PCFGs AND PARSING

Qing Wei
Department of Computer Science and Engineering
University of California San Diego
La Jolla, CA 92093
qiwei@eng.ucsd.edu

September 22, 2020

1 Implementation

1.1 Preprocessing

1.1.1 Horizontal Markovization

In order to make the raw trees appropriate for learning a PCFG form, the first preprocessing goal is to binarize the original tree. An example of the raw tree is shown in Fig. 1. In order to binarize it, we could introduce intermediate nodes. For any node in the tree that has more than two children nodes, the new node will point to its leftmost child and an intermediate node, which will act as the new parent of the node's original children besides the leftmost one.

The problem remains: what could be the label of the intermediate nodes that could mostly keep the original information in the raw trees? One intuition is to use the label to keep track of all its left siblings (as shown in Fig. 2 (a)). However, this setting may result in the labels of intermediate nodes being too complicated and overfitting. Therefore, we could use Markov assumption that the current node is only dependent with a few of its preceding nodes in the same level. Fig. 2(b) illustrated the 1st-order horizontal markovization of the tree in Fig. 1. It shows that markovization reduces the information in some of the intermediate nodes in Fig. 2 (a). In Section 2, we will compare the effectiveness of horizontal markovization with different order (h value).

1.1.2 Parent Annotator

Experiments show that the probability distribution of tags may be dependent on their parent tags. Based on this knowledge, we could apply parent ($v = 1$) or grandparent ($v = 2$) annotation on the parsing tree. On every node in the transformed tree, we add information of its v-order ancestors on its label, as shown in Fig. 3.

1.1.3 Debinarization

The intermediate nodes are clearly marked with '@' symbol, and the parent/grandparent notations occur only after a special symbol '∧'. Therefore, it is easy to remove all intermediate nodes, and erase the annotation on the trees.

1.2 Training

1.2.1 Grammar

The grammar (rules) is trained on the previously annotated binary tree. Therefore, only binary and unary rules will be produced, largely simplifying the process. The core part of grammar train is to get the scores of rules.

For unary rules $X \to Y$ or binary rules $X \to YZ$, the score (probability) is:

