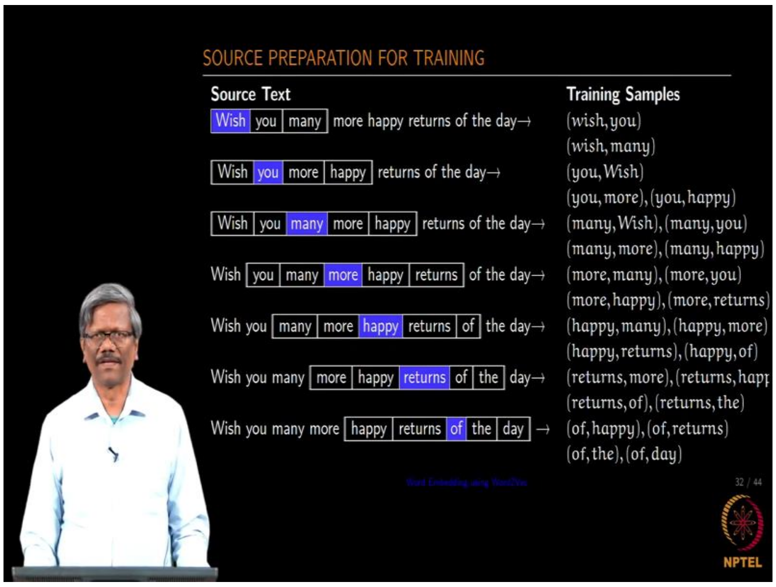
Applied Natural Language Processing
Prof. Ramaseshan Ramachandran
Department of Computer Science and Engineering
Chennai Mathematical Institute

Lecture - 46
Reduction of complexity-sub-sampling, Negative sampling

(Refer Slide Time: 00:15)



SOURCE PREPARATION FOR TRAINING

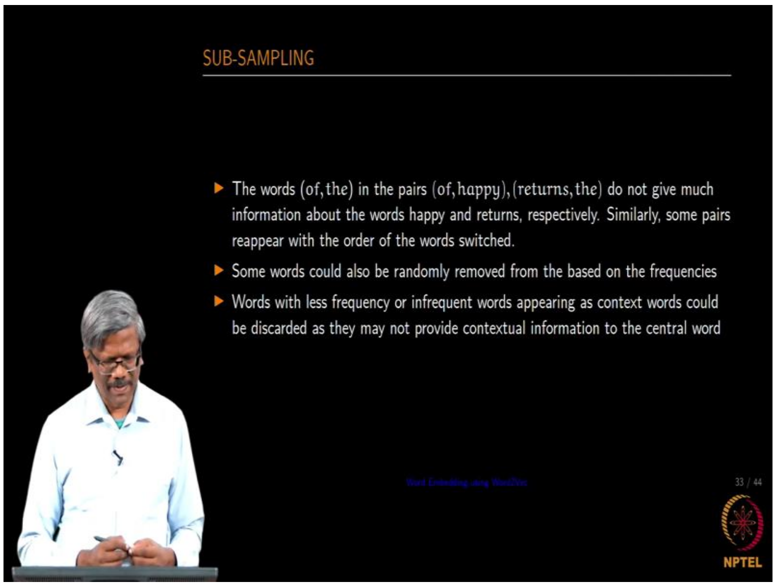
Source Text	Training Samples
Wish you many more happy returns of the day→	(wish, you)
	(wish, many)
Wish you more happy returns of the day→	(you, Wish)
	(you, more), (you, happy)
Wish you many more happy returns of the day→	(many, Wish), (many, you)
	(many, more), (many, happy)
Wish you many more happy returns of the day→	(more, many), (more, you)
	(more, happy), (more, returns)
Wish you many more happy returns of the day→	(happy, many), (happy, more)
	(happy, returns), (happy, of)
Wish you many more happy returns of the day→	(returns, more), (returns, happ
	(returns, of), (returns, the)
Wish you many more happy returns of the day →	(of, happy), (of, returns)
	(of, the), (of, day)

32 / 44

NPTEL

So, how do we really reduce the complexity of that? So, now, we need to start from the training samples right. So, this is the source preparation that I had shown earlier, if you look at this there are many bigrams that are repeating ok.

