NLP Homework 4

Quanjun Lang, Mou Zhang

October 2019

1 Question 1

1 Here are the four papers that I found:

- Knowledge-Based Question Answering as Machine Translation
- Cross-language Projection of Dependency Trees with Constrained Partial Parsing for Tree-to-Tree Machine Translation
- Efficient Top-Down BTG Parsing for Machine Translation Preordering
- Using Discourse Structure Improves Machine Translation Evaluation
- Summing up all the articles above, we find out that using parsing to get the structure of the a sentence is the preliminary step to get the meaning of the sentence, and after understanding the structure and meaning of the sentence, the machine translation or the answer system starts to work.

2 Question 2

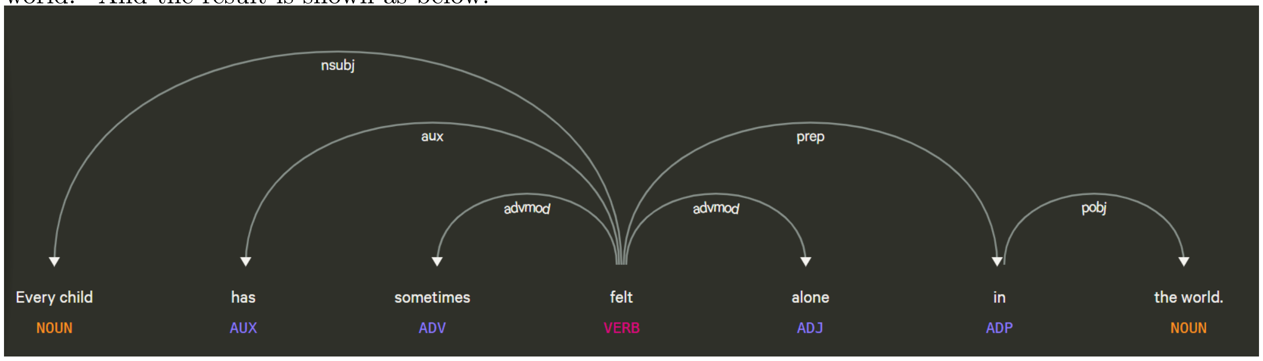
2

- (Interesting or surprising) I tried Berkeley Parser, for "Every child has sometimes felt alone in the world." And I noticed that the parsing tree is not a binary tree.
- (Wrong)
- (Confuse) I tried "Why did Bob eat an apple on the tree?" I meant to say the apple is on the tree, but the parser thinks that Bob was eatting on the tree.

3 Question 3

3

- When I tried the dependency parser, I used the sentence "Every child has sometimes felt alone in the world." And the result is shown as below:



- Dependency parser analyzes the grammatical structure of a sentence, establishing relationships between "head" words and words which modify those heads. The arrow from the word *felt* to the word *alone* indicates that *alone* modifies *felt*, and the label *advmod* assigned to the arrow describes the exact nature of the dependency.

- Dependency is a one-to-one correspondence: for every element (e.g. word or morph) in the sentence, there is exactly one node in the structure of that sentence that corresponds to that element. It is more efficiency and uses less space. In contrast, the constituency tree follows the conventions of bare phrase structure, whereby the words themselves are employed as the node labels. It is inefficient and uses more space.

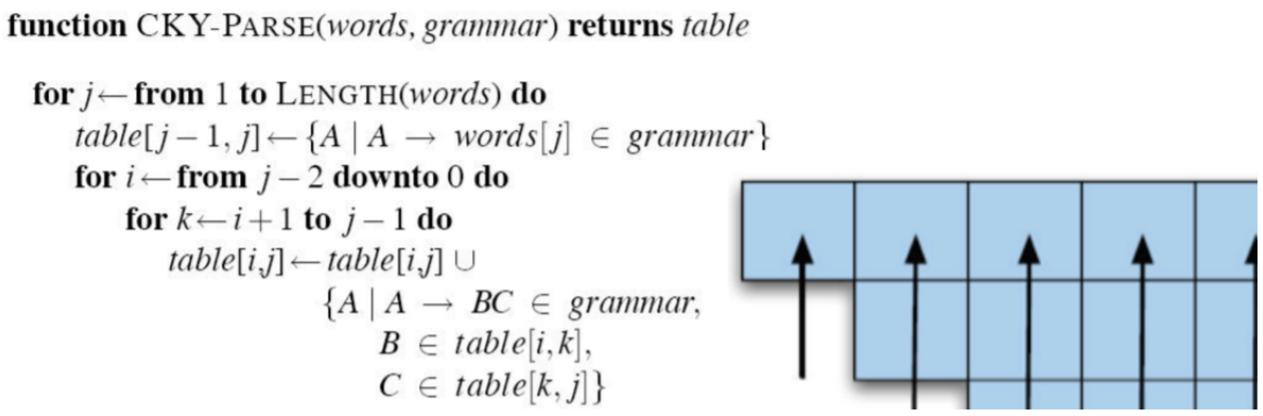
4 Question 4

4

1. In python, it's convenient to use list.append() to add elements to a list.
2. We use binary tree nodes to store those information. Each node has attribute weight name, left child and right child. The $-log_2 P$ is stored in the weight, the rule that has been used is stored in the name, and the right hand side of the rule corresponds to the two sons of the node.
3. We keep the possible parsing in a list of binary tree nodes. So that we can only use $O(n^2)$ space.

5 Question 5

6



```
function CKY-PARSE(words, grammar) returns table

  for j ← from 1 to LENGTH(words) do
      table[j−1, j] ← {A | A → words[j] ∈ grammar}
      for i ← from j−2 downto 0 do
          for k ← i+1 to j−1 do
              table[i,j] ← table[i,j] ∪
                          {A | A → BC ∈ grammar,
                               B ∈ table[i,k],
                               C ∈ table[k,j]}
```

The pseudo-code is shown above. The first for j represents the end of the current parsing string, and it's complexity is O(n). In CKY algorithm, each time we calculate the parsing in a block, we need an end point.The second for i represents the beginning of the current parsing string. its complexity is O(n). In CKY algorithm, each time we calculate the parsing in a block, we need a begin point.And the third for k means the midpoint of the parsing, which is also O(n). We need a parsing midpoint to find all possible parsing. Finally, in for k, it calculates every grammar in the grammar file A ->BC, whose complexity is O(|G|). It is because looking through every grammar is the way to find all grammars that fit the parsing. In summary, the complexity of the whole program is O(n * n * n * |G|) = O(n$^3|G|$)

7

- According to EFFICIENT IMPLEMENTATION OF THE CKY ALGORITHM, we got 1 way to improve the CKY algorithm.

- This is to improve the grammar loop process. Instead of finding every node as the midpoint, we keep a left-max node(the leftmost position midpoint can reach) and a right-max node(the rightmost place midpoint can reach) for every node in the CKY algorithm. Therefore we only need to find the midpoint between left-max node and right-max node. And after each step of finding midpoint, we update the leftmost point and the rightmost point to keep track of the show period of the midpoint. This will increase the performance a lot.