

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

**Lecture – 47**  
**Binay tree, Hierarchical softmax**

(Refer Slide Time: 00:15)

**SOFTMAX**

Softmax is a normalized exponential function

$$P(C_k|x_j) = \frac{e^{a_k}}{\sum_k e^{a_k}} \text{ where } k=1, K \text{ (1)}$$

- ▶ Has a flat hierarchy with a probability value for every output node of depth = 1
- ▶ Normalized over the probabilities of all  $|V|$  words
- ▶ Error correction happens for every output → hidden units
- ▶ Huge costs if the vocabulary size  $|V|$  is of the order of several thousands

The diagram on the right shows a neural network architecture with layers: Input Layer, Embedding Vectors, Hidden Layer, Context Vectors, Output Layer, and Softmax Layer. Below the layers, weights  $w_1, w_2, \dots, w_y$  are shown, and a bar chart represents the probability distribution  $P(C_k)$  where  $\sum_{j=1}^y P(C_j) = 1$ .

In continuation of our optimization with respect to the neural weights and updation and so on we are going to be looking at how we can also reduce the computation from the Softmax layer and then I have just added one more layer for the sake of understanding which is the Softmax layer. On obtaining the values in the output layer we are applying this Softmax here right; the Softmax is a normalized exponential function as given in this equation, alright.

So, the normalization comes from the summation of all the word value that we have computed ok. So, each one will have the probability value here and the summation of all of these would lead to one this we know very well right. And this is a flat hierarchy; that means, we compute the probability for every word in the vocabulary once the output layer computation is complete, alright. So, the depth of this layer is 1 ok. So, keep this in mind and then here is where the error correction starts

So, from here we actually update the context vectors, based on how much the value moved from the actual target and then update the h values and then the embedding vector

values and we keep doing it until we finally, get an equilibrium where there is no more change that you can find ok. And then if you look at the Softmax layer at the bottom, so every time when you do them or when you compute the probability for each word, you will see this right, for every word this is being computed. So, assuming now that if you are going to be having a million words in your vocabulary, so we have to be doing this operation every time right, for every the probability has to be obtained using this equation right; that means we are going to be doing it one million times. So, we do the summation once and then taking the exponential for each and every output layer that we get will be done and that increases the computational complexity.

So this is again a very good place for you to optimize your computation, alright. So, we try to reduce the computational complexities by taking sub-sampling, removing some of the samples that are not really contributing to the understanding of the word. And then we applied negative sampling wherein we add 5 words that do not come as part of the context in the case of skip-gram and then necessarily make them as 0.

So, that the computation of the context vectors was done only for the positive context value and the negative samples; that means, we are doing only for 0.1 percent to 0.05 percent or even lower depending on what is your what is the number of negative samples that we are looking at. So, the third one that we are looking at is improving the computational capabilities with respect to doing something with the Softmax layer ok. So, we are going to be looking at what we are going to be doing in the Softmax layer in from the next layer onwards.

(Refer Slide Time: 03:59)

**BALANCED BINARY TREE**

Root

$w_1$   $w_2$   $w_3$   $w_4$   $w_5$   $w_6$

- ▶ Move the words into a binary tree - depth depends on the vocabulary
- ▶ The path to any word in the vocabulary is known ✓
- ▶ Traverse through the binary tree to reach any word
- ▶ At every step, make a binary decision

to reach the word

- ▶ The length to reach any word in a balanced tree is  $\log_2(|V|)$  ✓
- ▶ Words could be arranged using
  - ▶ random order
  - ▶ IS-A relationship
  - ▶ TF-IDF, frequency

Color  
Red is a color  
blue

Word Embedding using WordNet

2 / 13

NPTEL

So, before getting into how we can reduce it, let us keep those words in the vocabulary in a binary tree format ok. So, we are going to be looking at only the balanced binary tree, I am sure you know what binary tree is and a balanced binary tree is. The balanced binary tree is one where you will have a binary branch for every node and for example, for node 4 we have 2 leaves node 5 and so on. So, if we start with a root node, you have 2 nodes and then for the second node we have 4 and 5, for the third node 6 and 7 so; that means, the binary tree is balanced. I am sure most of you know about it, this is just to refresh your memory ok.

