Polynomial time parsing of PCFGs

Christopher Manning

(some slides from Pi-Chuan Chang)

0. Chomsky Normal Form

- All rules are of the form X → Y Z or X → w.
- A transformation to this form doesn’t change the weak generative capacity of CFGs.
  - With some extra book-keeping in symbol names, you can even reconstruct the same trees with a detransform
  - Unaries/empties are removed recursively
  - n-ary rules introduce new nonterminals (n > 2)
  - VP → V NP PP becomes VP → V + V PP and VPP → NP PP
- In practice it’s a pain
  - Reconstructing n-aries is easy
  - Reconstructing unaries can be trickier
- But it makes parsing easier/more efficient

An example: before binarization...

After binarization...

Treebank: empties and unaries

Constituency Parsing

PCFG

Rule Probs $\theta_e$:

$\theta_{42}: S \rightarrow NP VP$

$\theta_{43}: NP \rightarrow NN NNS$

$\theta_{44}: NN \rightarrow \text{Factory}$

$\theta_{45}: NNS \rightarrow \text{payrolls}$

PTB Tree NoFuncTags NoEmpties High Low NoUnaries

TOP S-HLN NP-SUB(C) VP -NONE- VB

\( S \rightarrow \text{payrolls} \)
1. Cocke-Kasami-Younger (CKY)
Constituency Parsing

Factory payrolls fell in September

Viterbi (Max) Scores

NP—NN NNS 0.73
i_{max} = (0.13)(0.0023) (0.0014)
= 1.87 \times 10^{-7}

NP—NPP NNS 0.056
i_{max} = (0.056)(0.001) (0.0014)
= 7.84 \times 10^{-8}

Extended CKY parsing

• Unaries can be incorporated into the algorithm
• Messy, but doesn’t increase algorithmic complexity
• Empties can be incorporated
• Use fenceposts
• Doesn’t increase complexity; essentially like unaries
• Binarization is vital
• Without binarization, you don’t get parsing cubic in the length of the sentence
• Binarization may be an explicit transformation or implicit in how the parser works (Early-style dotted rules), but it’s always there.

The CKY algorithm (1960/1965)
… generalized

Function CKY(words, grammar) returns most probable parse/prob score = new double[#(words)+2][#(words)+2][#(nonterms)]
back = new Pair[#(words)+2][#(words)+2][#(nonterms)]
for i=0; i<#(words); i++
    for A in nonterms
        if A -> words[i] in grammar
            score[i][i+1][A] = P(A -> words[i])
        //handle unaries
        boolean added = true
        while added
            added = false
            for A, B in nonterms
                if score[i][i+1][B] > 0 && A->B in grammar
                    prob = P(A->B)*score[i][i+1][B]
                    if prob > score[i][i+1][A]
                        score[i][i+1][A] = prob
                        back[i][i+1][A] = B
                        added = true
        //handle unaries
        boolean added = true
        while added
            added = false
            for A, B in nonterms
                prob = P(A->B)*score[i][i+1][B]
                if prob > score[i][i+1][A]
                    score[i][i+1][A] = prob
                    back[i][i+1][A] = B
                    added = true
    return buildTree(score, back)
For each A, only keep the "A \rightarrow BC" with highest prob.

\[ \text{prob} = \frac{3}{2} \text{score} \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]

\[ \text{prob} = \frac{3}{2} \text{score}_{i,i+1} \text{A} = P(A \rightarrow \text{cats cats cats}) \]
Unary rules:
alchemy in the land of treebanks

Efficient CKY parsing

• CKY parsing can be made very fast (!), partly due to the simplicity of the structures used.
  • But that means a lot of the speed comes from engineering details
  • And a little from cleverer filtering
  • Store chart as (ragged) 3 dimensional array of float (log probabilities)
  • \[\text{score[start][end][category]}\]
  • For treebank grammars the load is high enough that you don’t really gain from lists of things that were possible
  • 50 wds: \((50x50)/2x(1000 to 20000)x4\) bytes = 5–100MB for parse triangle. Large. (Can move to beam for span[i][j].)
  • Use int to represent categories/words (Index)