(Refer Slide Time: 00:35)



SUB-SAMPLING

- The words $(\mathrm{of}, \mathrm{the})$ in the pairs $(\mathrm{of}, happy), (returns, \mathrm{the})$ do not give much information about the words happy and returns, respectively. Similarly, some pairs reappear with the order of the words switched.
- Some words could also be randomly removed from the based on the frequencies
- Words with less frequency or infrequent words appearing as context words could be discarded as they may not provide contextual information to the central word

Word Embedding using Word2Vec 33 / 44

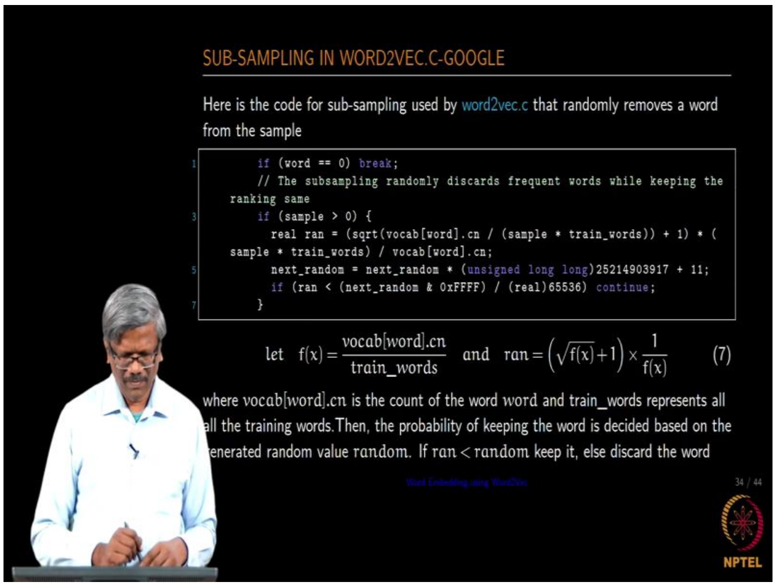
Let me show some examples, these are examples the words like the or the pairs of happy returns. Do not give much information about the words happy and return correctly. The context is not good enough in the bigram model. So, in the same fashion if you expand it to trigramcorrectly there could be some combination, where the context words will not give you any meaning to the target word that you are looking at. Or the target word will not give you any meaning to the context that you are looking at. Some words we can remove at random; similarly some pairs reappear with the order of the words switched ok.

So, you will find some of the words here as well for example, so here right, the words are switched. So, we can actually reduce this instead of keeping those two as the training sample we can reduce it to one. So, in this fashion if you find these kinds of duplicates even the words are switched within the bigram or trigram or five-gram that you are looking at for your context you can remove them. And then certain words like you know the combinations like of them do not really give any meaning to the processing the same word same here ok.

So, in the model this is just a simple example we do not normally use bigram for doing this training, we normally go with you know at least five words window correctly; so, this is called subsampling. So, when you remove these words from the samples, you get a smaller subset and then you use that smaller subset to train the networks. You know because of the size that increases along with the number of vocabulary you have right, so we have to somehow reduce the training samples.

So, this is one way to reduce the training sample. We can also remove certain words from the vocabulary based on the frequency. Suppose if some words you know appear only once in the entire corpus, that is not going to give us any context correct; so, we can probably remove that word from there ok. Again some as I mentioned earlier some of the words that appear as context words, could be discarded if they do not give any contextual information to the central word that we are talking about ok.

(Refer Slide Time: 03:37)



SUB-SAMPLING IN WORD2VEC.C-GOOGLE

Here is the code for sub-sampling used by word2vec.c that randomly removes a word from the sample

```
      if (word == 0) break;
      // The subsampling randomly discards frequent words while keeping the
  ranking same
      if (sample > 0) {
        real ran = (sqrt(vocab[word].cn / (sample * train_words)) + 1) * (
  sample * train_words) / vocab[word].cn;
        next_random = next_random * (unsigned long long)25214903917 + 11;
        if (ran < (next_random & 0xFFFF) / (real)65536) continue;
      }
```

$$\text{let} \quad f(x) = \frac{vocab[word].cn}{train\_words} \quad \text{and} \quad ran = \left(\sqrt{f(x)}+1\right) \times \frac{1}{f(x)} \tag{7}$$

