PCFG And Parsing

Nitesh Gundavarapu,
A53317353,
University of California, San Diego
nbgundav@ucsd.edu

Abstract

In this project, Probabilistic Context Free Grammar is explored for Parsing and CKY parsing algorithm is implemented. Further, grammar extraction is modified through horizontal and vertical markovization, unary splits etc. to improve upon the generalization accuracy, namely F1 score. Performance is optimized for through cache-locality techniques, using primitives and a few algorithmic tricks. Results on PennTreeBank dataset show an F1 score of ***85.22*** and a decoding time of 12s with 15 length validation split and an F1 scrore of ***81.61*** and a decoding time of 42min with a 40 length validation split h,v=2. Finally, the dependence/independence of operations in CKY is analyzed and a multi-threaded implementation is provided to parallelize the independent pieces of the algorithm to achieve a ***16.9*** minute decoding time on length 40 val set.

1 Introduction

1.1 CKY Parser

The objective of a parser is to extract the parse tree given a sentence that agrees the most with the underlying Grammar. So, this is a computationally expensive problem and bad algortihms or sloppy implementations can blow up the run-time. CKY Algorithm is a dynamic programming algorithm closely related to Viterbi and Bellman-Ford but with key modifications to handle specifics of parsing. For instance, unaries in grammar introduce cyclic dependecies which are resolved by constraining the parse trees to contain alternative binary and unary non-terminal nodes. Some of the key implementation challenges and details are discussed in Sec.2.1.

1.2 Grammar

Since parsing algorithms try to satisfy the underlying grammatical rules, grammar can affect both the accuracy and runtime of parsing algorithms. PennTreeBank dataset contains sentences annotated by linguists with grammatical rules i.e. parse trees and the same dataset is used in this project. In Sec. 2.3, different modifications to grammar inspired from linguistics are tried out and its effect on the parsing and runtime of CKY are analyzed.

2 Implementation Details:

In this section, a few key implementation details, challenges faced and proposed solutions are proposed to each case:

2.1 CKY Algorithm:

2.2 Cache-Locality

While computing the solution to the sub-problems, cache-locality can be taken advantage of by simply ordering the for loops in the same order as our storage charts for DP.

2.2.1 Base Case vs Unaries vs Pre-Terminals

In standard CKY, notice that once we compute the pre-terminals, unary rules are applied first for a span of length 1, followed by binary rule for a span of length 2. Binary DP array in this progression is never filled for a span of 1. So, a natural solution is to place the pre-terminals in the binary DP array. This avoids ambiguity between unary and pre-terminal resolution during the decoding phase.

2.2.2 Efficiency

If the probability for a left-child is zero, then the corresponding parent-children gramatical rule will never occur. This fact is taken advantage of by enumerating non-terminals by their left children.

2.3 Grammar Extraction:

2.3.1 Vertical Markovization:

PCFG assumes strong conditional independence between the non-terminals which is not empricially

MaxLength	F1Score	Decoding	h,v
15	85.22	11.3s	h=2,k=2
40	81.61	17min	h=2,k=2

Table 1: Best Models On PennTreeBank Validation Set

valid. An NP-VP co-occuring with parent S is more likely than NP-PP. But conditioned on another parent, the probability could be different. To take this into account, parent/ancestor context is added to the non-terminals in grammar extraction phase. The number of parents/non-terminals added is denoted by v

2.3.2 Horizontal Markovization:

Following similar logic, sibling context can influence the condtiitonal probabilities. The horizontal markovization context controlled by parameter h is implemented.

2.3.3 Unary Splitting:

Parse tree algorithms can often wrongly fit a high-probability parse tree by forcibly converting using unary rules. So, a unary context is added to the non-terminals connected through unary rule.

3 Experiments:

3.1 Main Results

F1 scores and run-times are presented for the best CKY performance on validation set of PennTreeBank is presented in Tab. 1. It is observed that a horizontal markovization with a context of two, a vertical markovization with a context of two, unary splitting and multi-threading as discussed in Sec.4, along with cache-locality hacks, using primitive types and extensive debugging leads to the final results.

3.2 Ablation Study:

In Tab.2, ablation for horizontal markovization keeping vertical fixed at v=2 is presented. In Tab.3 vertical markovization is varied keeping horizontal fixed at ∞.

We can observe that increasing both horizontal and vertical context beyond a point leads to over-fitting. This can be attributed to increased grammar size with increase in context. Increased grammar leads to more rules and that leads to over-fitting.

In Tab.**??**, ablations on all the important components of the algorithm are provided upon adding one after the other.

v=2,h=?	F1Score
Baseline (h=∞,v=1)	79.11
v=2,h=0	73.91
v=2,h=1	83.53
v=2,h=2	85.22
v=2,h=3	85.16
v=2,h=4	84.87
v=2,h=∞	84.81

Table 2: Varying Horizontal Markovization on PennTreeBank with maxLength=15

h=2,v=?	F1Score
Baseline (h=∞,v=1)	79.11
h=2,v=1	79.67
h=2,v=2	84.78
h=2,v=3	82.65
h=2,v=4	77.2

Table 3: Varying Vertical Markovization on PennTreeBank with maxLength=15. **Note**: Unary Splitting is not used for this experiment. So, h=2,v=2 is different here.

Model	F1Score
Baseline	79.11
h=2	79.67
h=2,v=2	84.78
h=2,v=2,tag split	85.22

Table 4: Ablation of various components over baseline

Threading	F1Score	Decode Time
Single	81.61	42min
Multi (4thread)	81.61	16.9min

Table 5: Multi-threading in context

4 Multi-threading:

In CKY algorithm, for a given left span and right span, and given solution to all the smaller span subproblems, different non-terminals are independent of each other. This fact can be exploited by a multi-threaded algorithm by computing the score of the non-terminals in parallel at each level i.e. if we have the non-termnials in the inner-most for loop, we can simply replace that for loop with a multi-threaded executor. The same is implemented and a 3X speed up is obtained in performance.

5 Conclusion:

In this project, CKY parsing and its implementation are explored in detailed and the effect of grammatical context on parse trees is explored. Further, competitive performance is demonstrated on PennTreeBank dataset. Finally, a fast and accurate solution is provided by taking advantage of inherent independence of sub-problems in CKY.