Term frequency is with respect to one document.

Collection frequency is the sum of counts in all the documents. And document frequency is the number of documents where this particular entry appears. Clear? Okay. So the word Romeo will have a document count of 1 whereas the word action which is a more generic word than Romeo, right? Let's say his collection frequency is still 113. But this collection frequency is distributed across documents.

It is not that only in one document the word action appears 113 times. In total it appears 113 times fragmented. And let's say there are 31 documents where the word action appears. So this is document frequency. Now I will take the inverse of this, inverse document frequency.

So 1 by 1, 1 by 31. Now what is the meaning of this inverse document frequency? When we take the inverse of this, if a word appears in all the documents like the word action or stop words, if it appears in all the documents, its importance will be reduced, which is obvious, we don't want to capture those words which appear in all the documents, rather than this we should consider those words which are representative.



Inverse Document Frequency (idf)

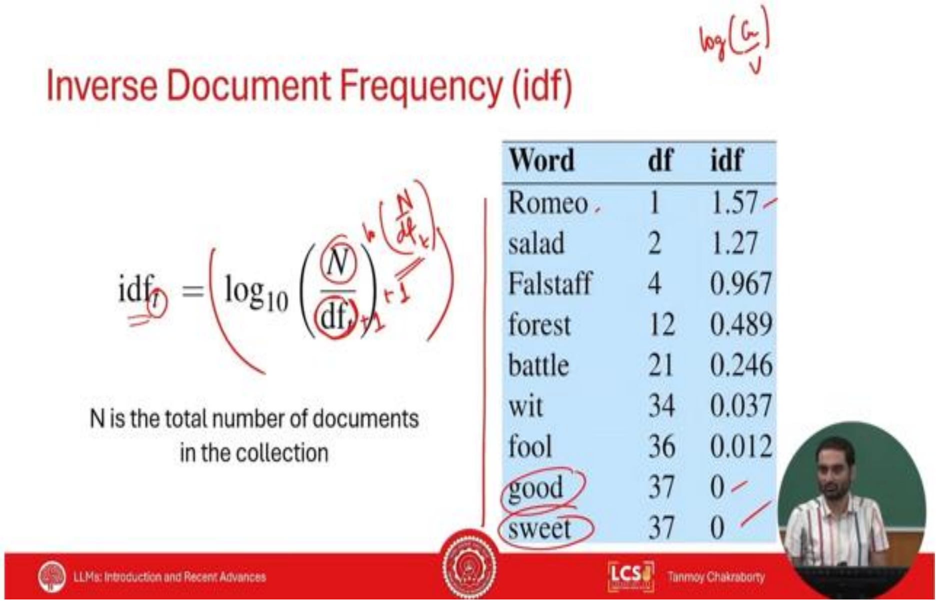
$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

LLMs: Introduction and Recent Advances

LCS Tanmoy Chakraborty



$\log\left(\frac{a}{v}\right)$

Inverse Document Frequency (idf)

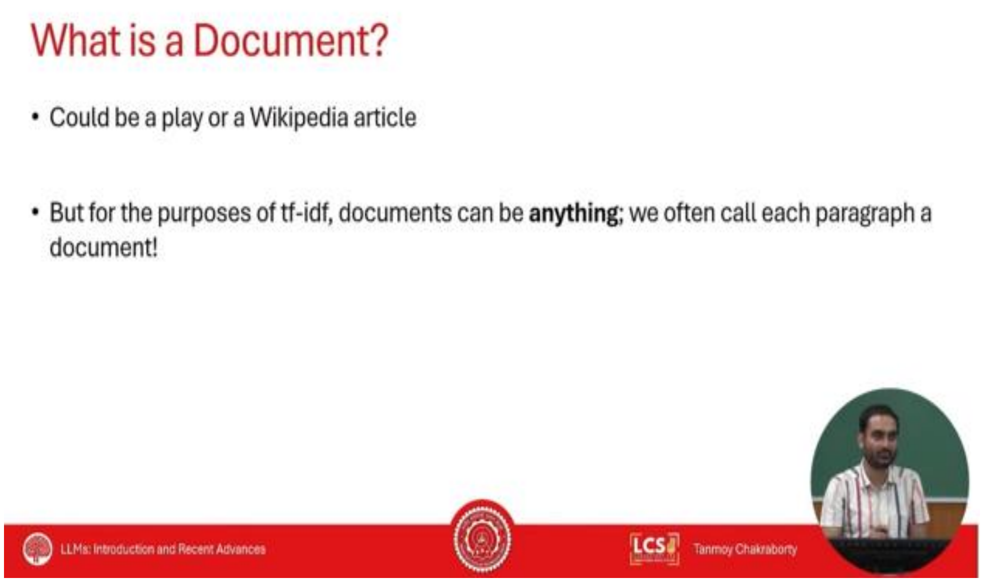
$$\text{idf}_t = \left(\log_{10}\left(\frac{N}{\text{df}_t + 1}\right) + 1\right) \quad \ln\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

LLMs: Introduction and Recent Advances

LCS Tanmoy Chakraborty



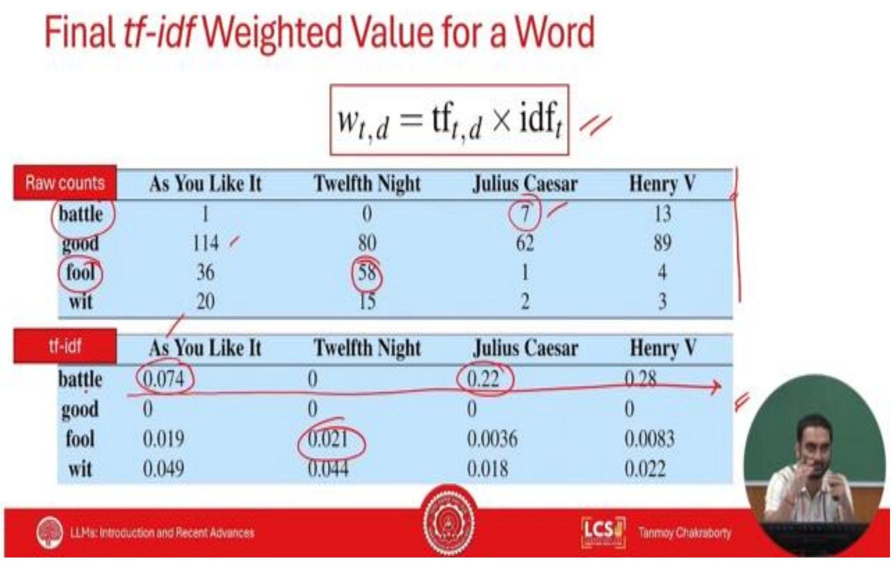
What is a Document?

- Could be a play or a Wikipedia article
- But for the purposes of tf-idf, documents can be **anything**; we often call each paragraph a document!

LLMs: Introduction and Recent Advances
LCS2
Tanmoy Chakraborty

So this is IDF, this is the formula of IDF, so df, 1 by df is inverse document frequency of the term t, remember here IDF is a function of only the term t, not t, d. Right and we take a log of this and just to normalize the value we normalize it by n, what is n? n is the number of documents, okay, so we just normalize it by n. Now this is the general generic idea formula but as you can see you know if df is 0 then what would happen right.

So we sometimes we also add 1 here right to the denominator. Sometimes what we do we also add 1 here. There are multiple variations of tf idf right in tf also term frequency instead of taking the log of count right what we do we normalize it by number of tokens in the document right. Let's say the number of tokens is v, for example, right. Sometimes we normally normalize it by max count, not the total count max count and there are different variations of this okay Now look at the IDF value of the different words, the word Romeo has the IDF value of 1.57. Whereas the word good or the word sweet, which are frequently occurring in all the documents, their IDF values are 0. So you are penalizing those words which appears in all the documents. So the combined version of this metric is TF-IDF where you multiply TF with IDF.



Final *tf-idf* Weighted Value for a Word

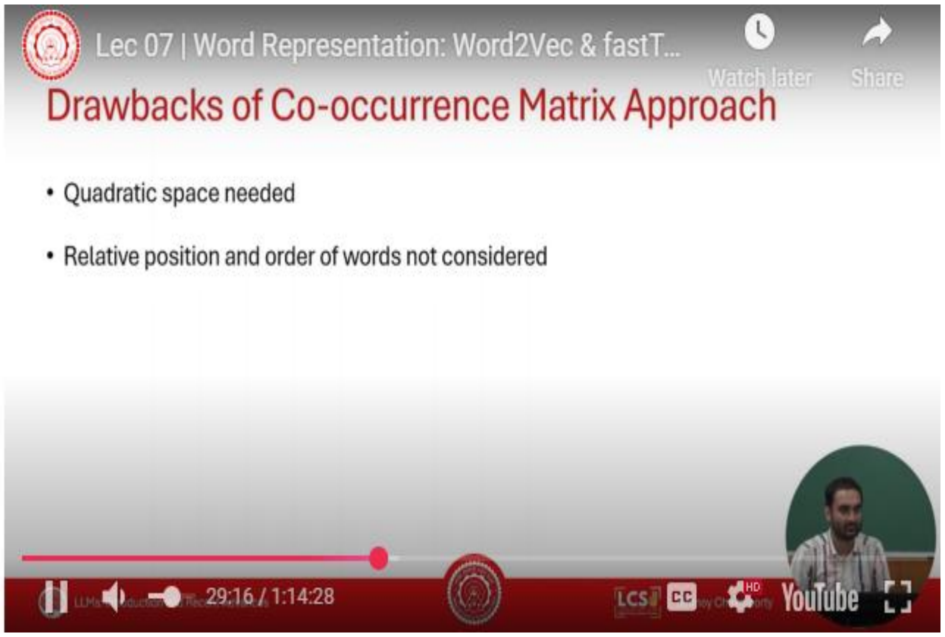
$$\boxed{w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t}$$

Raw counts	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

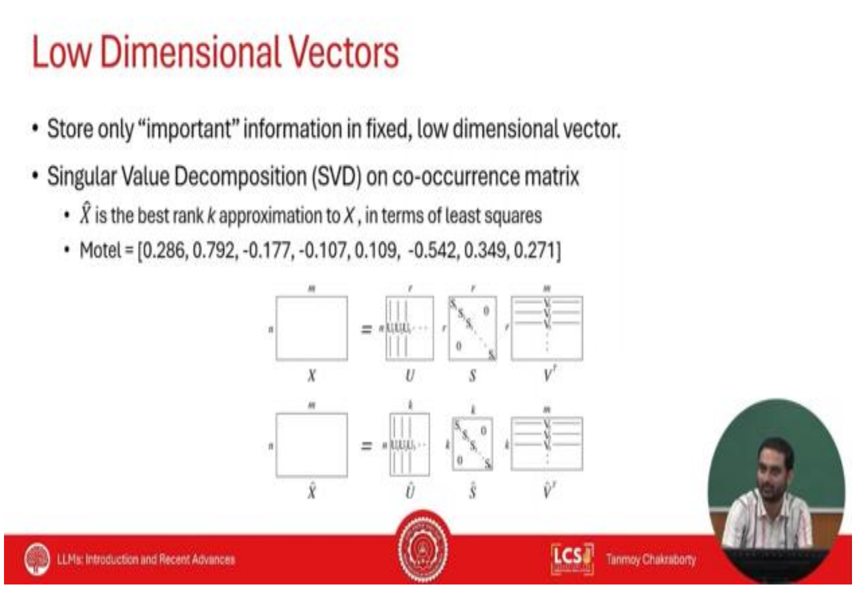
LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

So this is the formula of TF-IDF. This is the raw count table before TF-IDF. This is the TF-IDF table. In the raw term document matrix, you see that there are entries like let us say the word battle appears only in the Zulia Caesar novel. Its count is 7 but in TF-IDF its value is significantly high 0.22 whereas the word fool which is kind of a common word whose count is pretty high 58 but TF-IDF is pretty low. From row count, we moved to tfidf, right? So every entry here is the tfidf value for that word with respect to the document, right? And each row is a vector now, similar to earlier case. So it addressed the problem of frequently appearing words but there are still other problems. For example, if the document size increases, the vocabulary size increases, the matrix will increase, the vector size will also increase, there are so many zeros and so on, sparsity and so on and so forth.



