Parsing with CKY algorithm

Ziqian Luo
Carnegie Mellon University
ziqianlu@andrew.cmu.edu

Abstract

The project implements a generative PCFG parser. In order to build the parser, I firstly annotate the raw tree where a node may have more than two children. Then I implemented my parent annotation and Markovization to build a tree in PCFG form. When the intermediate node can remember two of its upper parents and two of its left neighbors, in this project, to say v=2 and h=2, has the best performance. In default setting, where maxTrainLength =1000 and maxTestLength=40, the final F1 score result is 80.02 and the decoding time varies from 790 seconds to 980 seconds on my MacbookPro (2.2 GHz Intel Core i7). The decoding time varies a lot and that interval is summarized by lots of experiments I did. There might be a little or huge difference on Test Machine but those results are really true which have screenshots. You can also use my Mac to run if there is a huge mismatch. There will be more discussion about the decoding time in Part 2. The project did not implement coarse-to-fine.

1 Implementation details

CKY is a parsing algorithm for context-free grammars which employs bottom-up parsing. The standard version of CYK operates only on context-free grammars given in Chomsky normal form (CNF). However any context-free grammar may be transformed to a CNF grammar expressing the same language (Sipser, 1997).

1.1 Binarize and Annotation

The first step is to binarize tree. Take v=2, and h =2 as an example. My policy is to binarize the tree when there is more than two children. The original code given will binarize a node when there is two children. This action will include a somewhat useless intermediate label. So I designed a method, binarizeTreeHelperForTwo(...), to deal with the two children situation. I am sure that it improves F1 score about 0.8 and maybe will also improve the decoding time since there will be fewer grammar rules. In fact the decoding time varies from every experiment so I cannot say which one is absolutely faster than the other one. But the average time is shortened about 100 second in decoding time.

As for the annotation, I use the following format of annotation shown in Figure 1:

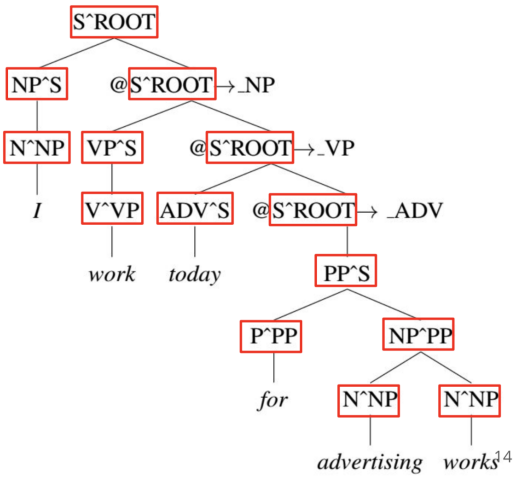

Figure 1: Annotation

It is the situation where v=2 and h=1. For intermediate node, it will remember its left one neighbor and its parent and grandparent node. For non-intermediate node, it will not remember its neighbor and remember v-1, that is one in this case, parent node. By the way, as for the unannotating the tree, there exists a method and I just call that like Baseline model.

1.2 Handel Binary Rules

As recommended in the recitation, there are two shape identical charts to deal with that. There is one chart to deal with binary rules and another

chart to deal with unary rules.

Each chart is three dimensional there the first two records row and column to locate and the last index used to store score.

When I am dealing with binary rules, I fill the binary charts and fill unary charts when dealing with unary rules. So, there two two chart are filled alternatively.

I used the filling policy TA introduced in recitation and it really increase the decoding a lot.That is, when we are finding a binary rule, we only need to check those binary rules that have left child C1, so I do not have to look for all those grammar rules. The algorithm is shown on Figure 2:

```
<fill in possible pre-terminals and unary closures of those>
for each max from 2 to n
  for each min from max-2 to 0
    for each mid from min+1 to max-1
      for each non-terminal C1 present at [min][mid]
        for each binary rule C -> C1 C2
          if unary_chart[min][mid][C1] and
             unary_chart[mid][max][C2] then
          binary_chart[min][max][C] = score(min, mid, max, C, C1, C2)
```

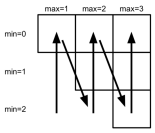

Figure 2: Chart

1.3 Backtrack

I used the a recursive policy to do the backtrack. Firstly, in order to do the backtrack, I should store the children positions of a node in both unary and binary situation. So I used another four identical three dimensional charts to store those positions. The backtrack is an inverse process to annotation while I trace back from up to the bottom.

