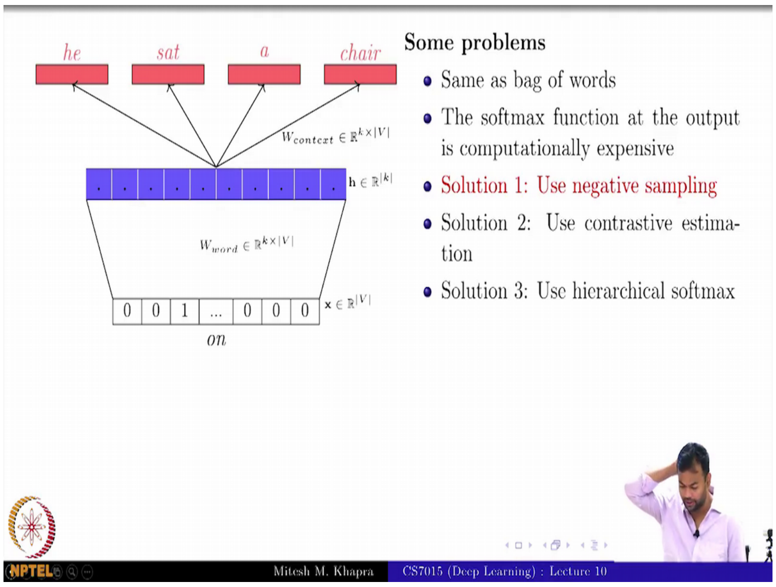
Deep Learning
Prof. Mitesh M Khapra
Department of Computer Science Engineering
Indian Institute of Technology, Madras

Lecture – 10
Skip-Gram Model (Contd.)

(Refer Slide Time: 00:11)



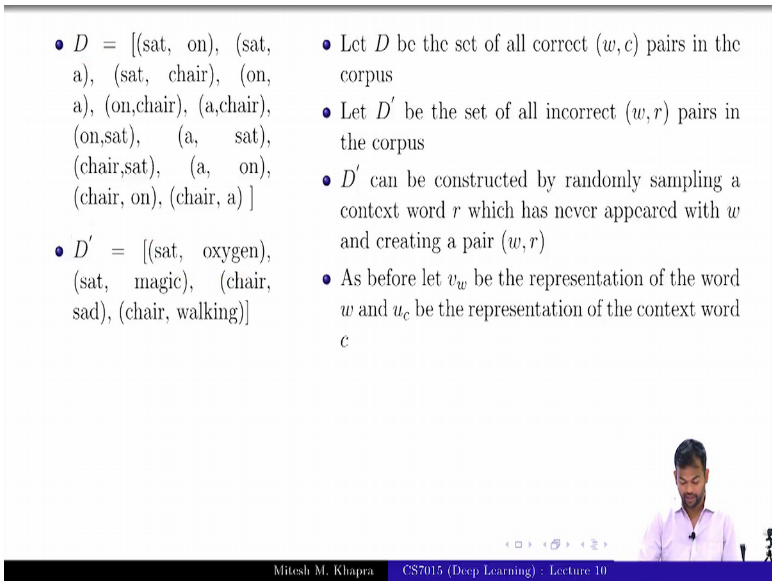
he
sat
a
chair
$W_{context} \in \mathbb{R}^{k \times |V|}$
$h \in \mathbb{R}^{|k|}$
$W_{word} \in \mathbb{R}^{k \times |V|}$
0 0 1 ... 0 0 0
$x \in \mathbb{R}^{|V|}$
on
Mitesh M. Khapra
CS7015 (Deep Learning) : Lecture 10

Some problems

- Same as bag of words
- The softmax function at the output is computationally expensive
- Solution 1: Use negative sampling
- Solution 2: Use contrastive estimation
- Solution 3: Use hierarchical softmax

So, what I will do is I will quickly, go over what we were doing yesterday. And then by the time people come in we can start with the new stuff right, So, we were looking at. So, that is so this needs to be corrected someone who pointed out yesterday, same as bag of words. It should be same problems as the bag of words model right. So, we are trying to fix this problem where we have this large softmax computation which is very inefficient, and you wanted ways of getting rid of that. So, the first thing that we were looking at is using negative sampling right.

