Dependency Parsing exercises: Graph-based parsing I

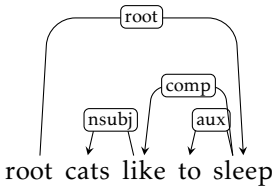
Solutions

1. We would like to parse the sentence "Cats like to sleep". To do so, we will use the weight function w, defined as follows:

$$
\begin{aligned}
w(root, cats) &= 1 \\
w(root, like) &= 2 \\
w(root, to) &= 0 \\
w(root, sleep) &= 3 \\
\\
w(cats, like) &= 0 \\
w(cats, to) &= 2 \\
w(cats, sleep) &= 0 \\
w(cats, root) &= 0 \\
\\
w(like, cats) &= 4 \\
w(like, to) &= 2 \\
w(like, sleep) &= 5 \\
w(like, root) &= 0
\end{aligned}
\qquad
\begin{aligned}
w(to, cats) &= 2 \\
w(to, like) &= 0 \\
w(to, sleep) &= 0 \\
w(to, root) &= 0 \\
\\
w(sleep, cats) &= 3 \\
w(sleep, like) &= 3 \\
w(sleep, to) &= 3 \\
w(sleep, root) &= 0
\end{aligned}
$$

   where $w(i, j)$ is the weight of the arc from node i to node j. Here, we ignore the labels of the arcs (only the weight of the best label is given).

   (a) According to the weights for the arcs starting from root, it would be easy to think that "sleep" should be the head of the sentence. This would lead to this parse:



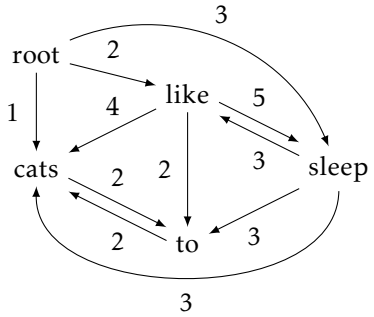
   What is the weight of this parse? Is this the best parse that can be obtained with "sleep" as head of the sentence?

   Solution: $w(root, sleep) \cdot w(sleep, to) \cdot w(sleep, like) \cdot w(sleep, cats) = 3 \cdot 3 \cdot 3 \cdot 4 = 108$. (We assume that the weights are already in log scale, otherwise we have to multiply them which is also fine.) This is the best parse that can be obtained with "sleep" as the child of "root". There must be exactly 3 other edges for the graph to be a tree. These edges cannot point to "sleep", as "root" is already the head of "sleep". Among all possible edges that satisfy this criterion, the ones that are shown have the highest weights (3, 3, and 4). Removing any of them and adding another would necessarily lower the score, with the possible exception of $(sleep, cats)$

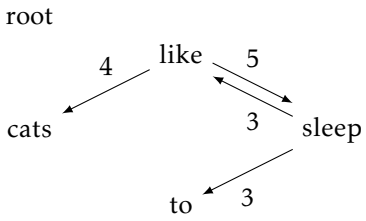
which also has weight 3. However, to add this we would have to remove (*like*,*cats*), which has weight 4, so the score would still be lower. □

(b) Using Chu-Liu-Edmonds' algorithm, determine the best parse of the sentence with the given weights. Please show all the indermediate graphs used during the parsing process. Compare the weight of the resulting analysis with the one of the previous question.

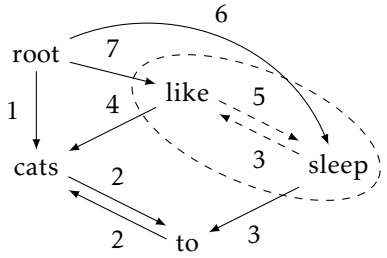
Solution: Since our weights are all above 1, we can skip converting them to log scale. We first draw the complete digraph of the edge weights (omitting 0 edges):



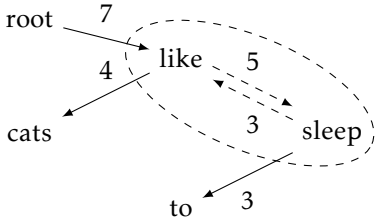
Now, we keep only the maximum incoming edge for each node:



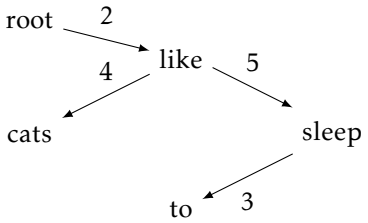
As we can see, there is a cycle between "like" and "sleep". So we create a version of the original graph where they are contracted into one node. We have outgoing arcs to "cats", with weight 4 from "like" and weight 3 from "sleep", so we keep the former. We have outgoing arcs to "to" with weight 2 from "like" and weight 3 from "sleep", so we keep the latter. The highest incoming arc weight to "like" is 2 (coming from "root"), and we have to add 5 to complete the cycle, giving 7. The highest incoming arc weight to "sleep" is 3 (coming from "root"), and we have to add 3 to complete the cycle, giving 6. We still draw the edges within the cycle so we can remember them later, but we don't consider them part of the graph now.



We now recursively start again, keeping only the maximum incoming edge for each node:



This is a tree! We now uncontract the cycle, restoring the original edge weights. Since we have an incoming edge to "like", we have to drop the edge that points to "like" within the cycle. We also have to restore the original edge weights.



And this is our maximum spanning tree. Its weight is $w(root, like) \cdot w(like, cats) \cdot w(like, sleep) \cdot w(sleep, to) = 2 \cdot 4 \cdot 5 \cdot 3 = 120$, higher than the tree where "root" pointed to "sleep", although that edge had a higher weight than the one from "root" to "like". This illustrated that you can't find the MST by greedily following the highest-weighted edge.

(c) Optional task for extra credit: Consider a word w that was never found in the corpus which was used to learn the weights. What are the problems encountered by the algorithm when w occurs in a sentence to be parsed? Think also about the labels of the dependencies (which we can usually ignore). Which solution(s) can you propose to solve these problems?

Solution: If we treat the weight of edges involving words that were not observed in the corpus as 0, as we naively might, then the weight of every possible tree will be 0, making it impossible to compare them. We can't even really apply the algorithm, as the logarithm of 0 is undefined. One solution is to introduce default weights for unknown words. They could, for instance, be estimated by treating each word that occurs only once in the training corpus as unknown.

If we don't ignore dependency labels, instead of weights like $w(like, cats)$, we have separate weights $w(like, cats, nsubj)$, $w(like, cats, dobj)$, etc. We are unlikely to observe all possible combinations. However, if we make sure that we have at least one nonzero edge for each pair of words, we should be fine as we can then ignore all other edges for this pair.