CS224N Tree Parsing

Jiayuan Ma
jiayuanm@stanford.edu

Xincheng Zhang
xinchen2@stanford.edu

October 23, 2013

1 Implementation Detail

1.1 CKY parsing

Following the pseudo code given in the lecture notes, we implemented the CKY parsing algorithm. Instead of allocating a single large three dimensional array for dynamic programming, we store our dynamic programming results and backtracking information as two single dimensional arrays. In addition, we use a two dimensional array `scoreIdx` to record the mapping from `(begin, end)` pair to the index `scoreIdx(begin, end)` of these single dimensional flat arrays.

The following are the definitions of data structures used in our implementation of CKY algorithm. As explained before, `scoreTable` and `backTable` are two single dimensional arrays for dynamic programming and backtracking. `scoreIdx` is used for translating two dimensional `(begin, end)` pairs into the singleton indices of `scoreTable` and `backTable`.

```
// Single dimensional array for the score
private List<Map<String, Double>> scoreTable;
// Single dimensional array for the backtrack information
private List<Map<String, Triplet<Integer, String, String>>> backTable;
// Two dimensional array for translating pairs into indices
private int [][] scoreIdx
```

During dynamic programming, it is observed that we do NOT need to delete any elements from arrays. Therefore, we chose to use `ArrayList` and `HashMap` in our implementation.

- `ArrayList` supports $O(1)$ random element accesses and amortized $O(1)$ element insertions. Since we know in advance there will be exactly $(N+1)N/2$ elements in the array, where $N = 1 + \#$of words in the sentence. We can allocate $(N+1)N/2$ slots ahead of time to avoid copying and reallocation.
- `HashMap` stores Nonterminal $\Longrightarrow$ probability mappings for a specific word span of `(begin, end)`. Given a decent hash function, `HashMap` supports $O(1)$ element accesses and insertions, which is good enough in our application.

On the other hand, we use `backTable` to store backtrack information needed to construct parse tree after dynamic programming. For each entry of `HashMap` in `backTable`, we store a mapping from nonterminals to triplets, where each triplets contain splitting point, left subtree labels, and right subtree labels. For unary rules, we marked the splitting point of the triplet as -1 to indicate that we have an unary rule here. During backtracking, if we want to access additional information with respect to a particular `(begin, end)` pair, we simply retrieve the `HashMap` by

Finally, the translation mapping `scoreIdx` is built on-the-fly. When we are filling the "grid" in CKY algorithm, we simply append the score and backtrack information to the corresponding array, and record their locations in `scoreIdx` for future look-ups.

```
scoreIdx[begin][end] = scoreTable.size();
scoreTable.add(score);
backTable.add(back);
```

1.2 Tree Annotation

We add a vertical markovization helper method, which take order as a parameter. It can decide how many level of parents nodes needed to be in the appending label of a tree node. For horizontal markovization, we simply add a forgetting procedure inside the binarization method provided. When we found the label has too many previous state, we will forgot them and leave the most recent labels only.

2 Performance Evaluation

of Standard CKY Parsing We ran a plain CKY parsing and a CKY parsing with 2nd order vertical for result comparison. Result: [Average] P: 81.31 R: 75.58 F1: 78.34 EX: 20.65 The subset of grammar being parsed well: The most simple case, which a sentence only contains one VP and verbs with variation does not appear a lot in the sentence seems to be parsed nearly perfect. As the miniTest example. Those tree tend to have small level. Ex 1:
(ROOT (S (PP (IN At) (NP (NP (DT the) (NN end)) (PP (IN of) (NP (DT the) (NN day))))) (, ,) (NP (QP (CD 251.2) (CD million)) (NNS shares)) (VP (VBD were) (VP (VBN traded))) (. .)))
To mark the verb to correct tag is a huge challenge. Since verb may have different tense and some time verb does not necessary means a VP appear, it may just be serving as a ADJP/ADVP purpose, while sometimes vice versa. The simple learning algorithm did not deal with this situation well. Ex 2: (ROOT (S (NP (DT The) (NN finger-pointing)) (VP (VBZ has) (ADVP (RB already)) (VP (VBN begun))) (. .))) Plain Guess: (ROOT (S (NP (DT The) (NN finger-pointing)) (VP (VBZ has) (ADJP (RB already) (VBN begun))) (. .))) In the plain mode, since @VP-
¿$_V BZ is dominated by ADJP, as verb must be followed by adjective. So the already begun is not being tagged correctly. (VP(VB$
$BZ(ADJP(RBalready)(@ADJP- >_R B(VBNbegun)))))$
Verb Challenge is actually a special case of the deficiency in the plain CKY parsing, when a tag is dominated by a special rule. Like we normally will assume tight is an adjective, but when it is following after a adverb and under a VP, it should be an adverb. The parser will ignore other parsing path and lead to a not desired result. In summary, it is hard for parser to learn multiple rule for one semantic type. Since in our setting, we simply compare the probability score of one particular semantic type.
Another problem is sometime the highest score parsing may not lead to a sentence (label S) ex: Guess: (ROOT (PRN (-LRB- -LRB-) (S (NP (NNP See)) (VP (VBD related) (NP (NP (NN