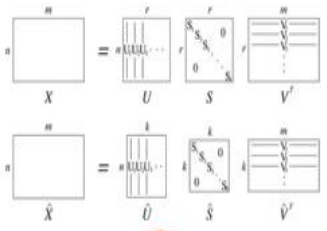
Drawbacks of Co-occurrence Matrix Approach

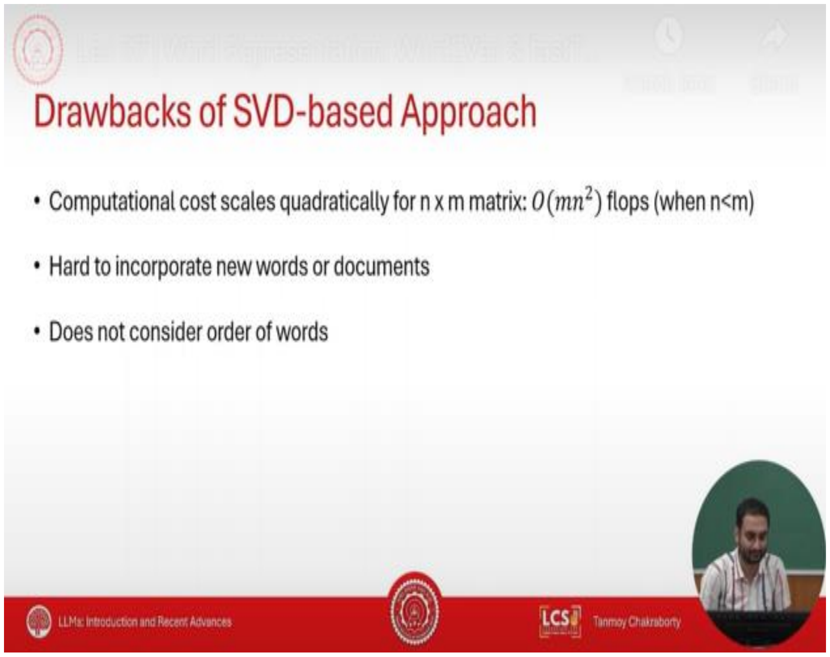
- Quadratic space needed
- Relative position and order of words not considered



Low Dimensional Vectors

- Store only "important" information in fixed, low dimensional vector.
- Singular Value Decomposition (SVD) on co-occurrence matrix
  - $\hat{X}$ is the best rank k approximation to X, in terms of least squares
  - Motel = [0.286, 0.792, -0.177, -0.107, 0.109, -0.542, 0.349, 0.271]





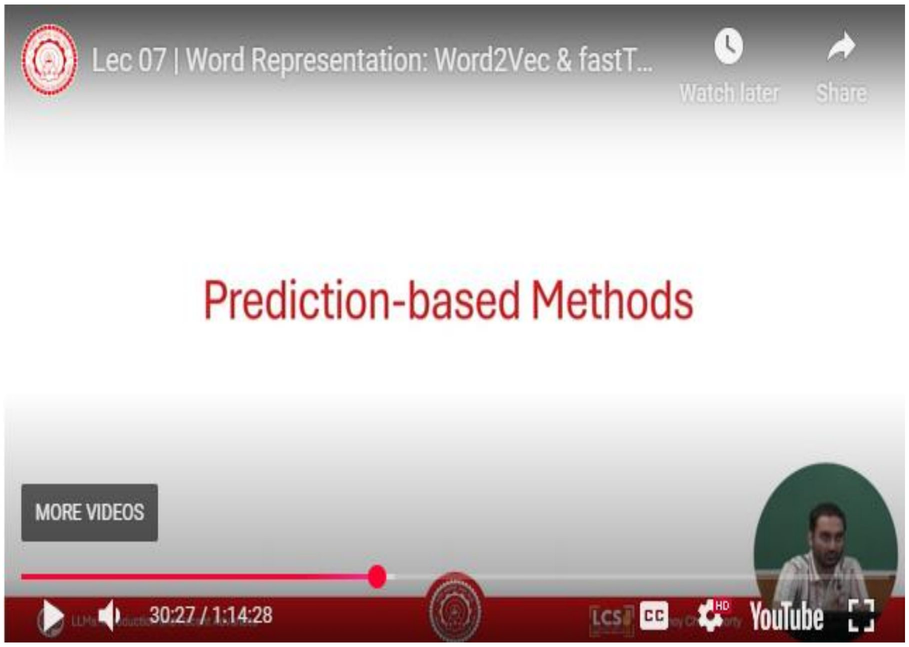
Drawbacks of SVD-based Approach

- Computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
- Hard to incorporate new words or documents
- Does not consider order of words

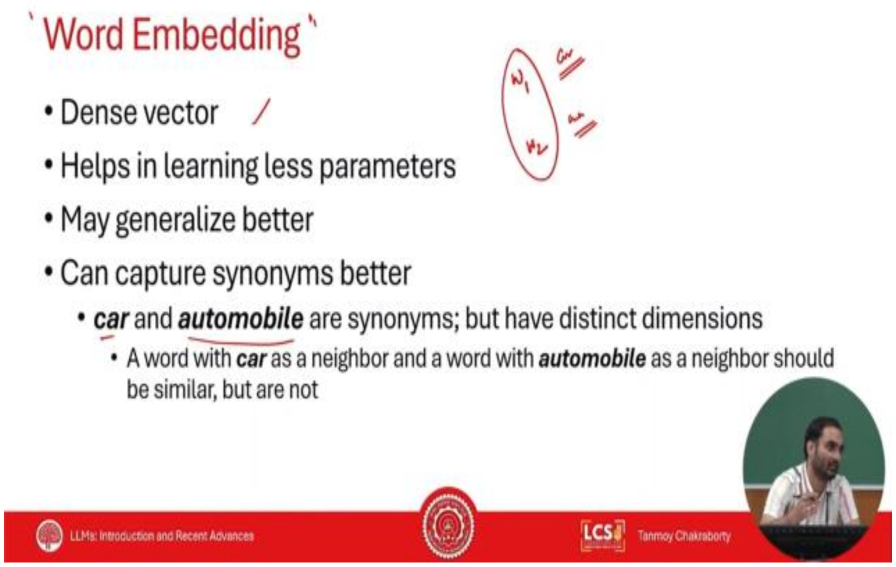
So now these are the problems of count based approaches. So one can say that you know so this sparsity problem or you know bigger size matrix problems right we can easily address by doing some sort of decomposition right. We can use SVD, for example, right on top of this count matrix, or TFID matrix I do SVD and then I decompose it.

And we will take the decomposed version of it. But SVD is again time consuming. It is quadratic in the number of rows and columns and the other problem is that as new words comes in, right you have to reconstruct the table, you have to again apply SVD on top of this and so on and so forth. The last and most important drawback of this kind of count based matrix is that it does not consider the ordering, ordering of words. I look at a particular context and within the context, this context window of size 10 for example, all the words are same.

It's a bag of word representation. All the words are the same. So it doesn't take into account the fact that if some word appears before the other word or some word appears after the other word, their meaning may change. So SVD may not be a good approach because of the quadratic complexity. and incorporating difficulties in incorporating new words and so on and so forth.


Lec 07 | Word Representation: Word2Vec & fastT...

Prediction-based Methods



Word Embedding

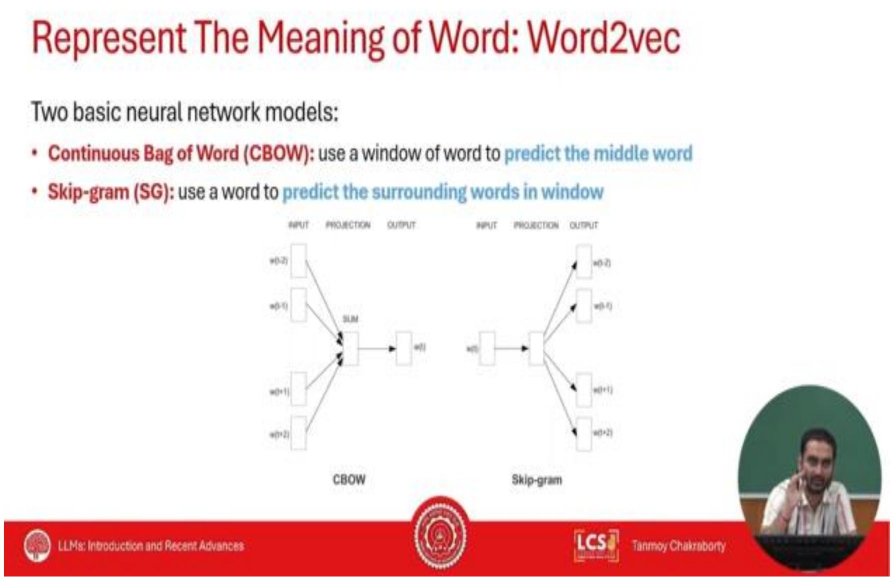
- Dense vector
- Helps in learning less parameters
- May generalize better
- Can capture synonyms better
  - ***car*** and ***automobile*** are synonyms; but have distinct dimensions
    - A word with ***car*** as a neighbor and a word with ***automobile*** as a neighbor should be similar, but are not

So to address this problem, now we move to the recent approach called Word2Vec, a prediction based approach.

We have seen count based approach, now we look at prediction based approach. So let us go back to 2013. when Google Thomas Mikolov and Jeff Dean they came up with the idea of what vector and this term what vector was coined that at that time what vector is nothing but the vector that we have been discussing but it is dense. Now it is not of size 1 million it's the size could be 100 or 200 or 500 right. What vector is a dense vector? Now why a dense vector is important instead of a sparse vector? Because if the feature size is, if the number of features is high right it may be prone to over fitting we need lots of parameters to train right we all know scars of dimensionality right. So if we squeeze it maybe we do not need to of course we do not need to train many parameters.

The second problem is that it will be more generalizable right. The other good part is that If you take this kind of dense vectors, oftentimes we see that the words car and automobile, they are synonymous. But if we look at the traditional count-based approaches or 1-0 encoding kind of approaches, this kind of synonymous words are not captured properly. In fact, oftentimes we say that if a word W co-appears with the word car, as well as with the word automobile right. Let's say there's a word w1 which co-appears with the word car there's a w2 which co-appears with the word automobile w1 and w2 can also be synonymous right.

So this kind of inference may not be very straightforward from a raw count-based approach.