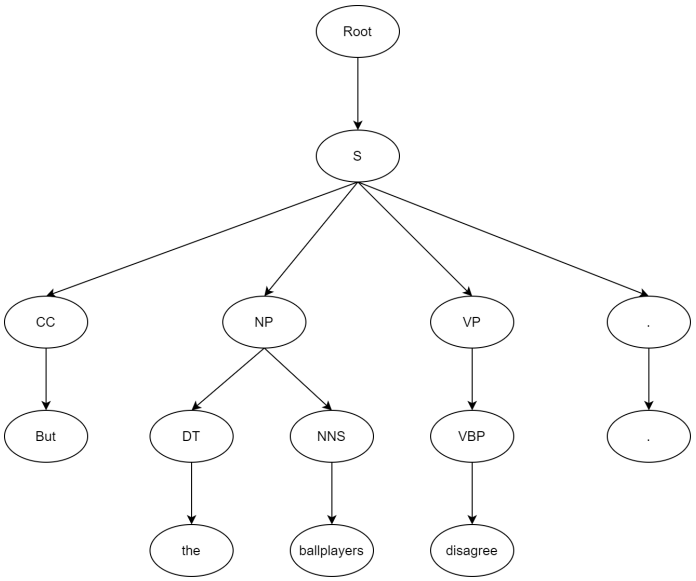
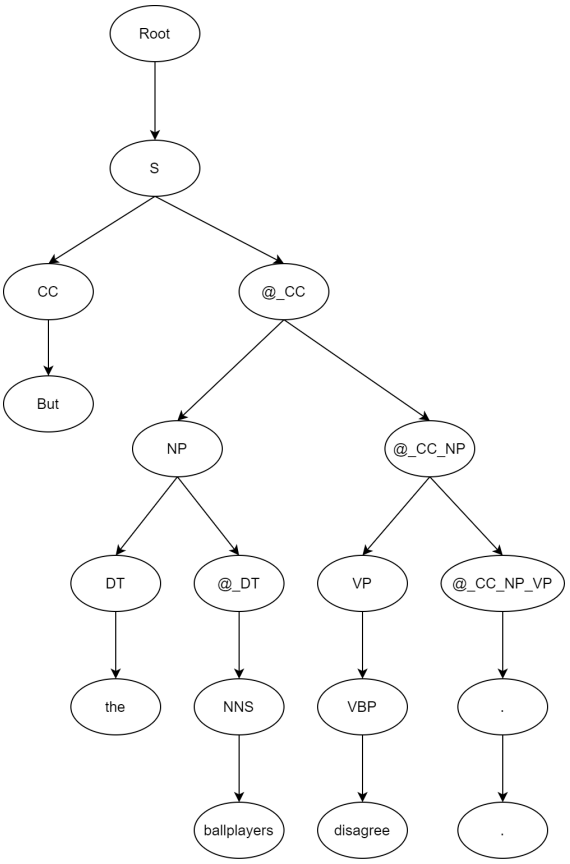
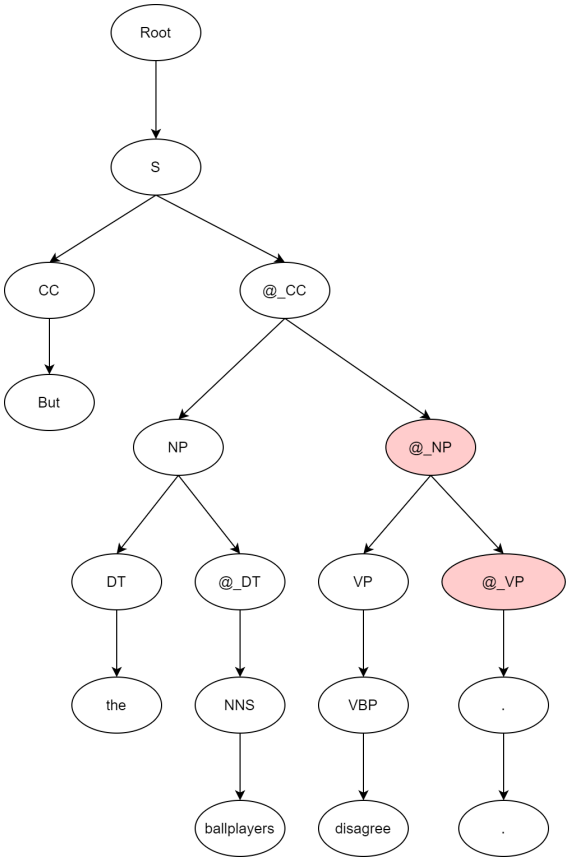


Figure 1: Illustration of a raw tree from the training dataset.



(a) Binarized tree with intermediate nodes track all its left siblings. The nodes whose labels start with '@' is the intermediate nodes.



(b) Binarized tree with Horizontal Markovization. The red nodes highlight the difference between Fig. 2(a).