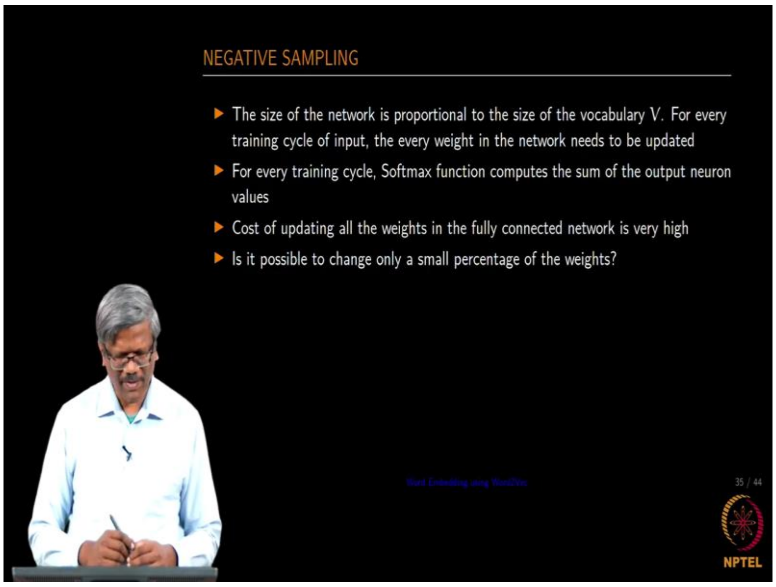
where $vocab[word].cn$ is the count of the word $word$ and train_words represents all all the training words.Then, the probability of keeping the word is decided based on the enerated random value $random$. If $ran < random$ keep it, else discard the word

Word Embedding using Word2Vec

34 / 44

So, now what I have in this slide is a real example that used in a C program. This is actually a C program written by Google to really demonstrate their word 2 vec model; this is implemented using Skip-gram. So, here they have used some subsampling mechanisms to remove the infrequently occurring words ok. So, if you really understand C; I would like you to go through this and then see how really they are utilizing the mechanism to really reduce the number of samples for the training alright.

(Refer Slide Time: 04:38)



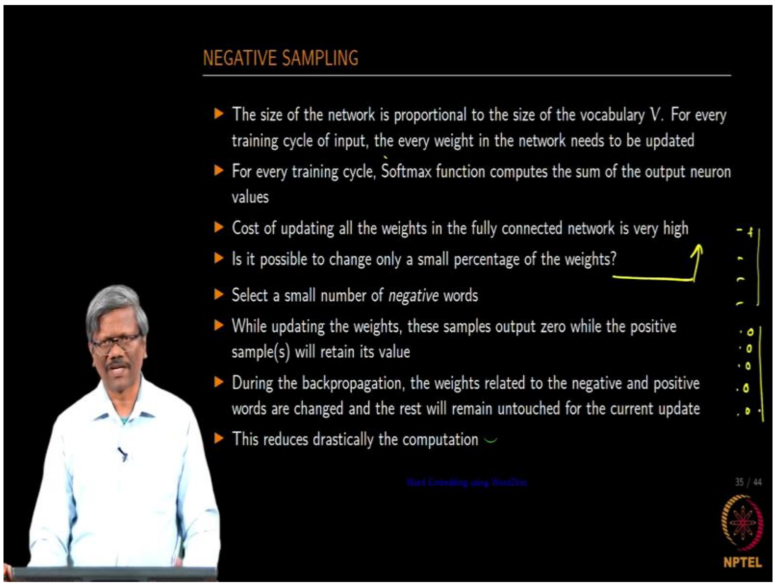
NEGATIVE SAMPLING

- The size of the network is proportional to the size of the vocabulary V. For every training cycle of input, the every weight in the network needs to be updated
- For every training cycle, Softmax function computes the sum of the output neuron values
- Cost of updating all the weights in the fully connected network is very high
- Is it possible to change only a small percentage of the weights?