We place the words in the leaves ok; that means, all the words would be available in this form ok. And we know that how to reach each word from the root node ok. So, this is your root node and if you want to reach  $W_3$ , I have marked that in red. So, you take a left reach node 2, take a right reach node 5 and then you get  $W_3$ . So, the path is unique for each and every word. So, this is one way of constructing the binary tree. So, how do we get these in this order? So, there could be multiple ways of doing it; one is we can take the vocabulary and then place them at random ok.

So, there is no relationship between  $W_1$  and  $W_2$ , it is just placed at random. So, another way is to look at the word net where the words are arranged in the IS-A relationship, I am sure you remember the IS-A relationship I spoke about in some other sessions earlier ok. So, for example, if you look at the color as one example, we can say red is a color,

blue is a color and so on. So, these types of IS-A relationship is captured and the words are arranged accordingly. So, when you use an IS-A relationship the binary tree is unbalanced, it is not going to be a balanced binary tree as I have shown in the example here.

And then we can use the frequencies of the words in the corpus. So, for the most frequently occurring word we can use a shorter path and then the least occurring words we can probably have a longer path in the binary tree, Again if you go with this approach the tree is not going to be balanced, it is an unbalanced tree ok. So, there are ways of arranging the tree; let us assume that we have arranged it in some form and it is available for use for the computation, for the sake of this lecture ok.

As I mentioned earlier the path is very well known for any word and then at every step right from your start from the root node, so you want to make a left or a right. So, you make a binary decision at every node in order to reach the word. And then the length or the number of nodes that you have to cross is given by the relationship ok. So, this will give you the length of a balanced tree. So, as I mentioned earlier, the words can be arranged in any one of these ways, alright.

(Refer Slide Time: 07:55)

**BALANCED BINARY TREE WITH 8 WORDS**

$P(n, \text{left}) + P(n, \text{right}) = 1$

$P(w_i) = \prod_{j \in N_i} P(n_i(w, j))$ , where  $N_i$  is the list of nodes to reach the word and

$P(W) = \sum_{i=1}^V P(w_i) = 1$

	$w_1$	$w_2$	$w_3$	$w_4$	$w_5$	$w_6$	$w_7$	$w_8$
$P(w_i)$	0.0456	0.0657	0.2805	0.1382	0.0321	0.1371	0.2707	0.0301

Thus the hierarchical Softmax is a well defined multinomial distribution among all words

3 / 13  
NPTEL

So, now let us take an example with 8 words, we want to find out the probability for each word ok; which is normalized as we had seen in the Softmax. So, the goal is to really find out whether we are able to get to the level of Softmax without having that

complexity of Softmax, alright ok. The path is given here at the bottom. I am just going to make up some values initially and then see how this could be translated into the word 2 VEC model, alright. Let us first start with the root node 1 and then I am trying to find out the probability of the word  $W_3$  ok. So, how do we find out, let us start with this first ok? So, when I start from node 1. So, I need to go to node 2 I traverse to the left and then I traverse to the right from 2 to 5 and then to the left to reach the value ok.

So, let us first assume that the probability as 0.53 and then the rule for filling in the probability when you move from 1 to 3. So, it has to be balanced and the sum of the probability should be equal to 1. So, the value is going to be 0.47 here so; that means, the  $P_n$  left plus should be equal to 1. So, this is the rule that we have to apply let us assume that we are able to find the probability of reaching from node to node 2 to 5 as 0.79; then the probability of reaching from 2 to 4 would be 0.21 ok, alright. So, if you start doing all of this to the left node starting from the root and then to the right at the end we will arrive at some values for each of those.