Figure 2: Illustration of the binarizaed form of the raw tree in Fig. 1

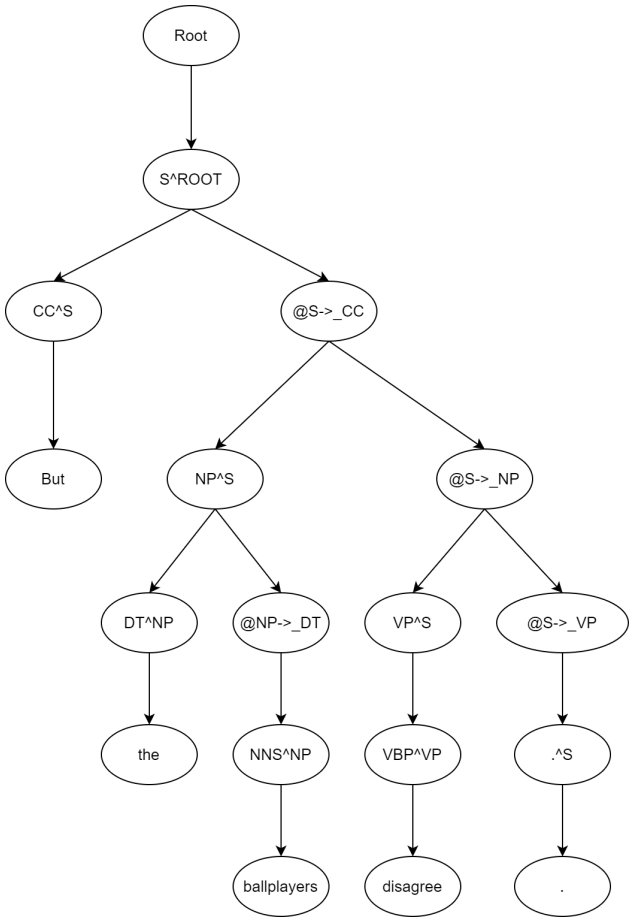


Figure 3: Illustration of the vertical annotated tree of the raw tree in Fig. 1. In any intermediate node (whose label starts with '@'), the notation before '->' is its parental tag. For other nonterminals, their parent tags are marked after the '∧' symbol.

$$P(X \to Y) = \frac{\#\ \text{COUNT}(X->Y)}{\#\ \text{COUNT}(X)}$$
$$P(X \to YZ) = \frac{\#\ \text{COUNT}(X->YZ)}{\#\ \text{COUNT}(X)}$$

1.2.2 Unary Closure

Unary closure is an important method applied to avoid recursive unary rules, which happens when $X \to Y$ is followed by $Y \to X$. In the parsing phase, we divide the scores to two layers, bottom and top, represent the next rule applied is unary and binary, respectively. Since we apply binary and unary rules alternately, it is important that we introduce identity unary rules (e.g. $X \to X$) and combine rules like $X \to Y$ and $Y \to Z$ to $X \to Z$.

In this case, if in the optimal parsing tree, we apply two binary rules consecutively, then the unary rule that is applied between is an identity rule; if in the optimal parsing tree, several unary rules are applied consecutively, then they are combined to a single closed unary rule. Thus, we will still get the optimal tree even when alternating binary and unary rules.

When combining rules $X \to Y$ with probability p_1 and $Y \to Z$ with probability p_2, unary closure will additionally produce a rule $X \to Z$ with a probability $\max(p_3, p_1 \times p_2)$. p_3 is the original probability of rule $X \to Z$ if it also occurs in the training set, or p_3 will be set to be negative infinity.

When combining these rules, it is important to keep track of the hidden path for the unwrapping process we will discuss later in Section 1.3.2.

1.2.3 Lexicon

Lexicon records the probability of a word that tagged with a nonterminal symbol. The score of a (tag, word) combination is computed as below:

$$P(\text{tag} = t) = \frac{\#\text{ of } t}{\#\text{ of total tokens}}$$
$$P(\text{word} = w) = \frac{\#\text{ of } w + 1.0}{\#\text{ of total tokens} + 1.0}$$
$$P(\text{tag} = t|\text{word} = w) = \frac{\#\text{ of } (w, t)}{\#\text{ of } w}$$
$$\text{score}(\text{tag} = t, \text{word} = w) = \frac{P(\text{tag} = t|\text{word} = w)}{P(\text{tag} = t)P(\text{word} = w)}$$

1.3 CKY Parser

Implement CKY to compute Viterbi trees

The above training process produces several important parameters for the parsing part: grammar (includes all nonterminals, scores of all unary rules and binary rules from the training set), lexicon (includes the tagging score of a nonterminal tag and a word).

In the parsing part, given a sentence, the parser will use the above information to construct score tables and with that table, to get the most likely parsing tree for the sentence.