35 / 44

So, there is another mechanism by which you want to minimize the computation right so that is called a negative sampling. I keep repeating this several times the computations of this model is very huge when the vocabulary size is very large ok. So, each and every weight updation costs money. So, we need to be able to reduce the number of operations in the entire backpropagation model. So, those are the ideas, which we are now using to reduce the number of computations this is one subsampling is one.

(Refer Slide Time: 05:27)



NEGATIVE SAMPLING

- The size of the network is proportional to the size of the vocabulary V. For every training cycle of input, the every weight in the network needs to be updated
- For every training cycle, Softmax function computes the sum of the output neuron values
- Cost of updating all the weights in the fully connected network is very high
- Is it possible to change only a small percentage of the weights?
- Select a small number of *negative* words
- While updating the weights, these samples output zero while the positive sample(s) will retain its value
- During the backpropagation, the weights related to the negative and positive words are changed and the rest will remain untouched for the current update
- This reduces drastically the computation

35 / 44

The second one is a negative sampling. So, why should I update all my weights every time when that does not really make any impact in the learning ok? So, which are those weights that I should not really bother about ok?

So, when you look at that right, the earning that happens only for the context words and the input word correct. The rest of the words there are some errors that happen and they really do not impact the training when we train the set of context words. And the target words or when you use them in the skip-gram model when you use the input word and try to predict the context word rest of the words do not really contribute anything to this.

So, can we really curtail the updating of the weights, by only choosing a certain number of weights to be updated and so on; so that is where negative sampling comes into play. So, in this case for every training cycle of input we update every weight right; that is what we have been doing. And then Softmax function computes the sum of the output neuron values, so it's again a huge computation for us right.

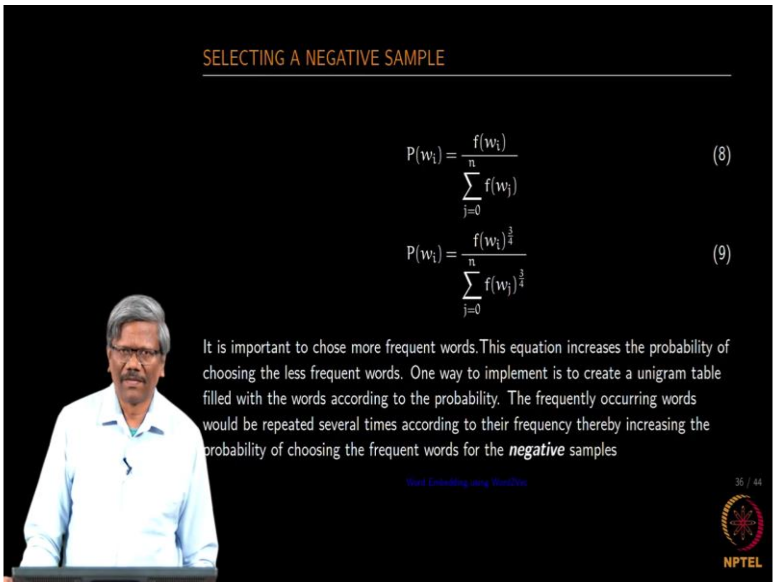
And then the cast of updating all the weights in the fully connected network is very high. So, is it possible to change only a smaller percentage of weights ok? So, that is where negative sampling comes into play and let us see how this is done. So, how do we really solve the problem that we have mentioned earlier right? So, now, select only a small number of negative words, I will talk about what that negative word ok; while updating the weights these samples output zero while the positive samples will retain the value.

For example if we have about five values that we want to retain, we call them positive samples. And then we select five more values and then make them as 0; because we want them to be 0. The number of operations that we are going to be performing is going to be only on these positive values and the negative samples right. So, when we do that we reduce drastically the computation, instead of computing every weight in the embedding as well as the context matrix, we are going to be only changing a certain number of weights. And this is what is given here.

While updating the weights, these samples output zero while the positive samples will retain the values. During the backpropagation, the weights related to the negative and the positive words are changed and the rest will remain untouched for the current update we do not touch any of the other weights; we only change these five or ten weights that we have chosen. So, this drastically reduces our computation is not it instead of updating for example, if you look at 1 million vocabularies by 300 hidden values, we have about 300 million weights to be updated every time correct that is for embedding weights as well as for context weights leave alone your Softmax computation and so on. So, we have reduced drastically by only choosing a certain number of negative words.