So, if again the left and the right should add up to 1, in all those cases. When you want to find the value or the probability of this word  $W$ , what you do is you just multiply it multiply this one and this one ok. So, we are taking these 3 and we obtain the value which will be equal to the probability of the word  $W_3$ . So, we need to finally, figure out whether the probabilities of all these words really add up to 1. So, if it adds up to 1, then we have really achieved the goal of what Softmax has achieved. And it is very easy to prove that it really actually computes the values of all the words and then if you add each one of them, in the end, you will end up with 1.

So, in this case I have just listed in this table the values of  $W_1$   $W_2$  and  $W_8$  here. And you can actually sum all of these and then find out whether it really gets through the level ok. And really it gets to that level so; that means, it is practically now shown that this layer at the bottom contains the normalized probability of all the words in the vocabulary ok. So, this is the instead of the one depth Softmax layer, now we have a hierarchical Softmax layer and it is defined by the multinomial distribution among all words ok.

(Refer Slide Time: 12:07)

HIERARCHICAL SOFTMAX - ADVANTAGES

- ▶ Decomposes the flat hierarchy into a binary tree ✓
- ▶ The path to reach every leaf (word) is unique
- ▶ Lays the words in a tree-based hierarchy - words as leaves
- ▶ Binary tree with  $V-1$  nodes for left and right traversal
- ▶ Every intermediate node denotes the relative probabilities of its child nodes
- ▶ Every leaf represents the probability of the word ✓

Word Embedding using Hierarchies

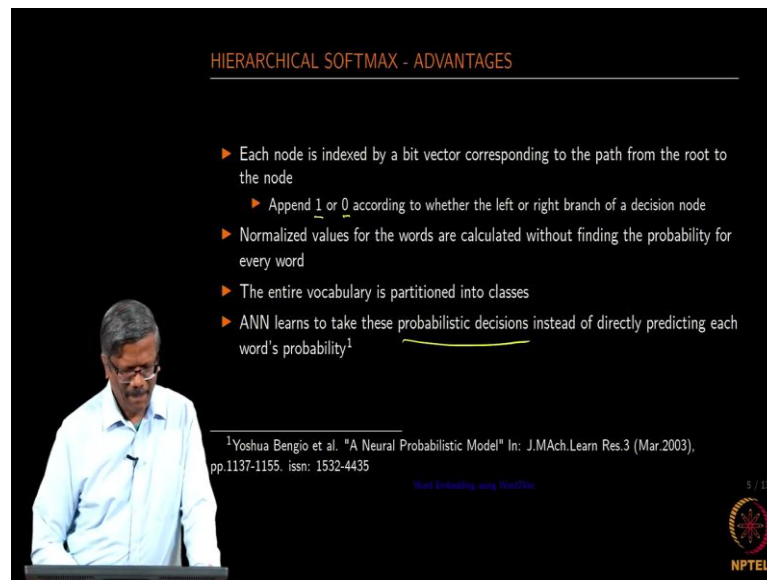
4 / 13

NPTEL

So, what are the advantages of this? So, why should we really care about the binary tree and the hierarchical model, and here are some of the advantages you will see. One is instead of keeping a flat hierarchy, we now have a binary tree with some depth right; the depth will be proportional to the size of the vocabulary that you have, and then the path to reach every word is unique. We have seen that in the previous case correct, so every word has a very specific path,.

So, as I mentioned earlier, it lays the words in the tree-based hierarchy words as leaves; and you will see that there will be  $V - 1$  nodes for the left and right traversing ok. So, if we have 8 words you will have about 7 nodes in the tree, every node denotes the relative probabilities of the child nodes. So, we have, if you go back here every probability defines the relative probability of the child nodes alright. And then as I mentioned earlier every leaf represents the probability of the word. So, that is a normalized probability let us use the right word.