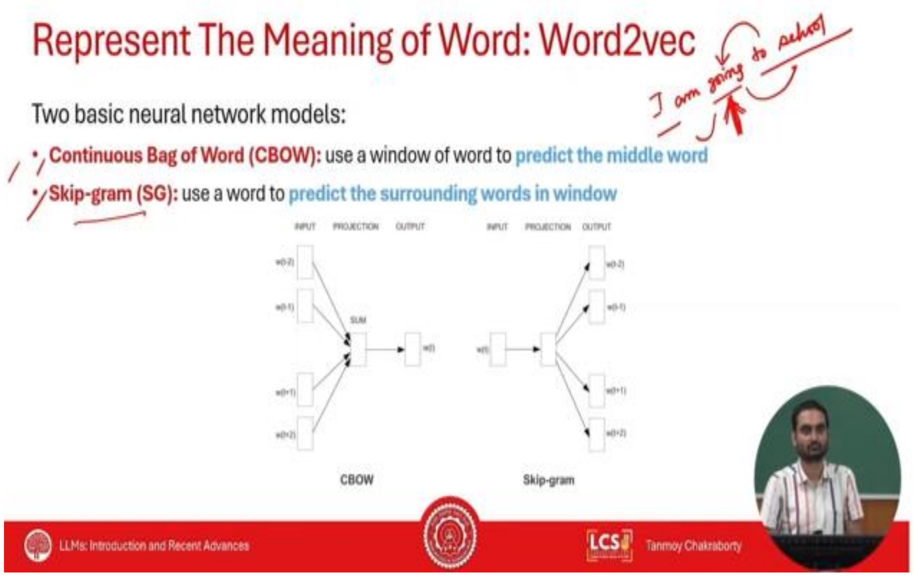
Represent The Meaning of Word: Word2vec

Two basic neural network models:

- **Continuous Bag of Word (CBOW):** use a window of word to predict the middle word
- **Skip-gram (SG):** use a word to predict the surrounding words in window

LLMs: Introduction and Recent Advances

Tanmoy Chakraborty



Represent The Meaning of Word: Word2vec

I am going to school

Two basic neural network models:

- **Continuous Bag of Word (CBOW):** use a window of word to predict the middle word
- **Skip-gram (SG):** use a word to predict the surrounding words in window

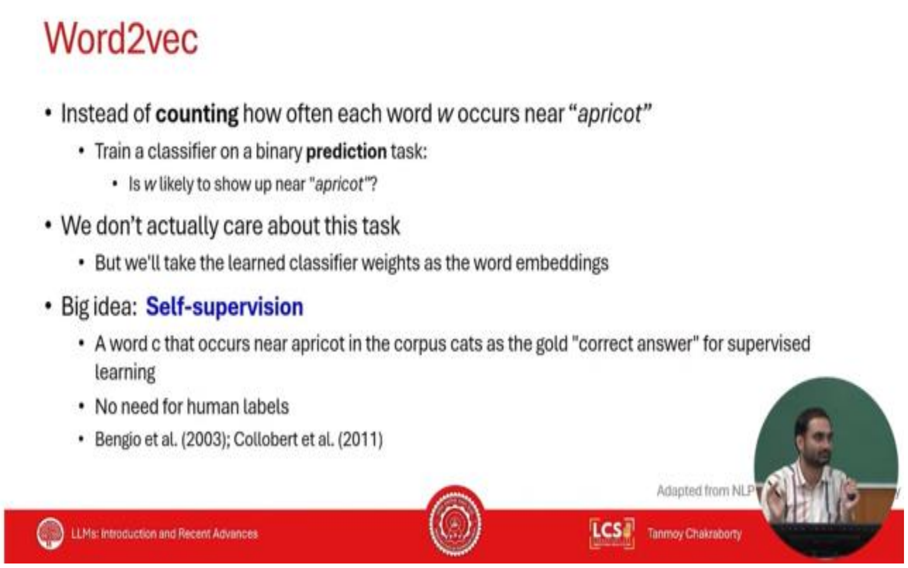
So what do we was proposed in 2013 and in the what to wake package, there are two types of approaches. We are almost the same, but prediction wise are slightly different. Remember, these are prediction-based approaches. We are moving from count count-based approach to the prediction-based approach.

So the first one is called CBOW, continuous bag of word model. and the second one is called the slipgram model. So the idea is almost the same. Let's say there is a sequence of tokens. Let's say I am going to school. I want to measure the vector corresponding to the word going, I want to measure the word vector corresponding to the word going, right in CBOW.

What I do? I will take the context tokens right to predict the word embedding of the given word, in skip-gram I will use the center token to predict the word embedding of the context words It's just opposite. I have a sequence of words, sequence of tokens, and let's say there is a central word. For example, in this case, the central word is going, for which I want to predict the embedding. In CBOM, I will look at the context words to predict the representation of the central word.

In SkipGram, I look at the central word to predict the representations of the context words. In a neural network terminology this is very straightforward. Let's say you have a multi-layer perceptron kind of network, a two-layer network. In CBOW what I do, I feed the representation of the context word and I will try to generate the representation of the central word. In skipgram, I feed the representation of the central word and try to generate the representation context word, okay. So let's focus on skip-gram because this is more popular right, Skip-gram with negative sampling, okay.

We will discuss what is negative sampling and so on okay.



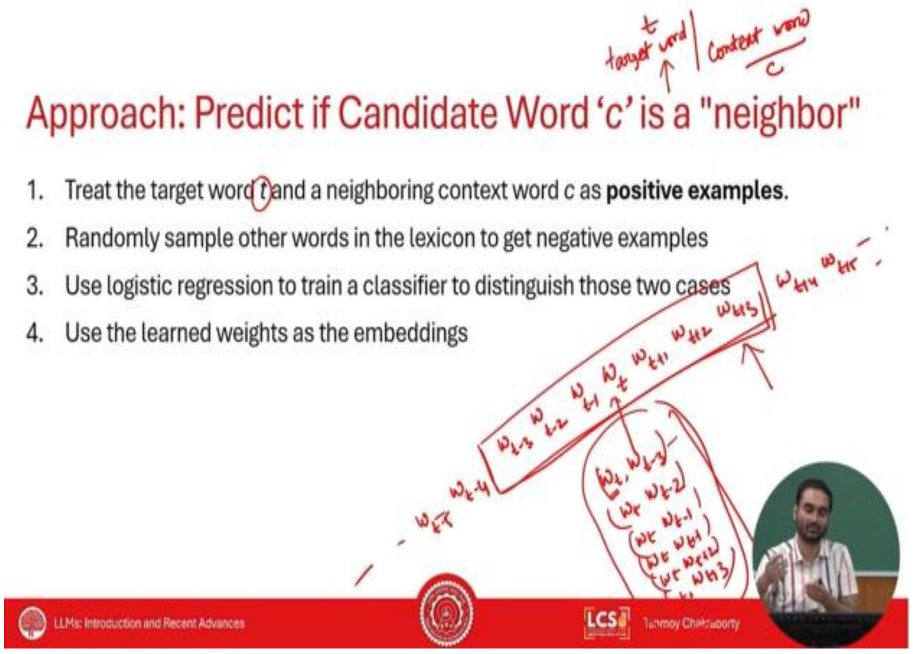
Word2vec

- Instead of **counting** how often each word *w* occurs near "*apricot*"
  - Train a classifier on a binary **prediction** task:
    - Is *w* likely to show up near "*apricot*"?
- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings
- Big idea: **Self-supervision**
  - A word c that occurs near apricot in the corpus cats as the gold "correct answer" for supervised learning
  - No need for human labels
  - Bengio et al. (2003); Collobert et al. (2011)

So what to make is an unsupervised approach I would say this is a self supervised approach you do not need any labeling any human label data Number one. Number two, this is just a prediction task. A simple logistic regression task. It's a simple logistic regression task.

All of you know logistic regression? It's a simple logistic regression classifier. It's a classifier which will classify that this word is not a context word, this word is a context word.

That's all. Very simple idea.



Approach: Predict if Candidate Word 'c' is a "neighbor"

1. Treat the target word t and a neighboring context word c as **positive examples**.
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

target word t
context word c
$w_{t-3}\ w_{t-2}\ w_{t-1}\ w_t\ w_{t+1}\ w_{t+2}\ w_{t+3}$
$w_{t+4}\ w_{t+5}$
$w_{t-4}\ w_{t-5}$
(w_t, w_{t-3})
$(w_t\ w_{t-2})$
$(w_t\ w_{t-1})$
$(w_t\ w_{t+1})$
$(w_t\ w_{t+2})$
$(w_t\ w_{t+3})$
LLMs: Introduction and Recent Advances
LCS
Tanmoy Chakraborty

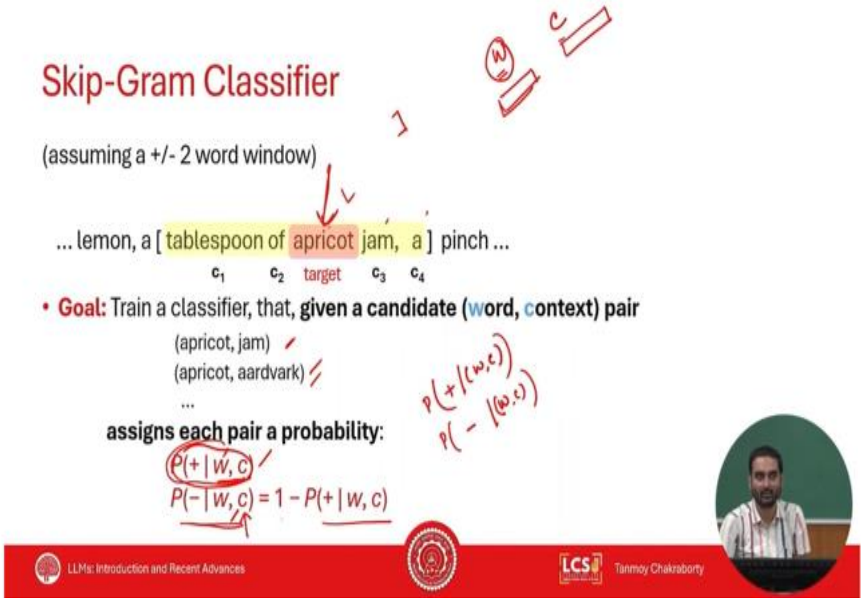
Let's look at the approach. So we have a target word T. Now remember, we will use two terminologies. One is called the target word, and the other is called the context word. Right, T and C. Okay, given a particular target word, I will look at what are the context words.

And how do we identify a context word? I have this window, right? I have this window of size 10. We have the documents, multiple sentences. I move the window one step at a time, right? At every step, I look at 10 words at a time, right? This is the middle word is a central word and the remaining nine words are the context words, right? So what I do for every central word, I make a pair. Let us say Wt is the target word and Wt minus 1, Wt minus 2, Wt minus 3, Wt plus 1, Wt plus 2, Wt plus 3, this is the context size. Moving window size right. And there are of course other words, here it's a wt plus t minus 4 wt minus 5 right, wt plus 4 wt plus 5 and so on and so forth and, this is my sliding window okay, this is my target word.

So what are the positive context words? The positive context words are Wt. So for Wt target word, the positive context words are Wt minus 3, Wt minus 2, Wt minus 1, Wt plus 1, Wt plus 2, Wt plus 3. These are the positive context words because these words co-appear with the word Wt within the context. So these are the pairs, these are the positive pairs given a particular window size, right. I identify I will first identify the context words and make these pairs. What are the non-context words with respect to this context size at this position? What are the non-context words? Non-context words are those words which are outside of this particular window, right? So there are, if the size of the document is n, there are n minus how many? 3 plus 3, 6 plus 7, n minus 7 tokens which are non-context words, right? Okay, so I will sample non-context words from the document.

So this is my context word pairs. I will sample equal number of non-context pairs from the document. In fact, in word2vec, what we do, in skipgram specifically, we take k times more negative samples. These are negative samples. These are positive samples and those are negative samples. Non-context, what are negative samples? So if I take 10 positive samples, I will take 40 negative samples 4 times or 5 times. Why? Why? It will give you more evidence, right? Not only that, sometimes what happens is that when you sample negative examples, it may happen that some of the positive examples can also come, because you are sampling from some distribution. It may happen that some of the positive samples can also come. In fact that particular word can also come as a context, negative context. Of course you can have tricks to remove those duplicates but it is always safe to consider more negative examples. We will discuss how to sample negative examples later. This is the sequence of tokens, I fix the context window, I sample positive instances, negative instances, I curate positive instances, sample negative instances, then I run this classifier.

And the classifier will classify positive instances from the negative instances. Now how I will tell you? Right so again you move this context window one word forward, right. Again, you curate all these things, again you run the classifier and you keep on doing this thing until unless the entire document gets scanned. This is the overall philosophy, okay, let's look at examples.