(Refer Slide Time: 00:47)



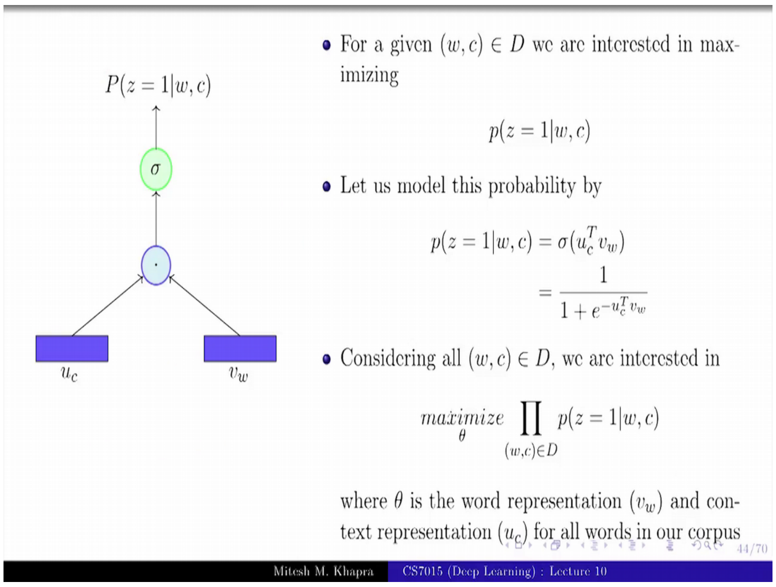
- D = [(sat, on), (sat, a), (sat, chair), (on, a), (on,chair), (a,chair), (on,sat), (a, sat), (chair,sat), (a, on), (chair, on), (chair, a)]
- D' = [(sat, oxygen), (sat, magic), (chair, sad), (chair, walking)]
- Let D be the set of all correct (w, c) pairs in the corpus
- Let D' be the set of all incorrect (w, r) pairs in the corpus
- D' can be constructed by randomly sampling a context word r which has never appeared with w and creating a pair (w, r)
- As before let v_w be the representation of the word w and u_c be the representation of the context word c

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 10

And here the key idea was to con construct this D and D prime, where D prime was the random corpus and D was a true corpus right.

And how do you create this random corpus is something that, was left at the end and which I need to go over today.

(Refer Slide Time: 01:05)



- For a given $(w, c) \in D$ we are interested in maximizing

$$p(z = 1|w, c)$$

- Let us model this probability by

$$p(z = 1|w, c) = \sigma(u_c^T v_w) = \frac{1}{1 + e^{-u_c^T v_w}}$$

- Considering all $(w, c) \in D$, we are interested in

$$\underset{\theta}{maximize} \prod_{(w,c)\in D} p(z = 1|w, c)$$

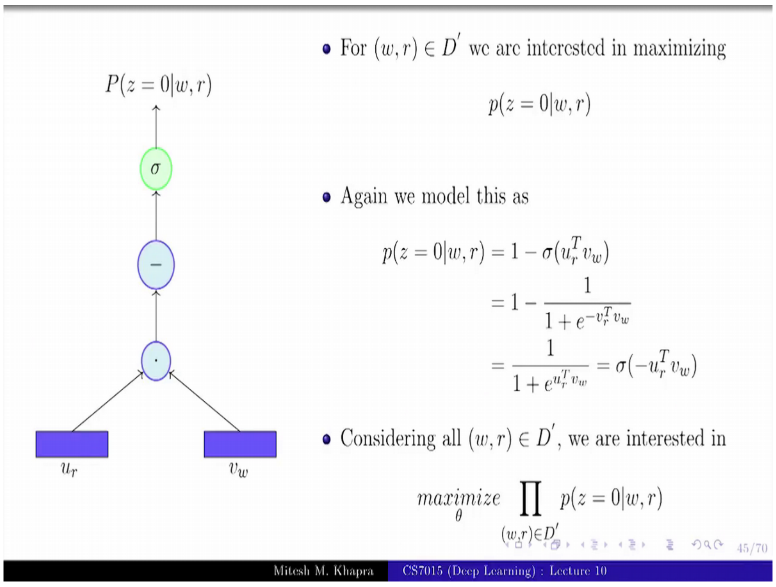
where θ is the word representation (v_w) and context representation (u_c) for all words in our corpus

44/70

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 10

So, I will go over that and then we realize that this actually could be modeled using such a network, where you take the dot product between the word representations, right?