(Refer Slide Time: 13:59)



**HIERARCHICAL SOFTMAX - ADVANTAGES**

- ▶ Each node is indexed by a bit vector corresponding to the path from the root to the node
  - ▶ Append 1 or 0 according to whether the left or right branch of a decision node
- ▶ Normalized values for the words are calculated without finding the probability for every word
- ▶ The entire vocabulary is partitioned into classes
- ▶ ANN learns to take these probabilistic decisions instead of directly predicting each word's probability<sup>1</sup>

<sup>1</sup>Yoshua Bengio et al. "A Neural Probabilistic Model" In: J.MAch.Learn Res.3 (Mar.2003), pp.1137-1155. issn: 1532-4435

Word Embedding using Hierarchies

9 / 13

NPTEL

So, again explaining further into the advantages of the hierarchical Softmax; each node is indexed by a bit vector corresponding to the path from the root to the node ok. So, here we append 0 or 1 or 0 according to whether we are taking the left path or the right branch when we make a decision at the node, right.

The normalized values are calculated without finding the probability of each word. So, instead of calculating the probability of each word using the Softmax, the normalized values are calculated without having to really compute that equation one that I had told earlier. It is done based on the product of all the probabilities that start from the root node to the leaf. And the entire vocabulary is partitioned into classes also, so if you look at this, so we can call this as the major class, right.

So, this is a binary and then when you go to the last node you will see the words in classes. So, this is what we are talking about when we mention IS-A relationship or frequency, you know we can keep all the words that are related by IS-A relationship as one group, ok. For example, if you look at the colors right, there could be some RGB values related to the colors wherein; the closer ones which we can mention for example, we are representing the colors in about 16 different ways, right.

So, the color 1 differs from the color 2 by a small fraction, color 3 and 4 they are close enough, but not very close to 1 and 2 or 5 and 6. So, in that way we can classify them and then arrange the tree in the classified fashion ok. Then what the network is going to

do is, it is going to learn the probabilistic decisions instead of directly predicting the probability. So, since we are going to be traversing from the root node to the leaf. So, instead of computing the probability using the Softmax, now it is going to be learning the decisions to move from one branch to the or one node to the other whether it has to travel to the left or right; and then what is the actual or the relative probability that we need to look at and then finally, take the probability of the word in question ok.

(Refer Slide Time: 17:13)

**HIERARCHICAL SOFTMAX - ADVANTAGES**

- ▶ Forms a hierarchical description of a word as a sequence of  $O(\log_2|V|)$  decisions ✓
- ▶ Reduces the computing complexity of Softmax -  $O(|V|) \rightarrow O(\log_2(|V|))$
- ▶ Path length of a balanced Tree is  $\log_2(|V|)$ . If the  $|V| = 1$  million words, then the path length = 19.9 bits/word
- ▶ A balanced binary tree should provide an exponential speed-up, on the order of  $\frac{|V|}{\log_2(|V|)}$  ✓
- ▶ Constructing an Huffman encoded-tree would help frequent words to have short unique binary codes
- ▶ H-Softmax in many cases increases the prediction speed by more than 50X times ✓

Word Embedding using Softmax

6 / 13

NPTEL

So, when we do this operation. So, instead of having an  $O(V)$  which is the Softmax complexity, we are going to be having a  $O(\log V)$  which is a lot smaller than what we achieved in the Softmax ok; from the Softmax of  $O(V)$  the order we are going to be having this order.

The path length is going to be 19.9 for a 1 million word vocabulary ok, and if you look at the order instead of 1 million where it is going to be 19.9 because we are going to be computing only one word at a time and not the entire vocabulary the probability of the entire vocabulary ok. So, that is where the computation complexity comes down drastically ok. The speedup is for a balanced tree, this is the speed up  $\frac{|V|}{\log |V|}$  and then we can use Huffman encoded-tree to help arrange the frequently occurring words in a place which is at a shorter distance.

So, when you do that the computation even reduces further downright, on average it will be equal to this; but when you arrange them according to the Huffman encoded tree it



will be a lot less than this as well ok. Because frequently occurring words will be kept at a place which is very close to the root and then the rest of the words should be you know placed according to this value ok. By using the Huffman encoding, we still get a lot more speed, you know that is in some cases it is shown it is about 50 X faster.