1.3.1 Score Table Construction

To avoid the occurence of recursive unary rules like $X \to Y \to X$, we alternate the binary and unary rules. After a closed unary rule is applied, the next rules applied on the child node must be a binary rule. To optimize for the process, the score table is divided to a bottom table (the last rule applied is binary), and a top table (the last rule applied is unary).

Given sentence, the algorithm to compute the score tables are described in Algorithm 1. Note that in this algorithm, all the scores are set to logarithm of probability, thus we use plus instead of multiplication in the computing equations.

'bottomTable' and 'topTable' are stored in two 3-dimensional arrays. The size of the first and second dimension is the length of sentence, and the last dimension correponds to the number of all tags. For example, 'bottomTable[i][j][X]' represents if the last rule applied is binary, the score of applying tag 'X' over span (i, j).

1.3.2 Inference

The inference process predicts the best parsing for the given sentence. Now we already have the 'bottomTable' and 'topTable'. A recursive algorithm could be used to parse the sentence. Two functions 'getBestUnaryRule(i, j, parentTag)' and 'getBestBinaryRule(i, j, parentTag)' are used. Each find the best unary rule and binary rule for span (i, j) using 'bottomTable' and 'topTable', respectively. The two functions are called alternately, which means that 'getBestUnaryRule' function will only call 'getBestBinaryRule' function and vice versa.

As we know that the root tag is 'ROOT' and the first rule must be unary, we could start the recursive process by calling the 'getBestUnaryRule(sentence, 0, $n - 1$, 'ROOT')'.

As unary closure is applied in the training phase, it is necessary to unwrap the closed unary rules to the original rule chains and remove identity unary rules.

2 Experiments

2.1 Performance

When setting both 'maxTrainLength' and 'maxTestLength' to 15, v and h are both set to 2, the resulting average F1 scores on validation set and test set are 84.78 and 84.75, respectively. The decoding time is around 10-15 seconds. The following sections will discuss the impact of some variables in the process.

Algorithm 1: CKY table constructing algorithm with alternating layers

Input : Grammar with all unary and binary rules and their ruleScore map; Lexicon with all tags and scoreTagging map; A sentence of length n

Output : BottomTable and TopTable

Initialization: set all elements of bottomTable and topTable to Double.NEGATIVE_INFINITY;

for $i = 0$ **to** $n - 1$ **do**
  for *tag in lexicon* **do**
    bottomTable[i][j][tag] = scoreTagging(sentence[i], tag);
  end
  for *unary rule* $X \rightarrow Y$ *in grammar* **do**
    bottomTable[i][j][X] = $\max$(bottomTable[i][j][X], ruleScore(X->Y) + bottomTable[i][j][Y]);
  end
end
for *diff* $= 0$ **to** $n - 1$ **do**
  for $i = 0$ **to** $n-$*diff*-1 **do**
    j = i + diff;
    for *binary rule* $X \rightarrow YZ$ *in grammar* **do**
      for $k = i$**to** $j - 1$ **do**
        topTable[i][j][X] = $\max$(topTable[i][j][X], ruleScore(X->YZ) + bottomsTable[i][k][Y] + bottomTable[k+1][j][Z]);
      end
    end
    for *unary rule* $X \rightarrow Y$ *in grammar* **do**
      bottomTable[i][j][X] = $\max$(bottomTable[i][j][X], ruleScore(X->Y) + topTable[i][j][Y])
    end
  end
end

2.2 Max Length of Sentence

In this experiment, we set both h and v to 2, fixed the 'maxTrainLength', and tested the F-1 score with different 'maxTestLength' on the validation set. The purpose of this experiment is to test when the grammar and lexicon is identical, if it is harder to parse longer sentences. The results, shown in Fig. 4, basically verified this assumption.