Skip-Gram Classifier

(assuming a +/- 2 word window)

... lemon, a [tablespoon of apricot jam, a] pinch ...

c_1 c_2 target c_3 c_4

- **Goal:** Train a classifier, that, **given a candidate (word, context) pair**

  (apricot, jam)

  (apricot, aardvark)

  ...

  assigns each pair a probability:

  $P(+|w,c)$

  $P(-|w,c) = 1 - P(+|w,c)$

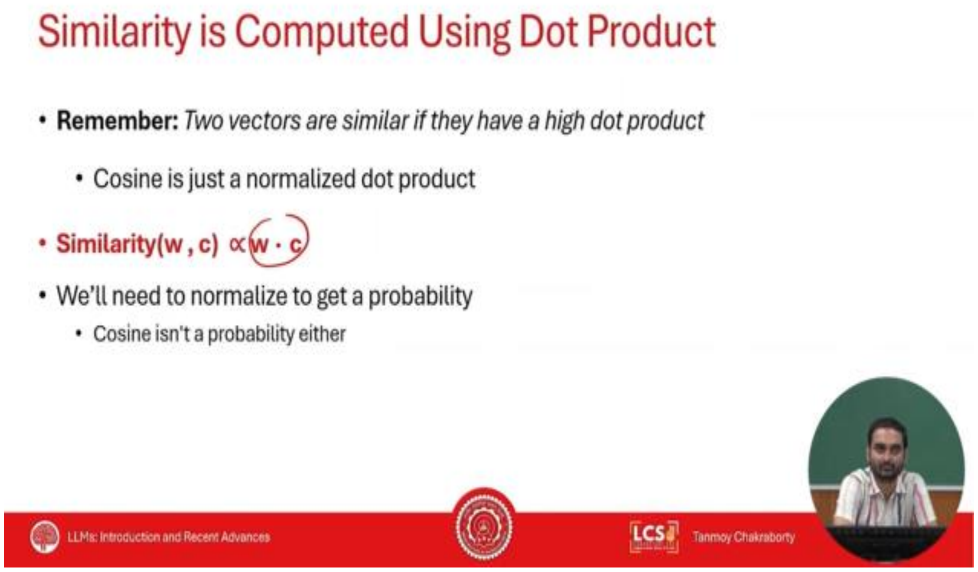
Let's say this is a real example. Lemon, a, tablespoon, of, apricot, jam, a, pinch. This is a sequence of words, right? And let's say the context window is this one from here to here, okay? And the target is apricot. Tablespoon and of are context words. Jam and a are also context words and other are non-context words, right? These are context pairs.

So for every word w, now this is w okay. For every word w we have a context vector c. Now remember this word w is represented by a vector, the word embedding vector which we are going to learn right. Through this classification, we are going to learn the embeddings of w and c. What is c, c is the context words and w is the target word t. We need a representation for C, we need a representation for W, right and over the iterations we will learn the representations of W and C. We start with a random initialization of W and C, it can be let us say 100 encoding, for example, simple or it can be random initialization and then you keep on iterating, right.

Okay, what is the task of the classifier? The task of the classifier is that given a pair of instances W and C, what is W? What is W? W is the target word, central word. What is C? C is the context word. Given this pair W and C, we measure the probability that this is a positive pair. So given that we have W and C, what is the probability that this is a positive pair? positive instance.

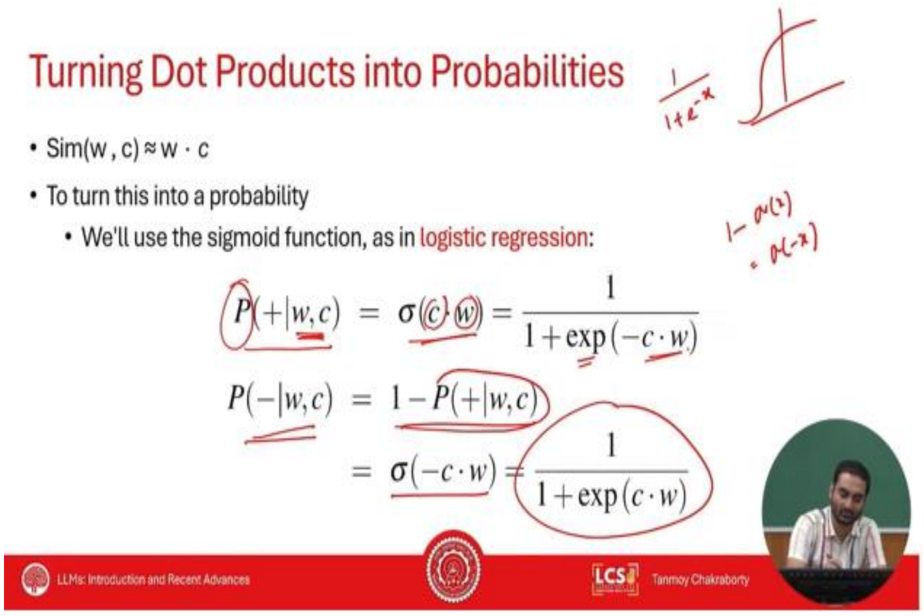
This is a positive instance. If it is a context, it is a positive instance, right. If C is a negative instance, then I will measure, given this tuple W and C, what is the probability that this is a negative instance. So we will measure these two probabilities, okay. Now negative probability is same as 1 minus positive probability. Right, so essentially we will measure this positive probability, okay.

What is the probability that a particular pair is a positive pair, okay.



Similarity is Computed Using Dot Product

- **Remember:** *Two vectors are similar if they have a high dot product*
  - Cosine is just a normalized dot product
- **Similarity(w , c) ∝ w · c**
- We'll need to normalize to get a probability
  - Cosine isn't a probability either



Turning Dot Products into Probabilities

- Sim(w , c) ≈ w · c
- To turn this into a probability
  - We'll use the sigmoid function, as in logistic regression:

$$P(+|w,c) = \sigma(c\cdot w) = \frac{1}{1+\exp(-c\cdot w)}$$

$$P(-|w,c) = 1 - P(+|w,c)$$

$$= \sigma(-c\cdot w) = \frac{1}{1+\exp(c\cdot w)}$$

$\frac{1}{1+e^{-x}}$

$1-\sigma(x)$
$=\sigma(-x)$

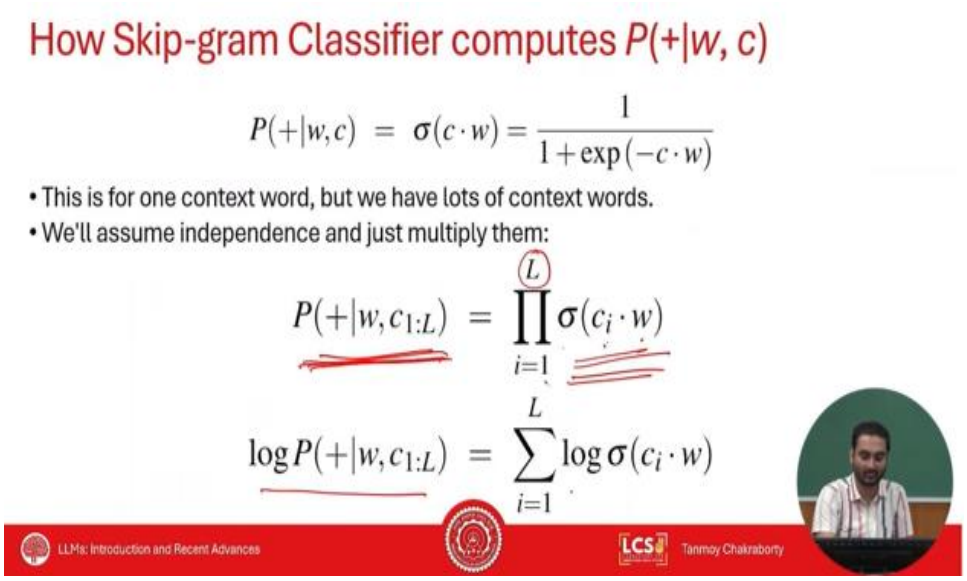
So how do you measure this probability? We measure this probability using the dot product right. The dot product is a measure of similarity.

We all know what the dot product right. Now let's look at the equation here. It's a very simple equation. So I mentioned that this is just a classifier. This is a logistic regression, right? To measure this probability that W and C are positive instances, right? W and C's pair is a positive instance. What I do, I measure this probability using a sigmoid function, right? What is the sigmoid function? right and the beauty of the sigmoid function is that it basically squashes everything between 0 to 1 right. Machine learning researchers always prefer this sigmoid function because not only it produces a probability but also the derivative has some beautiful formula right. If you take the derivative of sigmoid this is essentially sigmoid into 1 minus sigmoid and there are also other very interesting properties.

Of course if you ask a statistician they will say that you know we like I will not consider sigmoid as a probability because probability has certain assumes and certain other properties but for a machine learning researcher I think this is a good enough function right. So sigmoid of C and W, remember C is a vector embedding vector, right, W is also a vector, right. So this is computed by 1 by 1 plus e to the power minus c times x. This is a dot product, okay, right, and how do you compute the negative one 1 minus positive remember what is 1 minus sigma dot of x, 1 minus sigma dot of x is sigma dot of minus x, So sigmoid function has very beautiful properties.

So 1 minus the sigmoid is basically sigmoid of minus of that. And if you do this math you see that this is essentially this one. 1 by 1 plus exponential of C times W.

So now this is for one pair. This is for a single pair of word and context. Target and context. There are multiple such pairs. If the context size is 10, there are 9 such pairs of positive instances and there are 40 such pairs of negative instances if it is 4 times. So this is for one pair. So how do we then consider all the other pairs? For other pairs also I measure this and we assume that all these pairs are independent.



How Skip-gram Classifier computes $P(+|w, c)$

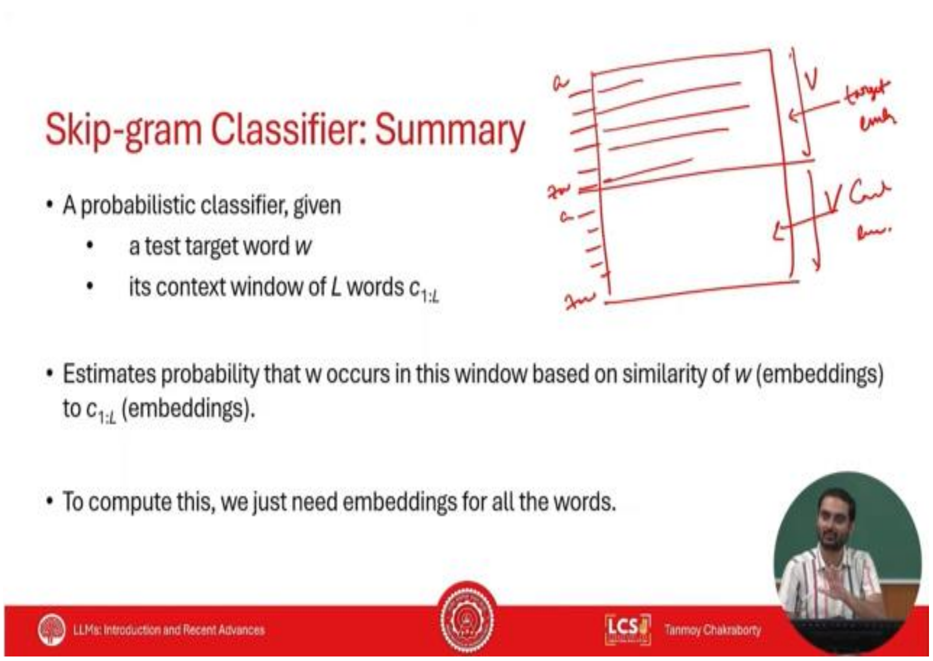
$$P(+|w,c) = \sigma(c \cdot w) = \frac{1}{1+\exp(-c \cdot w)}$$

- This is for one context word, but we have lots of context words.
- We'll assume independence and just multiply them:

$$\underline{P(+|w,c_{1:L})} = \prod_{i=1}^{\boxed{L}} \underline{\sigma(c_i \cdot w)}$$

$$\underline{\log P(+|w,c_{1:L})} = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



Skip-gram Classifier: Summary

w
V
target emb
2w
c
V Cont emb.
L
2w

- A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$
- Estimates probability that w occurs in this window based on similarity of w (embeddings) to $c_{1:L}$ (embeddings).
- To compute this, we just need embeddings for all the words.