(Refer Slide Time: 01:23)



$P(z = 0|w, r)$
σ
$-$
$\cdot$
u_r
v_w

- For $(w, r) \in D'$ we are interested in maximizing

$$p(z = 0|w, r)$$

- Again we model this as

$$p(z = 0|w, r) = 1 - \sigma(u_r^T v_w)$$
$$= 1 - \frac{1}{1 + e^{-v_r^T v_w}}$$
$$= \frac{1}{1 + e^{u_r^T v_w}} = \sigma(-u_r^T v_w)$$

- Considering all $(w, r) \in D'$, we are interested in

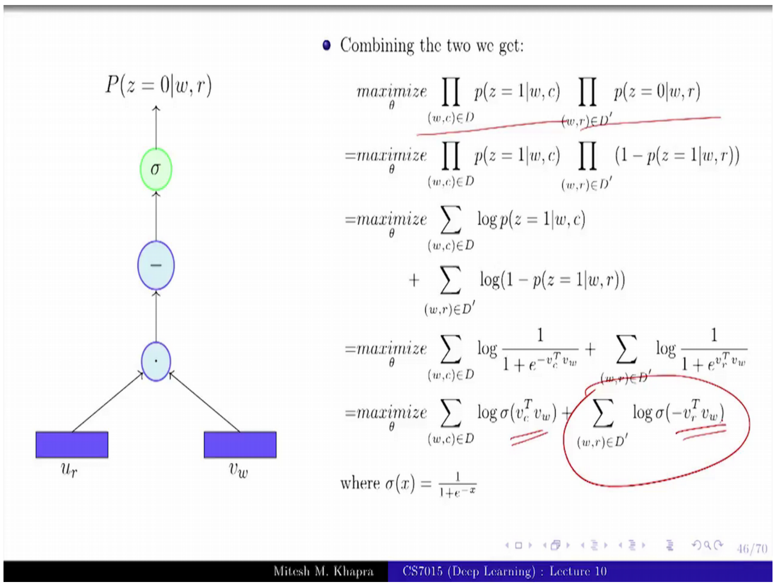
$$\underset{\theta}{maximize} \prod_{(w,r) \in D'} p(z = 0|w, r)$$

45/70

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 10

And try to maximize this to dot product for all the correct pairs by setting up your loss function accordingly.

(Refer Slide Time: 01:26)



$P(z = 0|w, r)$
σ
$-$
$\cdot$
u_r
v_w

- Combining the two we get:

$$\underset{\theta}{maximize} \prod_{(w,c) \in D} p(z = 1|w, c) \prod_{(w,r) \in D'} p(z = 0|w, r)$$
$$= \underset{\theta}{maximize} \prod_{(w,c) \in D} p(z = 1|w, c) \prod_{(w,r) \in D'} (1 - p(z = 1|w, r))$$
$$= \underset{\theta}{maximize} \sum_{(w,c) \in D} \log p(z = 1|w, c)$$
$$+ \sum_{(w,r) \in D'} \log(1 - p(z = 1|w, r))$$
$$= \underset{\theta}{maximize} \sum_{(w,c) \in D} \log \frac{1}{1 + e^{-v_c^T v_w}} + \sum_{(w,r) \in D'} \log \frac{1}{1 + e^{v_r^T v_w}}$$
$$= \underset{\theta}{maximize} \sum_{(w,c) \in D} \log \sigma(\underline{\underline{v_c^T v_w}}) + \sum_{(w,r) \in D'} \log \sigma(\underline{\underline{-v_r^T v_w}})$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$