For long sentences, it is especially hard to get the exact same parse as the gold parsing. And the F-1 score decreases as the length of sentence becomes larger, too. Larger amount of training data also increases the F-1 score and exact matching rate of the parser. For example, consider when test sentence length is between 31 and 40, when 'maxTrainLength' is set to 40, the F-1 score is significantly higher than when 'maxTrainLength' = 15. It is due to the lack of experience in the training set to process longer sentences when 'maxTrainLength' is low.

2.2.1 Order of Markovization

It is natural to assume that the order of Markovization (value of h, v) has a large impact on the model performance. A series of experiments were conducted on the validation set with the max length of training and test both set to 15 and the choices of v and h were different throughout the experiments. The results are shown in Fig. 5. The sizes of grammar when using different h and v are also provided for reference.

There are a little information we could tell from the graphs:

- Both horizontal and vertical annotation is crucial for the accuracy of parsing. When h and v are both set to zero, the F-1 score is only 46.36, and the exact parsing is only 1.66%.
- The grammar size increases with larger v or h values. Peer or parental annotations will distinct further between the same tags. Thus the number of tokens and the grammar size will raise along with the values of v or h.
- If the value of h is fixed, the F-1 score usually increases in the beginning, and decreases after $v = 2$ or 3 although the grammar size is still increasing. If a $v \leq 3$ is fixed, the F-1 score increases sharply after h changes from 0 to 1, afterwards, it only changes slightly with different h. One possible interpretation is that the larger the value of h and v is, the more history information is kept. However, it also means a larger grammar and far

Figure 4: The F1 score and exact matching rate with respect to different size of training data and different length of test sentences.

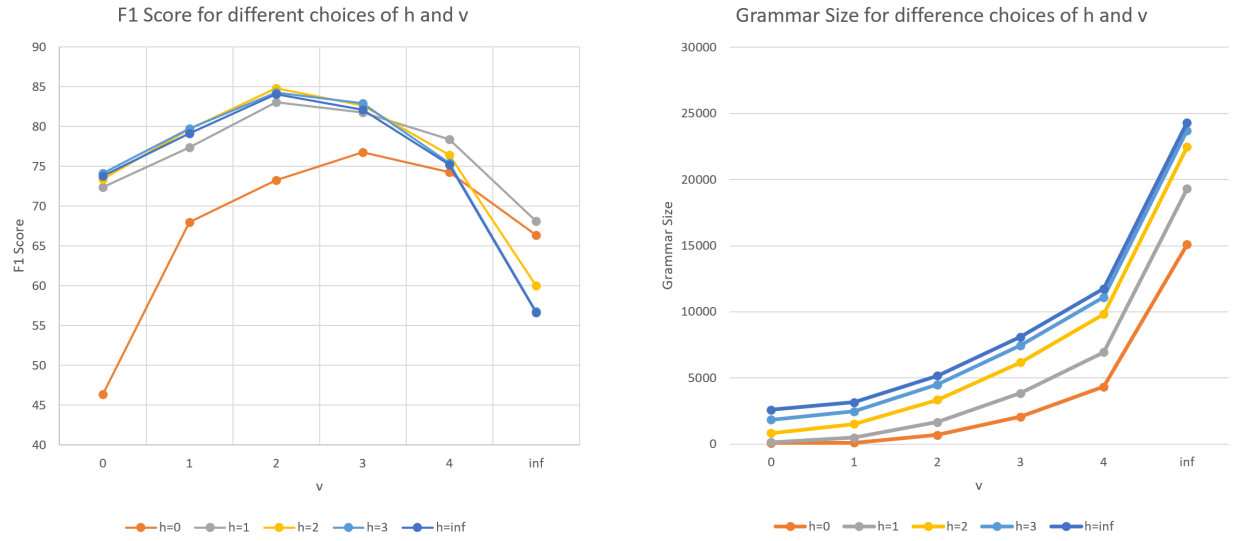


Figure 5: The F1 score and grammar size with different values of the Markovization order h and v.

sparser rules. Thus, there are many unseen rules in the test data, and rules in the training data but are never used afterwards.