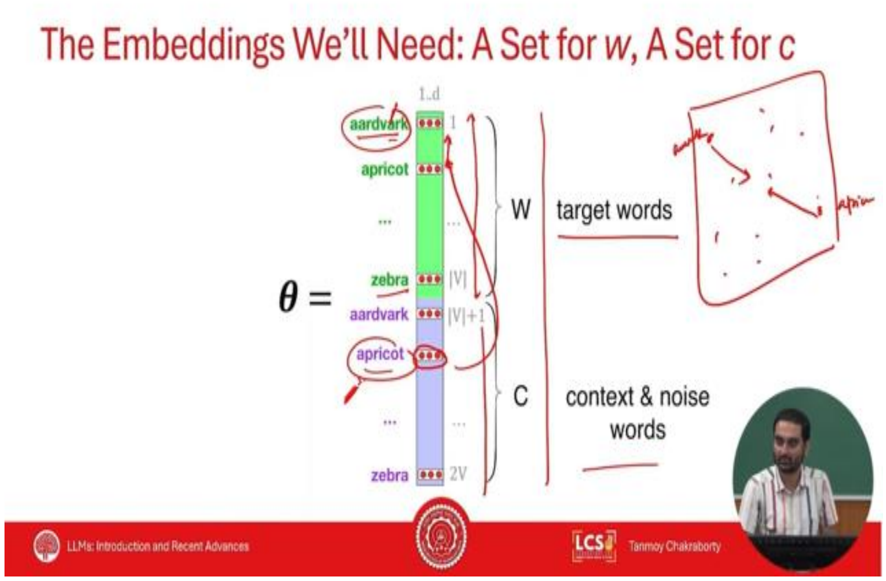
LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

If they are independent then this is just a product of all the probabilities. This is just a product of all the probabilities, isn't it? There are l such instances right, a sigma dot of ci times w where i tends to i ranges from 1 to l, l is the context size, the window size, is this clear.

Okay, so this is essentially the you know, all the probability is considered together so we do not consider the sum, we do not consider the product often times, we take the log of it, right, we take the log of this probability and which will essentially be the sum of all the logs right. So what we have done so far tell me. We fix the context window size, for every context window we sample positive instances and negative instances, we curate positive instances and sample negative instances and then we measure the probability and then the positive probability should be higher than the negative probability that the classifier needs to make sure, the classifier basically logistic regression works based on the log likelihood estimate right. So we will have this log likelihood estimate and that's my objective function then we take the derivative because this is the objective function, log, we maximize the log likelihood, right, then we take the derivative, What are the parameters here? The parameters are the vectors are this is C and W vectors.

Now here is the question. So tell me for every word, how many vectors we are computing? Two. Two vectors, right? C and W or C and T, right? Where, you know, a time when the word acts as a central word and the time when the word acts as a context word. right. So if you think of this matrix, if you think of the embedding as a bulky matrix and assume that there are V number of vocabulary tokens or vocabulary entries present in the document right, V is the size of the, V is the number of tokens right. So we have V entries here and we have V entries here right. Let us say from A to zoo here from A to zoo right and this entries the upper part of the matrix represents the target embedding and the lower part represents the context embedding.

So we are essentially learning two V number of vectors. Right and what is the size of the vector is the hyper parameter. We have to decide beforehand, let's say the size of the vector is 10, right. So what's the size of this matrix 2v times 10, this main number of parameters we are updating. Okay so now look at this one.


The Embeddings We'll Need: A Set for *w*, A Set for *c*

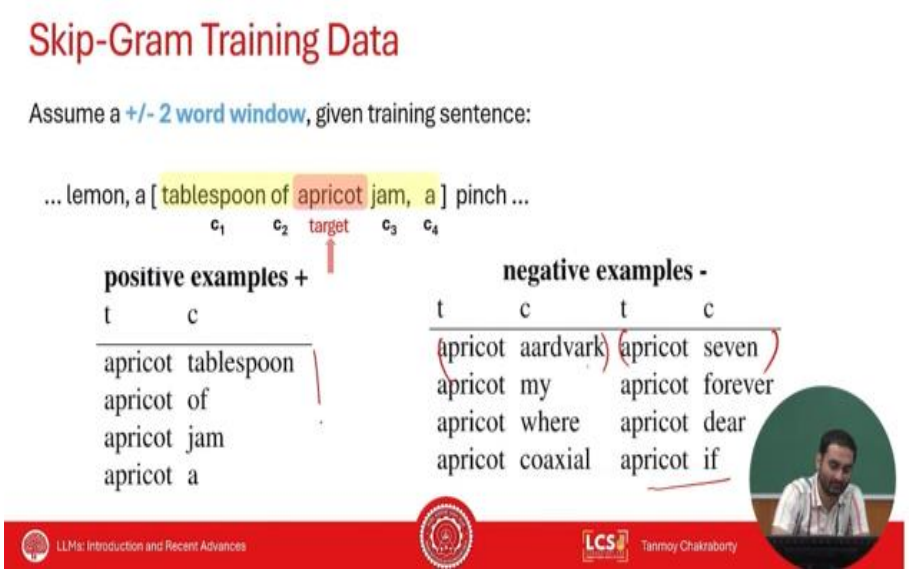
So this is this bulky matrix which we are estimating. So the word starts from adverb to let's say zebra and this mod V size part of the matrix corresponds to the target words and the similar mod V type size sub matrix corresponds to the context words.

Okay. And when I say, when I say that the word artwork, for example, the word aardvark and the word apricot, just again, for the sake of, you know, give an example, let's say when I say that the target was what is aardvark and the context, what is apricot, let's say, and the apricot is a positive context for the word aardvark. Right. Let's say this is my initial matrix when you initialize it, right. And when I say that this is a positive context of this, what does it mean? It means that over the iterations, I will try to move the representation of the context of apricot closer towards the target of aardvark.

Let me repeat what I just mentioned. Aardvark is the target word and apricot is the context word, right? When you initialize this matrix at the beginning, all this initialization happened random, right? And over the iterations, what is the target? The target is that positive context should come closer to the target and negative context should basically be far from the target. So over the iterations, I will update this matrix such that the positive context will come closer and negative context will basically go far. So apricot, you will see that over the iterations, if you look at the embedding space, let's say these dots indicate different embeddings in the vector space.

Let's say this is aardvark and this is apricot. this is the target of aardvark, this is the context of apricot, right. Over the iterations, we will say that this will move closer and closer, these two things move closer and closer and the negative instances will move further and further, okay.

Word2vec: Learning the Embeddings



Skip-Gram Training Data

Assume a +/- 2 word window, given training sentence:

... lemon, a [tablespoon of apricot jam, a] pinch ...

c_1 c_2 target c_3 c_4

positive examples +

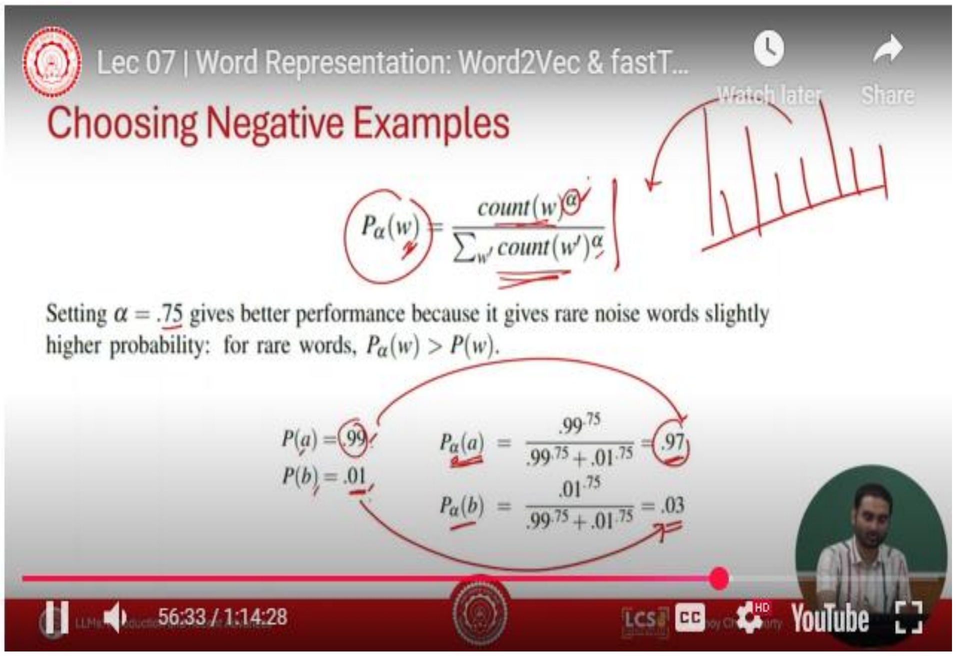
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

Now, let us look at the training process of Word2vec, okay. At every context window, I have a target and I have a context. I have multiple contexts, right? So in this case, apricot and tablespoon, apricot and off, apricot jam, apricot A, these are positive instances, right? And we sample K negative instances from the remaining documents, right? For example, apricot aardvark, apricot my, apricot where, apricot if, these are all negative instances, okay. So and then I train the logistic regression so that this and this the positive pair should be classified from the negative pair okay.



Choosing Negative Examples

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Setting $\alpha = .75$ gives better performance because it gives rare noise words slightly higher probability: for rare words, $P_\alpha(w) > P(w)$.

$$P(a) = .99 \qquad P_\alpha(a) = \frac{.99^{.75}}{.99^{.75} + .01^{.75}} = .97$$

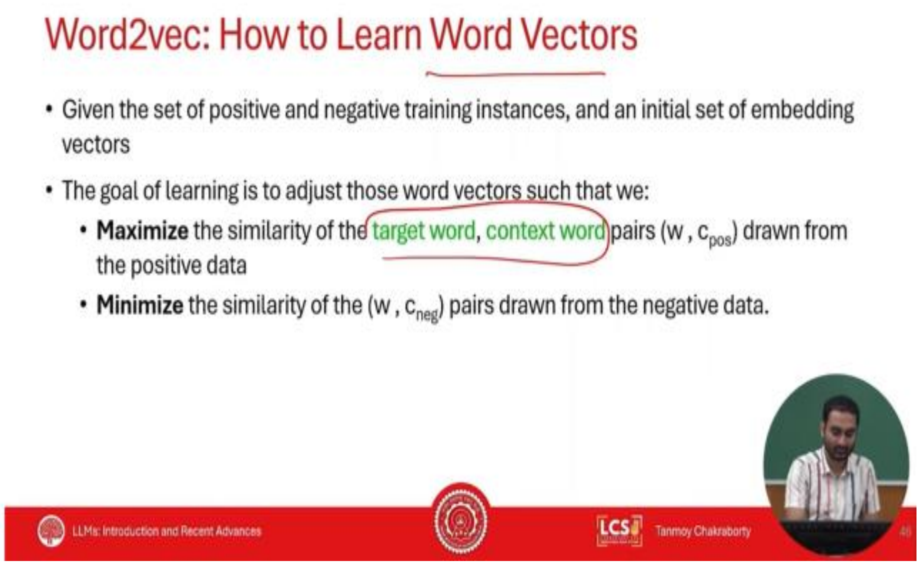
$$P(b) = .01 \qquad P_\alpha(b) = \frac{.01^{.75}}{.99^{.75} + .01^{.75}} = .03$$

How do we sample negative examples? We sample negative examples based on the unigram distribution right. So for every word I have a probability from the document I can easily measure the unigram distribution of every word, unigram probability of every word, right. So there is this unigram probability and then I sample every time I toss a coin and sample a negative example from the unigram probability, okay. But with the small tricks here, because what will happen if I only consider unigram probability, what will happen? Only the words which are frequently occurring, because their probability is high, those words will be sampled again and again.

But those words are not at all meaningful, non-contextual words. If I keep on sampling what if, if I keep on sampling the word, let's say, the or a, these are not meaningful words. So what I do instead of a raw unigram probability I will use some sort of modified unigram probability. So count of this W by sum of count this is the unigram probability I will essentially use another parameter alpha. So count to the power alpha by sum of count and every count has this alpha okay as a power.

Why this is needed? This alpha, addition of this alpha will give you a slightly higher edge to the rarely occurring words or low frequent words. Right let's if you are not convinced let me give an example. Let's say there are two words a and b and these are the raw unigram probability 0.99 and 0.01 Now if you use alpha, if you add alpha as a power and let's say the value of alpha is 0.75. So the modified unigram probability of A is 0.97 and the modified unigram probability of B is 0.03. So 0.03 is higher than 0.01, although a highly frequent word has still higher frequency, higher probability, but at least after this modification, rarely occurring words, they will have slightly higher probability and high frequent words will have slightly lower probability, right? And now you can control the value of alpha.

So this is the probability based on which I will sample negative instances.