46/70

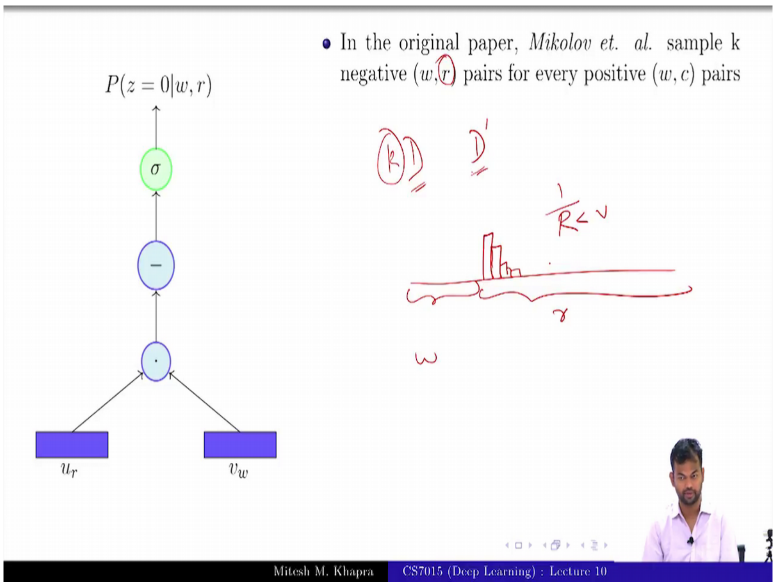
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 10

And try to maximize or rather minimize this dot product, minimize this dot product for all the incorrect pairs by again setting the objective function appropriately right.

So, we had this objective function where we want to maximize the probability that the pair is correct for the correct pairs and maximize the probability, that the pair is incorrect for the incorrect pairs, and both these probabilities we had modeled using a sigmoid function, and inside the sigmoid function we had the dot product between the corresponding representations.

So, the net effect is you either maximize the dot product of the correct pairs, or minimize the dot product of the, or rather and in minimize the dot product of the incorrect pairs fine.

(Refer Slide Time: 02:01)



- In the original paper, *Mikolov et. al.* sample k negative (w, r) pairs for every positive (w, c) pairs

And then so now today the part which was remaining about the comparison between D and D prime; so, what I was saying last time is that D prime is actually k times D; that means, in sample more negative examples than positive examples. So, if you think about it actually the number of negative examples in the language is much, much more than a number of positive examples. Let us say if you have 50 k words in your vocabulary most of them do not appear together, right? So, that number is actually very, very large as compared to the number of words which can occur together right.

So, how do you account for this natural imbalance? So, they said that if you keep it same, then we are saying that the size of D prime and D is going to be same; that means, the words which appear together and not to appear together we are keeping those 2 corpora as the same. So, that does not sound reasonable; so, they decided that we will keep it k times. Now this k was a hyper parameter which was tuned based on the data that they had. And can you guess, how they would have tuned it, no what do you tune your parameters on, what did how did you tune your parameters for the back propagation of the word, no using what ?

Student: (Refer Time: 03:12).

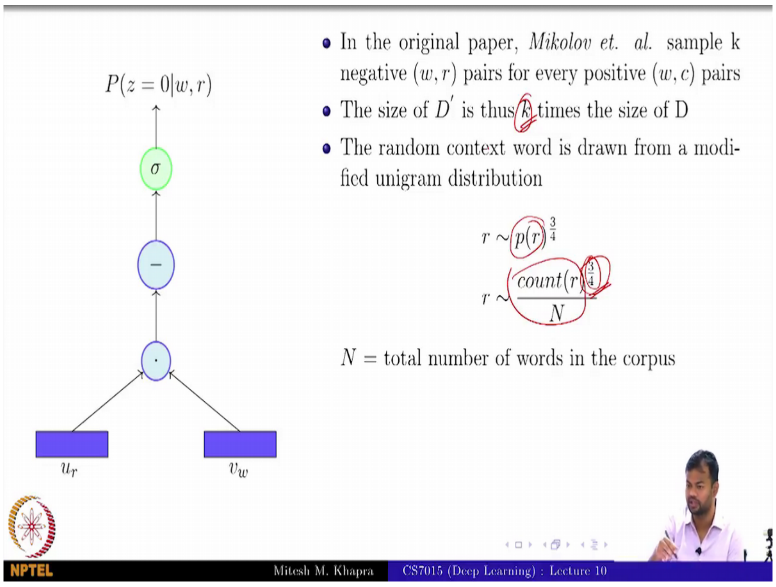
A validation set is it too early in the morning, it fine validation set. So, they might have had some validation set. And if you look at the original word to word code which someone had posted yesterday, which allows you to compute the distance matrix right. So, you could what you could do is you could learn these representations, take a few pairs of words. And take a few pairs of good words right say cat and dog or cat and feline and so on. And also bad words like cat and truck bad combinations rather. And see if the distance between cat and truck is much, higher than the distance between cat and feline or cat and dog, right?