So, these negative words are nothing but words in the vocabulary itself. But these words do not really contribute anything to the context word; they are not one of those context words, they are chosen through some mechanism and they are taken and those weights are updated.

(Refer Slide Time: 10:00)



SELECTING A NEGATIVE SAMPLE

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} f(w_j)} \tag{8}$$

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f(w_j)^{\frac{3}{4}}} \tag{9}$$

It is important to chose more frequent words.This equation increases the probability of choosing the less frequent words. One way to implement is to create a unigram table filled with the words according to the probability. The frequently occurring words would be repeated several times according to their frequency thereby increasing the probability of choosing the frequent words for the ***negative*** samples

36 / 44

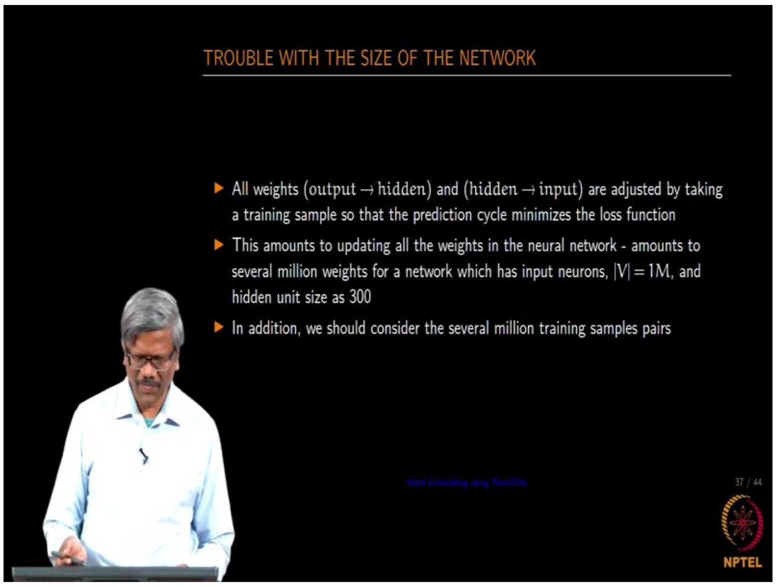
There are several mechanisms that are available and this is one mechanism. So, we want to be able to pick up only those most frequently appearing words as one of those negative samples ok. This I am sure you would know, it is the probability of a given word in the corpus right. So, we are applying some mechanism, which really boosts the probability of less frequent words and then reduces the probability of highly appearing words in the corpus.

$$P\left(w_j\right) = \frac{f((w_i)^{\frac{3}{4}}}{\sum_{j=0}^{n} f((w_i)^{\frac{3}{4}}}$$

And then there is another mechanism that you do is once you compute the probability and then find out how many times a particular word appears in the entire vocabulary. So, you have the frequency table as well. Using that frequency table you can find out how many times you want to repeat certain words in a table.

For example, you create a unigram table of size of the vocabulary; and then pick up the most frequently appearing word and then copy them in the unigram table several times according to the frequency of the words. And then use this formula to pick a word or the number of words that you want to use as the negative sample you got it. So, this is how you use the find the negative sample and then use it in our computation of updating weights.

(Refer Slide Time: 11:47)


TROUBLE WITH THE SIZE OF THE NETWORK

- All weights (output → hidden) and (hidden → input) are adjusted by taking a training sample so that the prediction cycle minimizes the loss function
- This amounts to updating all the weights in the neural network - amounts to several million weights for a network which has input neurons, $|V| = 1M$, and hidden unit size as 300
- In addition, we should consider the several million training samples pairs

37 / 44

So, as I mentioned earlier the number of weights is phenomenal right for the computation we want to be able to reduce that. For example, the word 2 vec given by Google uses 1 billion words and it contains about 30,000 to 40,000 words as vocabulary. And to train the network or to generate the word vector it takes about 40 minutes or 1 hour depending on your computer power.