# Natural Language Processing HW 5

Christopher Buss, Zhengxian Lin, Sanad Saha

## 1 Learning a grammar

#### Q1.a Preprocessing: replace all one-count words (those occurring only once, case-sensitive) with <unk>. **Explain why we do this.**

One-count words will already have a very low probability in our tree, so we can reduce the amount of rules in our grammar by mapping one-count words to <unk> tokens. Again, since these words are so rare, that this approach will simplify our grammar without affecting its performance.

#### Q1.b Binarization: we binarize a non-branching tree (recursively) in the following way. What are the properties of this binarization scheme? Why we would use such a scheme? (What are the alternatives, and why we don't use them?) Is this binarized CFG in Chomsky Normal Form (CNF)?

The binarization scheme has two properties: 1) new rules can have redundant left-hand sides and 2) no rule can produce more than two non-terminals. An example of the first property is *VB’* in the resulting tree above. Though this deviates from the original tree, it allows the new tree to be more flexible with new examples, which helps with the problem of data sparseness. The second property is required if we want to use the CKY algorithm. The other option would be to not binarize our grammar and use the Earley algorithm instead. However, the Earley algorithm is considerably more complicated than the CKY algorithm, so we opt to binarize our tree.

This binarized tree isn’t quite in Chomsky Normal Form. It is true that none of its rules produce more than two non-terminals, but there are still rules in which a non-terminal produces a non-terminal. Since each terminal is preceded by a part-of-speech tag, we end up having rules where a non-terminal can produce another *single* non-terminal. For example, we can have,

NP -> PRP

PRP -> “it”

The first rule, NP -> PRP violates Chomsky Normal Form.

#### Q1.c Learn a PCFG from trees. We group rules into three categories: binary rules (A -> B C), unary rules (A -> B), and lexical rules (A -> w). How many rules are there in each group?

We have used our learn\_pcfg.py to learn a PCFG from the binarized version of the trees. We grouped the rules into the required categories, count of each rules appearance is given below:

Binary Rules: 226

Unary Rules: 176

Lexical Rules: 197

## 2 CKY Parser

#### Q2.a Design a toy grammar toy.pcfg and its binarized version toy.pcfg.bin such that the above two trees are indeed the best parses for the two input sentences, respectively. How many strings do these two grammars generate?

#### Q2.b Write a CKY parser. Verify that you get the desired output in toy.parsed. Note that your output trees should be debinarized (see examples above).

#### Q2.c Now that you passed the toy testcase, apply your CKY parser to the real test set. Your program should handle (any levels of) unary rules correctly, even if there are unary cycles. How many sentences failed to parse? Your CKY code should simply output a line NONE for these cases (so that the number of lines in test.parsed should be equal to that of test.txt). What are the main reasons of parsing failures?

#### Q2.d Now modify your parser so that it rst replaces words that appear once or less in the training data. Now how many sentences fail to parse?

#### Q2.e Evaluate your parsing accuracy. Report the labeled precisions, recalls and F-1 scores of the two parsing results. Do their differences make sense?