So, you select that k which gives you the best performance on your validation set and the validation set here would essentially be to find if you get good representations for word pairs that you care about and for word pairs that you do not care about. Now the other thing was how do you create this R? So, you have v words in the vocabulary you are looking at one of those w.

You know that some of those have appeared with w in some context, but there is this large set which has not appeared with w in any context right. So, you are going to draw are from this set and the simplest thing to do would be to just draw the uniform distribution; that means, all words and let us call this suppose there are capital R words here all of these words could be drawn from using the quality 1 by R right, where R is less than v.

Is that fine? That is one way of doing it just randomly pick any word from the remaining words and put it a pair it with w, but you would also want to account for the individual frequencies of those words, right? If the word is actually very frequent pair it up more with w. If it is not frequent do not pair it up enough. Does that make sense? So, I could actually use the frequencies of each of these words. And sample according to that frequency right? Instead of using a unigram distribution.

(Refer Slide Time: 05:15)



- In the original paper, *Mikolov et. al.* sample k negative (w, r) pairs for every positive (w, c) pairs
- The size of D' is thus k times the size of D
- The random context word is drawn from a modified unigram distribution

$$r \sim p(r)^{\frac{3}{4}}$$

$$r \sim \frac{count(r)}{N}^{\frac{3}{4}}$$

N = total number of words in the corpus

So, they did something similar, but they had this hyper parameter again. So, basically I was sampling using the probability of r, which is equal to count of r divided by the number of times number of all the words in the corpus. That is actually the frequency of r divided by the total number of words in the corpus. So, instead of just taking that they had this wearied factor of 3 by 4; do you we realize that if you take this 3 by 4 you get the best performance.

So, let me just make a few comments on that. So, the original code of or rather the original skip gram or the bag of words model, actually worked very well and it kind of hard a lot of seminal effect or a lot of revolutionary effect on the field of NLP right. So now, everyone started talking about word vectors, and how you can use this meaningful representations of words, as features for various down steep NLP thus right.

So, at the end in NLP what you are doing is you are collecting of a bunch of words a document or a sentence or something and trying to do some processing on that. Now earlier used to construct features out of these sentences using some handcrafted features, but now someone said that there is this automatic way of constructing word features right, which is using this method. So, people really bought onto that idea and a lot of work started happening. And then later on at the end of the course we will see something that what it eventually led to.

But later on when people started analyzing this more carefully right, they realized that the original word to like implementation, had a lot of these heuristics or lot of these parameters, which need to be really tuned to the core for it to be able to compete with SVD right. So, that is what we look at the end. So, SVD was already one way of computing word representations ah, which while popular was not so popular it was used for various reasons, but it was not like every NPL application is using SVD representations right, but now it is almost like every NPL application is using word representations.

So, later on we will see that some of these things like 3 by 4 or k the value of k, the value of learning rate and some other hyper parameters. If you really tuned them very, very well it is only then that as this word to make algorithm can beat the world representations learned by SVD, or rather the other thing that if you introduce some parameters in SVD and tune them, because remember for SVD there was no tuning right, we just got a solution. We just had the closed form solution which is the Eigen vectors.

But you could do some things for creating the co-occurrence matrix. If you introduce some factor there which is also looks like this 3 by 4 or something like that, or if you also introduced something which looks like a k. Then you will be able to get the same kind of representations or equally powerful representations from SVD as what you get from word to it. So, that is why I am stressing on these hyper parameters there is some significance of those.