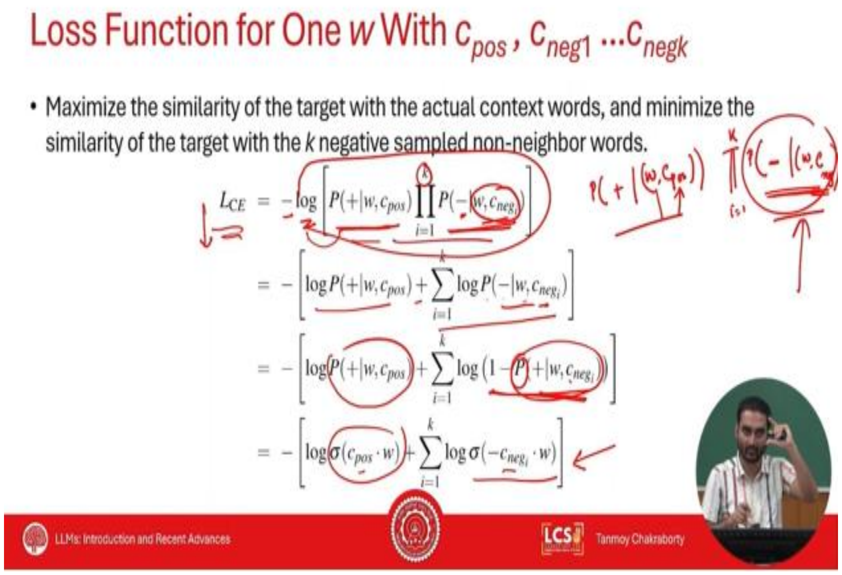
Word2vec: How to Learn Word Vectors

- Given the set of positive and negative training instances, and an initial set of embedding vectors
- The goal of learning is to adjust those word vectors such that we:
  - **Maximize** the similarity of the target word, context word pairs (w , c_{pos}) drawn from the positive data
  - **Minimize** the similarity of the (w , c_{neg}) pairs drawn from the negative data.

LLMs: Introduction and Recent Advances

LCS

Tanmoy Chakraborty



Loss Function for One w With $c_{pos}, c_{neg1} \ldots c_{negk}$

- Maximize the similarity of the target with the actual context words, and minimize the similarity of the target with the k negative sampled non-neighbor words.

$$L_{CE} = -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right]$$

$$= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right]$$

$$= -\left[\log P(+|w,c_{pos}) + \sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right]$$

$$= -\left[\log \sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log \sigma(-c_{neg_i}\cdot w)\right]$$

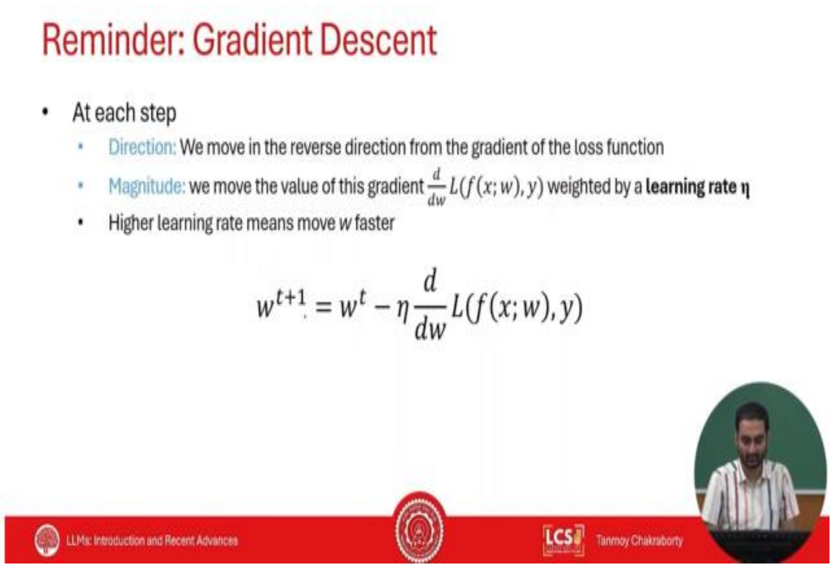
Okay, so how to learn what vectors we will maximize the target word context word pairs probability. We will minimize the target word negative context word pair probability. This is the target. So what is the final objective function for the logistic regression? Logistic regression maximizes the likelihood or minimizes the negative block likelihood.

So probability of the positive instance. And there are key negative instances I have sampled. Negative probability of, sorry, positive instance and the negative instance and we will see that whether this pair, this negative pair is what is the probability that this pair is a negative pair. So P of W C pos what is C pos? C pos is the embedding of the positive context right and W is the target word, right? So, what is the probability that this is positive, right? Similarly, I have a W and C neg okay, this one is a negative context, right What is the probability that this is a negative context, right, okay? And there are K number of negative samples that I have already sampled so I take a product of all this K and I maximize both these and this I mean you can you can say that why I am maximizing this I should minimize this, the second part, What is the second part? Given a target word and a negative context word, what is the probability that this is a negative instance? Should I maximize or should I minimize? Maximize.

Maximize. I want to maximize this also. I also want to maximize this. So I take the product, log of this. So this is the log likelihood minus negative log likelihood. I want to minimize this. Okay now if you do some small arithmetic so you move log inside so log this plus sum of log this right and we have already seen that okay so what is this by the way P of minus minus given W comma C neg is same as 1 minus P of plus given this right. This is same as this and how do we measure this probability using sigmoid, sigmoid and dot product right.

So this is this probability replaced by sigmoid This will be replaced by sigmoid. 1 minus sigmoid is 1 minus sigmoid x is sigmoid of minus x. So this is same as sigmoid of minus x. So this is objective function that I want to minimize now.

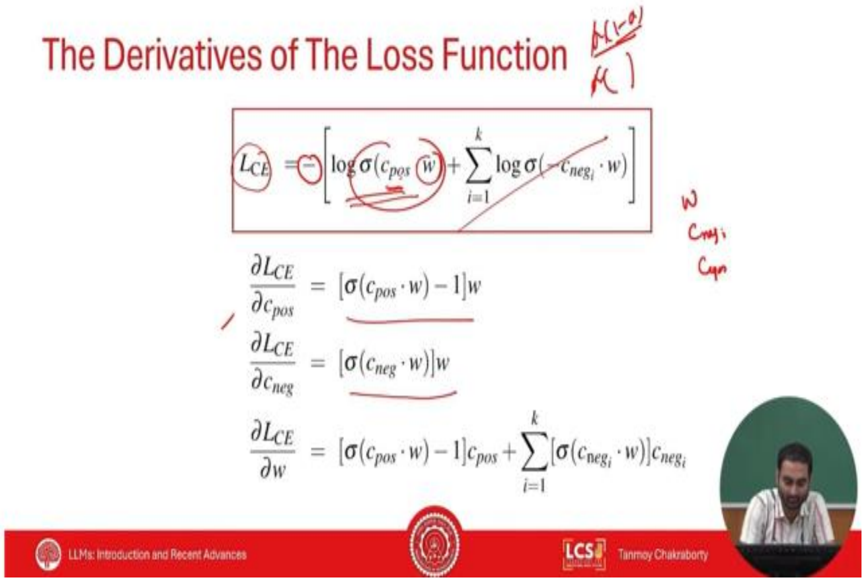
And what are the parameters? Parameters are C positive, C negative, W. By the way, C positive, C negative, these are just notations. Both of them are C. Just for our understanding, C positives are those instances which are positive and C negative are those instances which are negative. So in this bulky embedding matrix, we have only W and C or whatever T and C.



Reminder: Gradient Descent

- At each step
  - Direction: We move in the reverse direction from the gradient of the loss function
  - Magnitude: we move the value of this gradient $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate η**
  - Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$



The Derivatives of The Loss Function

$$\boxed{L_{CE} = -\left[\log\sigma(c_{pos}\cdot w) + \sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]}$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = [\sigma(c_{pos}\cdot w)-1]w$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = [\sigma(c_{neg}\cdot w)]w$$

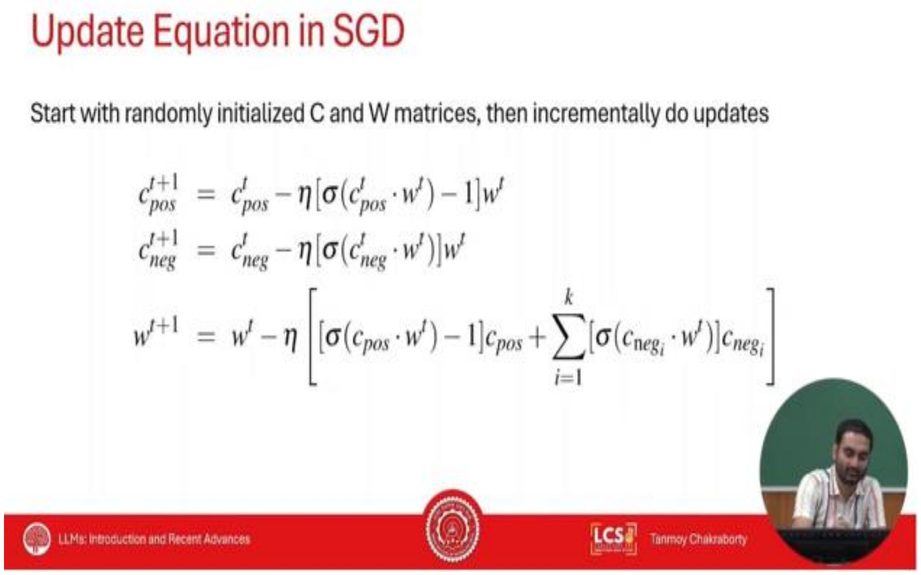
$$\frac{\partial L_{CE}}{\partial w} = [\sigma(c_{pos}\cdot w)-1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i}\cdot w)]c_{neg_i}$$

LLMs: Introduction and Recent Advances
LCS2
Tanmoy Chakraborty

Okay as I already mentioned and I will use gradient descent to update this. So this is the loss function okay, there are parameters w, c, neg and cpos and we take the derivative, let us look at the derivative, a very simple state for our derivative.

You take the derivative of this with respect to cpos okay. So this part will be ignored, Right we only look at this part log of x 1 by x right So this will be 1 by 1 by sigmoid right and then we take the derivative of this right. What is the derivative of sigmoid? sigmoid into 1 minus sigmoid so sigmoid into 1 minus sigmoid sigmoid sigmoid will cancel out right 1 minus sigmoid will basically remain now we will already have a negative here. So this will be sigmoid minus 1 and of course there is a W. So this is the derivative.

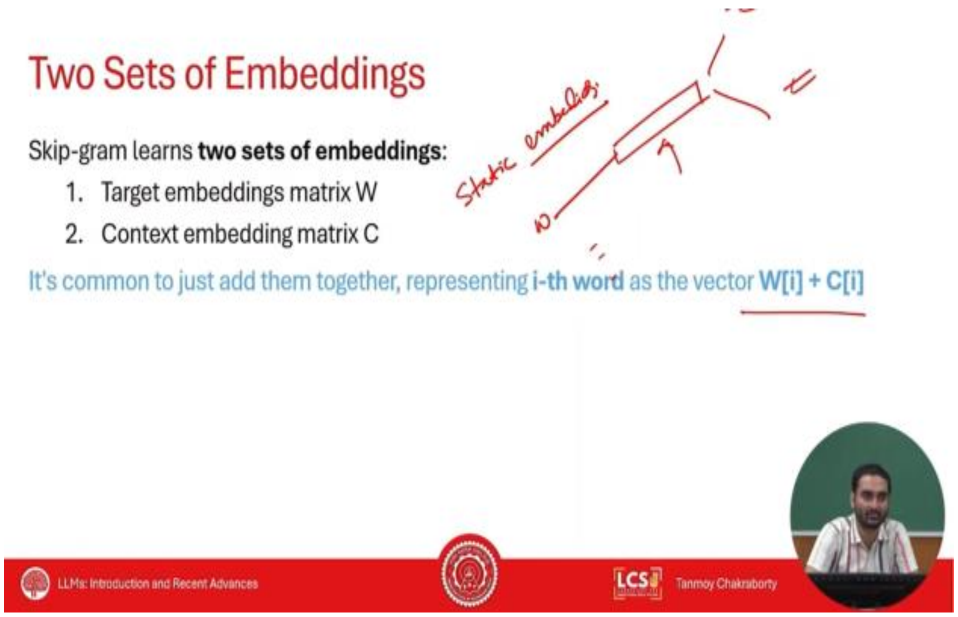
Similarly if you do a small calculation here, this will be the derivative with respect to C neg and this will be the derivative with respect to W.