story)) (: :) (“ “) (NP (NP (NNP Fed) (NNP Ready)) (TO to) (NP (NNP Inject) (NNP Big) (NNPS Funds))) (” ”) (: –) (NP (NNP WSJ) (NNP Oct.) (CD 16) (, ,) (CD 1989))))) (-RRB- -RRB-)))
The precision and recall is 0 in this particular case.
2nd order vertical markovization may be a fixed for the dominant issue. By adding parent node tag into label, it actually split the tag to several. And we can do the counting under each of them. which gives the parsing more possible rules to choose from. The result: [Average] P: 83.71 R: 81.33 F1: 82.50 EX: 32.26
With vertical markovization, it guess Ex2 100(VP$^S(VBZ^VPhas)(@VP^S->_V BZ^VP(ADVP^VP(RB^ADVPalready))(@$
$BZ^VP_ADVP^VP(VP^VP(VBN^VPbegun)))))$
it can learn ADVP$^VPcombineswithanotherVPisactuallyappearalotunderVP->VBZ^VPspecificcategory,sobegunca$
But it may still not be able to split the tag enough for correct grammar to be learned. And larger grammar space may lead us learn some not valuable and misleading rule, for Ex 3:
Plain Guess: (ROOT (S (NP (DT The) (NNS futures) (NN halt)) (VP (VBD was) (RB even) (VP (VBD assailed) (PP (IN by) (NP (NNP Big) (NNP Board) (NN floor) (NNS traders))))) (. .)))
Parent annotated guess: (NP (DT The) (NNS futures) (NN halt)) (VP (VBD was) (SBAR (RB even) (S (VP (VBD assailed) (PP (IN by) (NP (NNP Big) (NNP Board) (NN floor) (NNS traders))))))) (. .)))
Gold: (ROOT (S (NP (DT The) (NNS futures) (NN halt)) (VP (VBD was) (ADVP (RB even)) (VP (VBN assailed) (PP (IN by) (NP (NNP Big) (NNP Board) (NN floor) (NNS traders))))) (. .))) Here we classify the even assailed by as a cause when using parent annotation. It is because it see more VP$^SthenasimpleVP.Thisshouldbesolvedbythereferencepapersuggest,shouldntparentannotateallthenode.Somehuman$
$definedruleshouldbeusedinordertopreventsplitthegrammarovergreedy.$
Possible Improvement The problem of split tag is it may create too many rules and make the probability score of rule not reliable since it might not be able to aggregate enough samples for one particular rule. And some rule are not useful so we should not populate it at all. So we also implement the horizontal markovization to merge tags by forgetting the far away states.
Implemented Extension and Analysis As describe above, we implement horizontal markovization by modify the binarization to take a order parameter for forgetting states. By doing this we try to merge states together to avoid create a over-large grammar space. Result: [Average] P: 84.59 R: 82.63 F1: 83.60 EX: 33.55
The improvement does not seems to be huge. FIXME
We also tune the vertical markovization order to be 3 and test again. Result: [Average] P: 84.02 R: 84.47 F1: 84.24 EX: 37.42 This is the final version we submitted for machine grading.
We also try some other heuristic. For horizontal markovization, instead of only remember