2 Performance

After a lot of experiments, I found v=2 and h=2 has the best performance and in this situation, it has 8848 states in default setting with maxTrainLength = 1000 and maxTestLength = 40 . The F1 Score is 80.02 and the fastest decoding time I recorded is 790.085 seconds on my Mac and the worse is about 990s. There might be some or huge difference in other testing machine so if you use a machine that has processor with 2.2 GHz Intel Core i7 and memory with 16 GB 2400 MHz DDR4 and similar results will be got if run several times. There are some screen shots to show that:

3 Experiments

Tree annotation will influence the decoding time and F1 score. The best result is v=2 and h=2, judged by both F1 score and decoding time. If

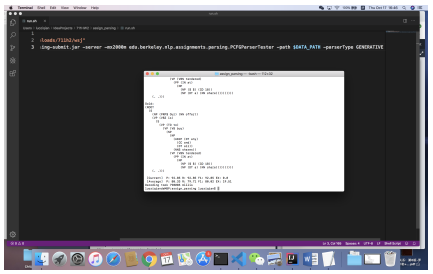

Figure 3: shot1 with 790 seconds

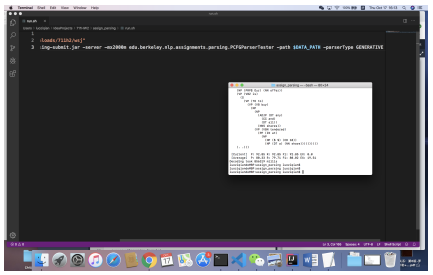

Figure 4: shot2 with 856 seconds

Figure 5: shot3 with 940 seconds

Annotation	F1	Decoding Time
v=2 h=3	**80.04**	**1580s**
v=2 h=2	**80.02**	**790s**
v=2 h=1	71.01	273s
v=2 h=0	66.51	85s
v=1 h=inf	72.73	1360s
v=1 h=3	72.53	726s
v=1 h=2	72.64	348s
v=1 h=1	61.94	101s

Table 1: Comparison With Two Models

we only consider the best F1 score, v=2 and h=3 might be a better choice. The result is shown in Table 1 and best F1 score is shown in Figure 6.

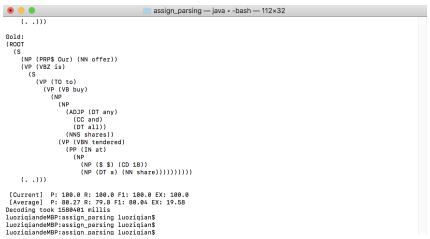

Figure 6: shot4 with highest F1

When v=2 and h is INF, there are about 14000 states and took a long time to decode. Actually, if v and h are both too large, it will launch OOM error.

If we only look at v we can find that the increase in v will improve F1 score a lot and it can exceed 80 when v is just 2. Increasing h will also improve the final F1 score but when it goes to INF, the result does not get much higher, so we can say h does not improve F1 a lot when it grows bigger. It's somewhat obvious that increase v will have a bigger impact towards F1 score compared with increasing h. Since the F1 bar is only 79 so I stopped my experiment at v=2 and h=2, a better F1 score might exist but also need to take a longer decoding time as there are more states. So v=2 and h=2 may be a good trade off point in this project.

4 Discussion

4.1 Future Work

I will increase both v and h under memory limit to see what is the best v and h that will give the highest score, in the condition we really do not care about decoding time. I will also do the coarse to fine in the future.

4.2 Error Analysis

I have seen some sentence that got zero or very low F1 simply due to they predict wrongly at the top of tree so the children are all wrong. These are flaws of evaluation of the results. There might exist some potential problems in the training data that will be improved by coarse to fine method.

References

Michael Sipser. 1997. Introduction to the theory of computation. *PWS Pub*, 8(1):381–385.