Update Equation in SGD

Start with randomly initialized C and W matrices, then incrementally do updates

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta[\sigma(c_{pos}^{t} \cdot w^{t}) - 1]w^{t}$$
$$c_{neg}^{t+1} = c_{neg}^{t} - \eta[\sigma(c_{neg}^{t} \cdot w^{t})]w^{t}$$
$$w^{t+1} = w^{t} - \eta\left[[\sigma(c_{pos} \cdot w^{t}) - 1]c_{pos} + \sum_{i=1}^{k}[\sigma(c_{neg_i} \cdot w^{t})]c_{neg_i}\right]$$



Two Sets of Embeddings

Skip-gram learns **two sets of embeddings**:

1. Target embeddings matrix W
2. Context embedding matrix C

It's common to just add them together, representing **i-th word** as the vector **W[i] + C[i]**

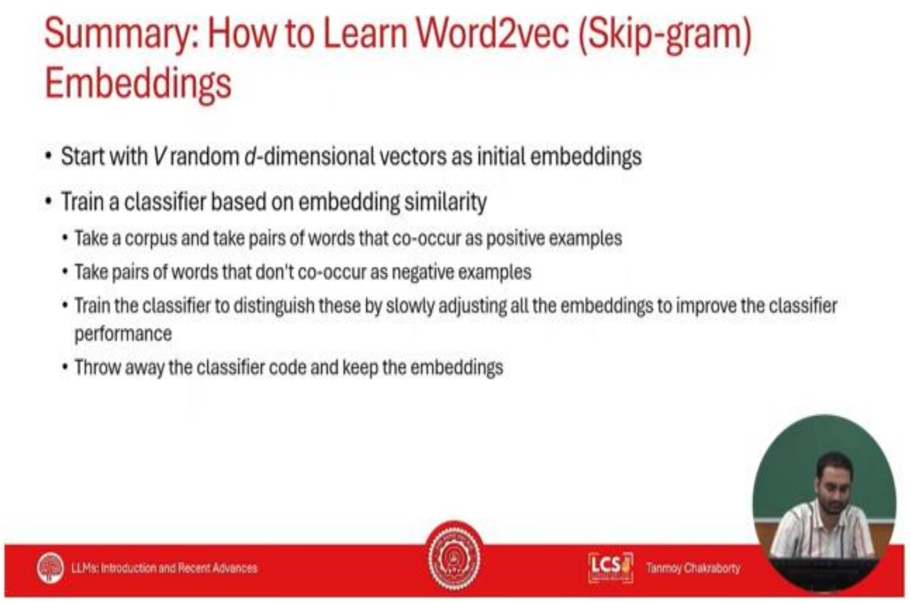
Static embedding
W

Once we obtain the derivative, these are the update rules. The positive vectors will be updated using this formula, the negative vectors will be updated using this formula and W will be updated using this formula. Now we obtain two matrices W and C, I mentioned multiple times two matrices W and C. Now, you can ask which one I should choose, right? The best option is that you choose two things separately and see which one is better.

The default option is that you take the sum of apricot using apricot acting as a context, apricot acting as a target. You take the sum of both the vectors and that's going to be your final vector. What to make is a static embedding. Remember this, it's a static embedding, meaning that for every word w, you will only obtain one vector.

I mean, of course, you can say two vectors, but ultimately one vector. It is not the case that for every senses of this word, you will have different vectors. Modern approaches like both kind of methods, there you will see that we will get contextual embeddings.

Okay. Okay.



Summary: How to Learn Word2vec (Skip-gram) Embeddings

- Start with *V* random *d*-dimensional vectors as initial embeddings
- Train a classifier based on embedding similarity
  - Take a corpus and take pairs of words that co-occur as positive examples
  - Take pairs of words that don't co-occur as negative examples
  - Train the classifier to distinguish these by slowly adjusting all the embeddings to improve the classifier performance
  - Throw away the classifier code and keep the embeddings

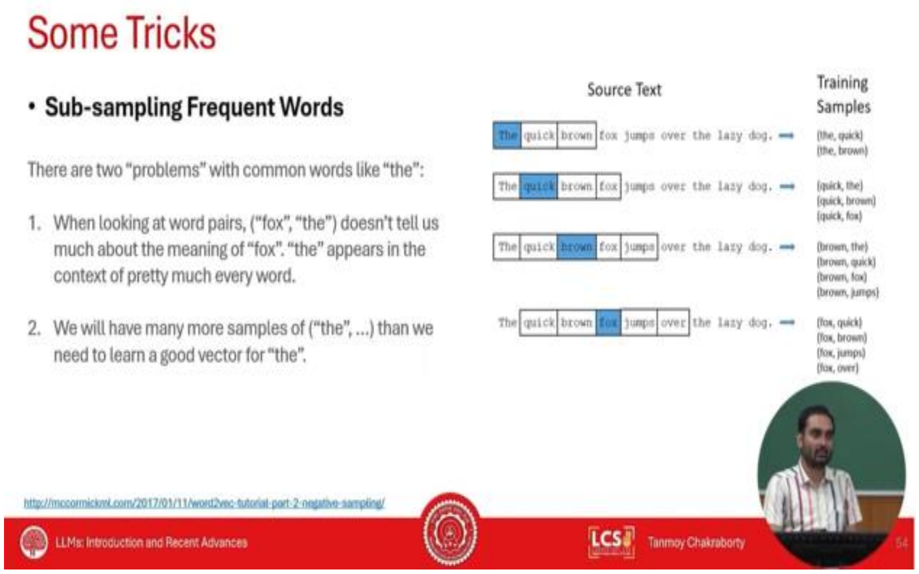
LLMs: Introduction and Recent Advances

LCS2

Tanmoy Chakraborty

So summary, this is a summary slide. We start with random initialization of this entire matrix. What's the size of the matrix? D dimension V plus V. So 2V times D matrix. We start with the random initialization and then I move the sliding window. Every movement I will curate positive instances, negative instances, k times more negative instances.

I will run the logistic regression, update the parameters. I can move, run and so on and so forth. Okay, so this is the prediction-based approach. What is the difference between a count-based approach and a prediction-based approach? Count-based approaches, including tf idf raw count, and prediction best approaches like what to wake right what are the differences count based approaches uh create this matrix one time scans you just scan all the document the entire document and create the matrix and you are done right So it preserve the statistics very very very carefully whereas in case of what to get there is no such thing right count based approaches the negative part is that as the size increases the matrix size will increase and you keep you need to recompute all the entries. Here as the size increases you may just need to learn or you may need to run the logistic regression one more time okay.



Some Tricks

- **Sub-sampling Frequent Words**

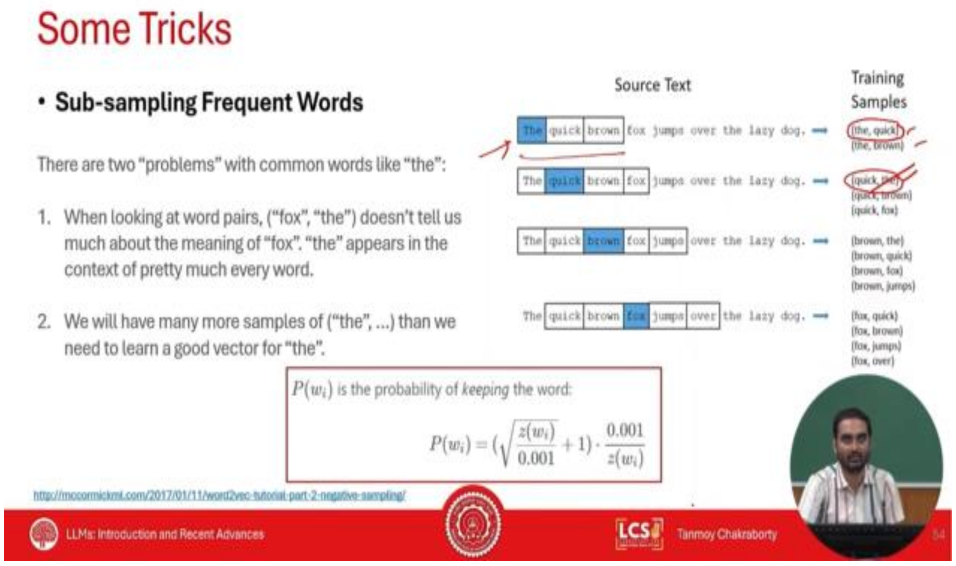
There are two "problems" with common words like "the":

1. When looking at word pairs, ("fox", "the") doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much every word.
2. We will have many more samples of ("the", ...) than we need to learn a good vector for "the".

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

LLMs: Introduction and Recent Advances LCS2 Tanmoy Chakraborty 54



Some Tricks

- **Sub-sampling Frequent Words**

There are two "problems" with common words like "the":

1. When looking at word pairs, ("fox", "the") doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much every word.
2. We will have many more samples of ("the", ...) than we need to learn a good vector for "the".

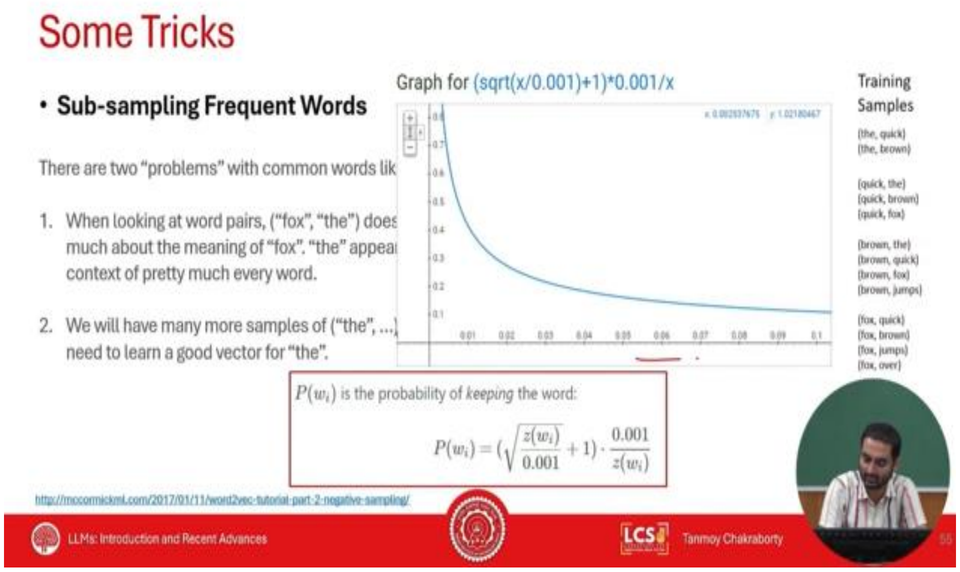
Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

LLMs: Introduction and Recent Advances LCS2 Tanmoy Chakraborty 54



Some Tricks

- **Sub-sampling Frequent Words**

There are two "problems" with common words lik

1. When looking at word pairs, ("fox", "the") does much about the meaning of "fox". "the" appea context of pretty much every word.
2. We will have many more samples of ("the", ...) need to learn a good vector for "the".

