A Quick Summary: Hierarchical Probabilistic Neural Network Language Model

Original Paper:
https://www.iro.umontreal.ca/ lisa/pointeurs/hierarchical-nnlm-aistats05.pdf

21st Feb 2019

1 Ideas:

(a) Traditionally, when a language model outputs a vector $v \in \mathbb{R}^V$, where V is the size of our vocabulary, we want to calculate the softmax (which gives us the probability) of this, which is given by:

$$softmax(v)_i = \frac{exp(v_i)}{\sum_{j=1}^{V} exp(v_j)}$$

However, doing so requires $O(V)$ computations, which can be forbidding if our vocabulary is very large. So, instead of doing that, the authors came up with an extreme method to do it in $O(log_2(V))$ computations instead.

This proposed model can work during training, where we are given the next word. However testing requires us to use the traditional softmax still.

2 Explanations:

(a) The method that was presented in this paper involved forming a tree of depth $log_2(V)$, such that the vocabulary is found on all the leaves. Each node is associated with a vector $w_{j,k} \in \mathbb{R}^n$, where j indexes the level (vertical), and k indexes the horizontal direction.

I will not use the actual formula to calculate the probability of each node as presented in the paper, but provide a brief intuition of what goes on under the hood. Basically, after the RNN gives an output word representation $y \in \mathbb{R}^n$, we can calculate the probabilities of all our nodes by taking $\sigma(y^T w_{j,k})$. (The actual formula for this probability presented in the paper is a little more sophisticated as it takes into account the outputs of previous words as well, but the idea is similar. [For reference, the actual equation in the paper is under section 4]). However, we don't need to do this for **all** our nodes! We only need to calculate these probabilities for the nodes that our path (which we know) passes through.

Perhaps it is necessary to state that the structure of the tree was created prior to training, so positions of the words on the leaves are fixed. The position of each word can then be specified by a vector $r \in \mathbb{R}^{ceiling[log_2(V)]}$ of 0's and 1's. For example, if $r_1 = 0$ and $r_2 = 1$, we first go down the left path and then the right path.

i During **training**, one can simply calculate the probability of the word that actually appears, since this is known to us. We simply traverse down the tree to the leaves, multiplying the accumulated probability with the next probability associated with the next node in the path. Using the final probability calculated, we can implement backpropogation after calculating the loss.

ii During actual **prediction**, in order to find the word with the maximum probability, we will still have to do traditional softmax which requires $O(V)$ computations.

iii However if we just want to **evaluate** the generalization error, we can still use the same scheme as we did during training.

3 Model:

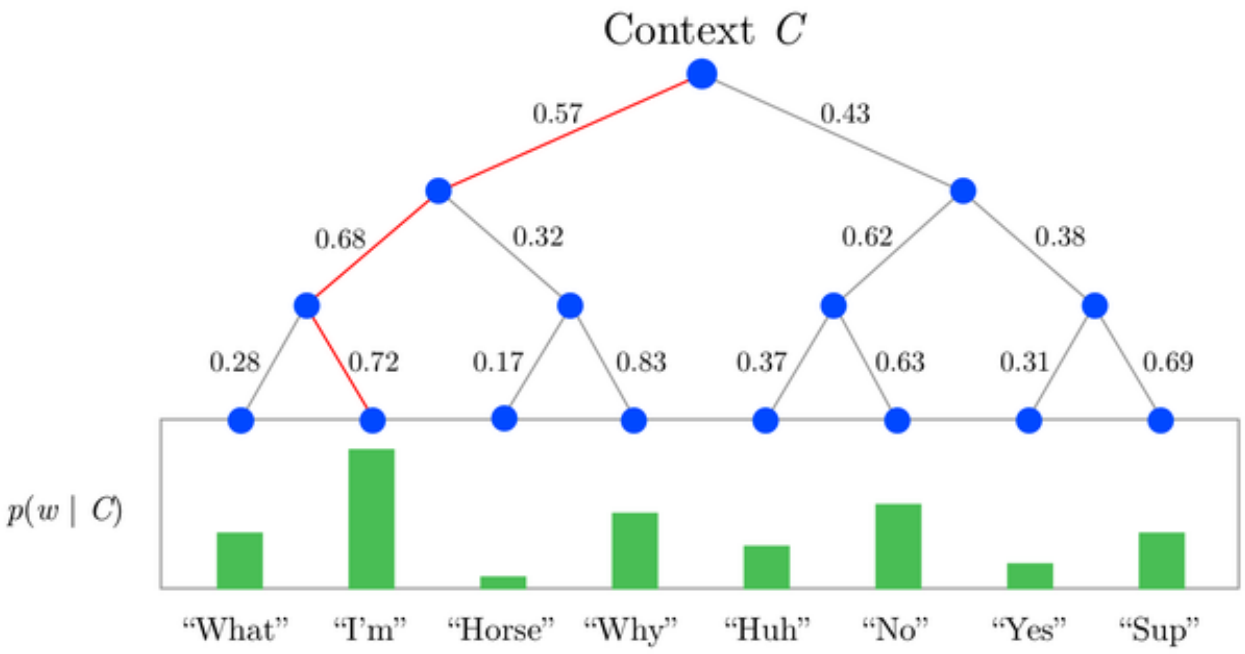

Figure 1: Illustration of the model (source: https://www.quora.com/What-is-hierarchical-softmax)

4 Results:

(a) Speedup of training/evaluation by 200 times compared to original softmax.

(b) Perplexity higher than original softmax

5 Notes:

(a) What if we took a greedy approach to the actual prediction instead? (i.e. select the highest probability at every level).

(b) Structure of words is fixed - but what about words with double meanings?