Graph for (sqrt(x/0.001)+1)*0.001/x

Training Samples

(the, quick)
(the, brown)

(quick, the)
(quick, brown)
(quick, fox)

(brown, the)
(brown, quick)
(brown, fox)
(brown, jumps)

(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)

$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = (\sqrt{\frac{z(w_i)}{0.001}} + 1) \cdot \frac{0.001}{z(w_i)}$$

http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

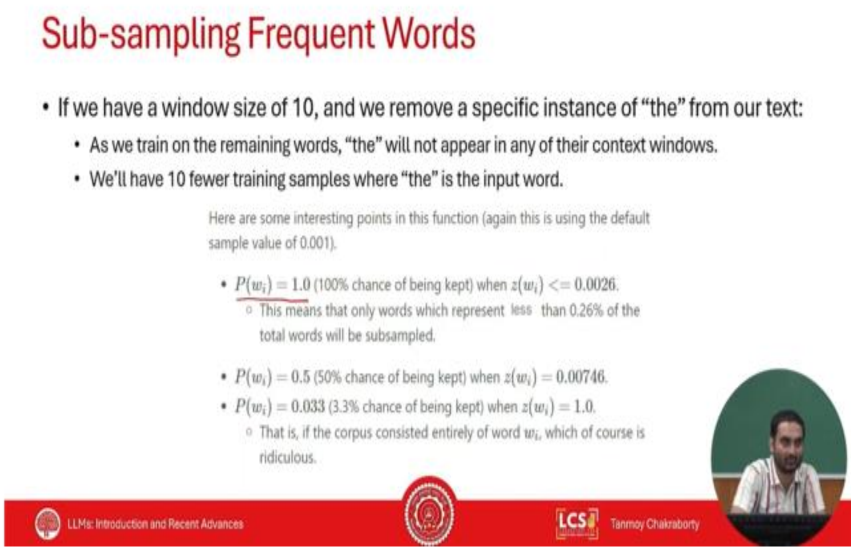
LLMs: Introduction and Recent Advances | LCS | Tanmoy Chakraborty | 55

So Couple of small tricks that this what to make paper considered. Another tricks that they considered is called subsampling frequent words. Let's say this is the sentence. The quick brown fox jumps over the lazy dog. Okay. And this is my window. What are the samples I can curate? The quick, the brown, right? From here, let's say quick, quick the, quick brown, quick fox and so on and so forth. Right? If you look at here very carefully, quick, the quick and quick the, they are basically the same. Isn't it? Because ultimately we will take the dot product of two vectors. So I can ignore one of these instances. The second problem is that the word the, right? The second problem is that the word the, right? is not at all a very meaningful word because it is unclear whether the word the acts as an article, I mean acts as a meaningful article here or the literal meaning of the is useful in this context or not. So what we do here in sub-sampling is that we use another probability.

And with this probability, I will remove some of the words from the documents. And I will not consider those words while curating the positive samples and negative samples. What does it mean? Now, this is the probability. This is, again, a kind of a unique probability, but small changes here and there. So p of w is a probability of a particular word. This is what is Z of w? Z of w is the raw frequency of the word wi and this is some sort of complicated formula that they proposed without much rationally behind it right but you can also think of many such formula which at least which can guarantee that this trend you know remains as it is.

What is this trend? It says so the x-axis is zwi and y-axis is pwi, okay. So if a word is less likely in terms of frequency they will be sampled more and if it is highly likely they will be sampled less. Meaning that if a word appears a lot of times like the word the, you will ignore or you will remove that word from the document. If it appears less number of times, you will keep those words.

When I say remove a word from the document means with certain probability, not all the times, with certain probability.



Sub-sampling Frequent Words

- If we have a window size of 10, and we remove a specific instance of "the" from our text:
  - As we train on the remaining words, "the" will not appear in any of their context windows.
  - We'll have 10 fewer training samples where "the" is the input word.

Here are some interesting points in this function (again this is using the default sample value of 0.001).

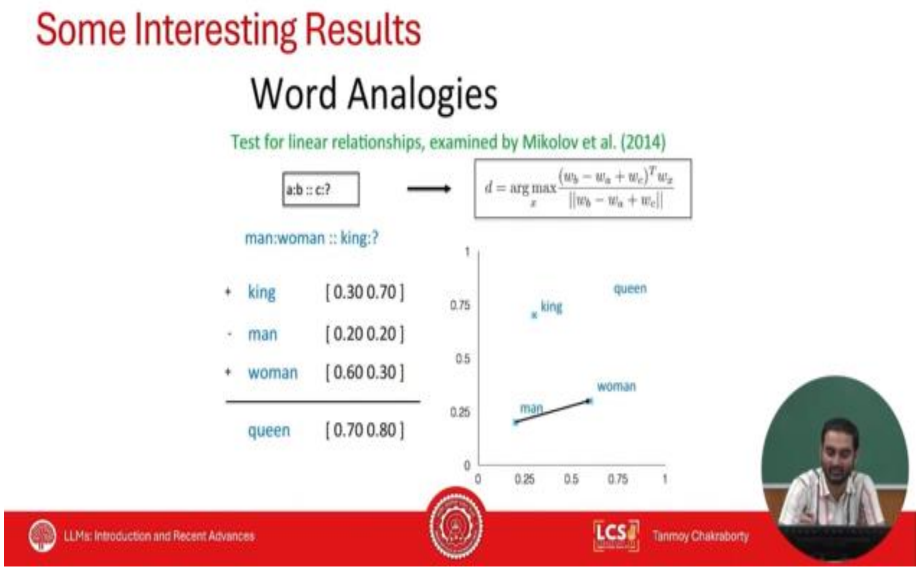
- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
  - This means that only words which represent less than 0.26% of the total words will be subsampled.
- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
  - That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

LLMs: Introduction and Recent Advances

LCS Tanmoy Chakraborty

So let us look at some of the values. So in order to make the value of p to be 1, if you really want to make a probability to be 1, the frequency should be 0.26%, less than 0.26%. So it means that if a word appears in a document 0.26% of time, it is highly likely with probability 1, you will sample that word. If the word appears with this probability, with 50% chance, you will sample that word. If a word appears 100%, what does it mean? It means that there is only one word in the document, right? I mean, there is only one word appearing multiple times in the document, there is no other word.

So the chance that this word will be sampled is 0.033 probability. This is just a hack to remove frequently appearing words from the training instances.



Some Interesting Results

Word Analogies

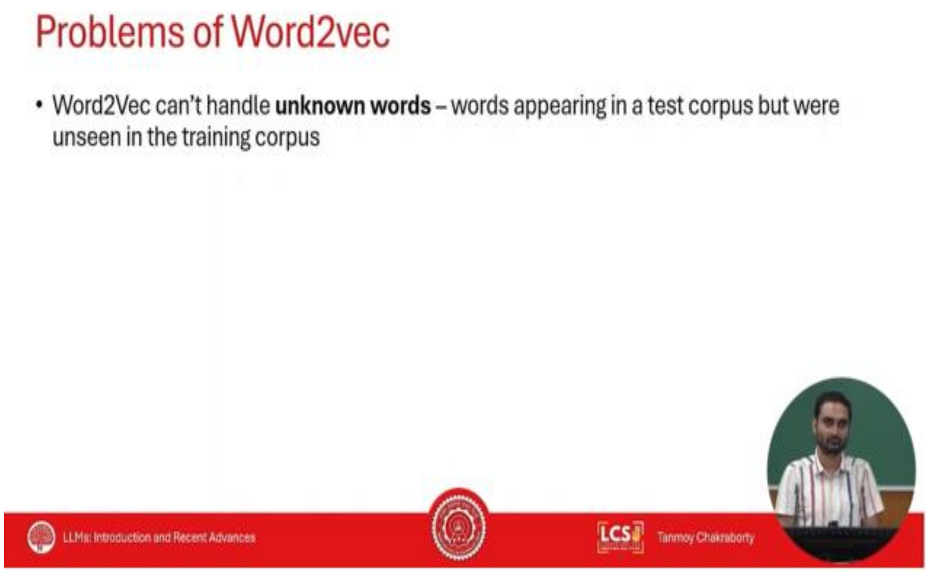
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:? $\longrightarrow$ $d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty



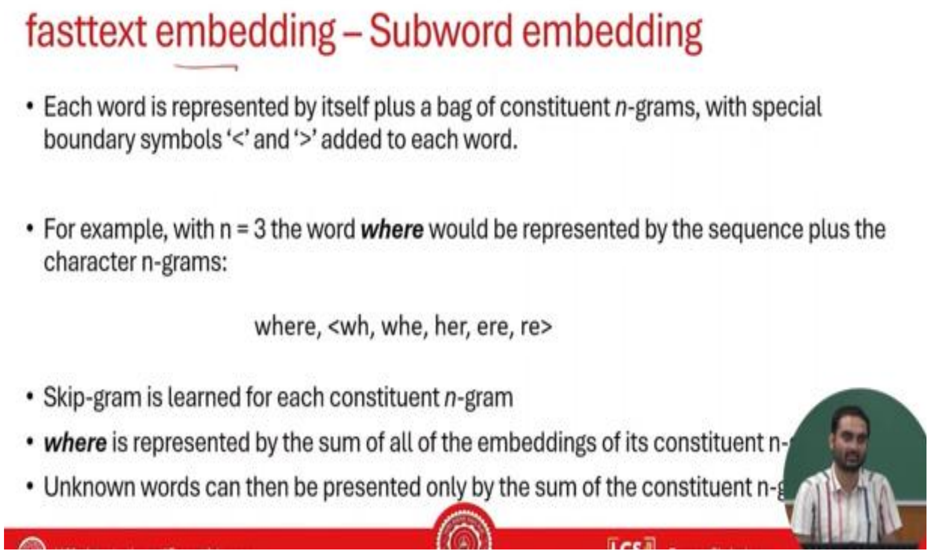
Problems of Word2vec

- Word2Vec can't handle **unknown words** – words appearing in a test corpus but were unseen in the training corpus

LLMs: Introduction and Recent Advances LCS Tanmoy Chakraborty

That's all about what to wait right this is a very nice analogy that they showed in the paper that with now you have all these embeddings right. So with this embeddings, now you can do all sorts of what analogy test, right. In fact you can do linear operations here. There is a beautiful example, you take the embedding of king and you subtract the embedding of man from the embedding of king. So king the word king has an embedding, the word man also has an embedding Okay, you subtract this from the king, you get a resultant embedding right So and then you add the embedding of woman, right to that resultant embedding this will be equivalent to the embedding of queen.

Right so you subtract man from king add woman you get the embedding of queen What are the problems of what embedding the first problem is- this is static approach right it doesn't consider different senses of it, the second problem is that when an unknown word comes in the test set, what embedding is already trained on a training set, You have a dictionary where the embeddings are stored, already stored. Now in a test set, when you see a new word which did not appear in the training set, how do you get the embedding of this? You will not get it.


fasttext embedding – Subword embedding

- Each word is represented by itself plus a bag of constituent *n*-grams, with special boundary symbols '<' and '>' added to each word.
- For example, with n = 3 the word ***where*** would be represented by the sequence plus the character n-grams:

where, <wh, whe, her, ere, re>

- Skip-gram is learned for each constituent *n*-gram
- ***where*** is represented by the sum of all of the embeddings of its constituent n-
- Unknown words can then be presented only by the sum of the constituent n-

To address this problem, what people suggested is something called subword embedding. So from word, I move to subword. What is a subword? Subword is a sequence of characters.

Right. So the method that was proposed by Facebook is called fast text. Fast text is a subword embedding method. Right. What is the subword embedding method? Very same as skip gram. Right. So here, and again, I will not go into details of this. So the idea is that In skipgram, what embedding method, in word2vec, what is our vocabulary? Our vocabulary is the set of words, right? Here, the vocabulary is set of subwords. And how do we define a subword? A subword is defined by n-grams. Right let us say the word where w h e r e here if the n-gram size is three I will take wh whe her ere.

These are three sub words right along with this I will use this angle bracket as the start of the word and the end of the word and those angle brackets are also considered as characters. So you have this as a n-gram, as a 3-gram. This is also 3-gram character, 3-gram character, 3-gram character, 3-gram character, and the word itself. So for every word, I have a set of subwords, that word itself, and the other n-grams, n-gram characters.

Once we got it, I will just run the script. I will just run the schibram and schibram will give you sub word level embeddings. Now let's say a new word comes in which is let's say nifty. So the hope is that nif for this sub word I have an embedding. This n-gram character, this character n-gram has appeared maybe, right? So I will take the sub-word embedding of nif, I will take the sub-word embedding of ift, I will take the sub-word embedding of fty, right? And of course this angle brackets, right? And I will sum them up, okay? So this is the idea of fast text. I will stop here. and maybe in the next lecture I will start discussing glove embedding. Thank you.