Efficient CKY parsing

• Provide efficient grammar/lexicon accessors:
  • E.g., return list of rules with this left child category
  • Iterate over left child, check for zero (Neg. inf.) prob of \(X[i,j]\) (abort loop), otherwise get rules with \(X\) on left
  • Some \(X[i,j]\) can be filtered based on the input string
  • Not enough space to complete a long flat rule?
  • No word in the string can be a CC?
  • Using a lexicon of possible POS for words gives a lot of constraint rather than allowing all POS for words
  • Cf. later discussion of figures-of-merit/A* heuristics

Quiz Question!

What constituents (with what probability can you make?)

3. Evaluating Parsing Accuracy

• Most sentences are not given a completely correct parse by any currently existing parser.
• For Penn Treebank parsing, the standard evaluation is over the number of correct constituents (labeled spans).
  • \([\text{label}, \text{start}, \text{finish}]\)
• A constituent is a triple, which must be exact in the true parse for the constituent to be marked correct.
• The LP/LR \(F\), is the micro-averaged harmonic mean of labeled constituent precision and recall
• This isn’t necessarily a great measure … me and many other people think dependency accuracy would be better.
How good are PCFGs?

- Robust (usually admit everything, but with low probability)
- Partial solution for grammar ambiguity: a PCFG gives some idea of the plausibility of a sentence
- But not so good because the independence assumptions are too strong
- Give a probabilistic language model
  - But in a simple case it performs worse than a trigram model
- WSJ parsing accuracy: about 73% LP/LR F1
  - The problem seems to be that PCFGs lack the lexicalization of a trigram model

Putting words into PCFGs

- A PCFG uses the actual words only to determine the probability of parts-of-speech (the preterminals)
- In many cases we need to know about words to choose a parse
  - The head word of a phrase gives a good representation of the phrase’s structure and meaning
    - Attachment ambiguities
      - The astronomer saw the moon with the telescope
    - Coordination
      - the dogs in the house and the cats
    - Subcategorization frames
      - put versus like

(Head) Lexicalization

- put takes both an NP and a VP
  - Sue put [the book] NP on the table VP
  - * Sue put [the book] VP on the table NP
- like usually takes an NP and not a PP
  - Sue likes [the book] NP on the table PP
  - * Sue likes [the book] PP on the table NP
- We can’t tell this if we just have a VP with a verb, but we can if we know what verb it is

4. Accurate Unlexicalized Parsing: PCFGs and Independence

- The symbols in a PCFG define independence assumptions:

  S → NP VP  
  NP → DT NN  

  - At any node, the material inside that node is independent of the material outside that node, given the label of that node.
  - Any information that statistically connects behavior inside and outside a node must flow through that node.

Non-Independence I

- Independence assumptions are often too strong.

<table>
<thead>
<tr>
<th></th>
<th>All NPs</th>
<th>NPs under S</th>
<th>NPs under VP</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP PP</td>
<td>11%</td>
<td>9%</td>
<td>21%</td>
</tr>
<tr>
<td>DT NN</td>
<td>6%</td>
<td>9%</td>
<td>25%</td>
</tr>
<tr>
<td>PRP</td>
<td>4%</td>
<td>3%</td>
<td>4%</td>
</tr>
</tbody>
</table>

- Example: the expansion of an NP is highly dependent on the parent of the NP (i.e., subjects vs. objects).
Michael Collins (2003, COLT)

Independence Assumptions
- PCFGs
- Lexicalized PCFGs

73% accuracy

88% accuracy

Non-Independence II

- Who cares?
  - NB, HMMs, all make false assumptions!
  - For generation/LMs, consequences would be obvious.
  - For parsing, does it impact accuracy?
- Symptoms of overly strong assumptions:
  - Rewrites get used where they don’t belong.
  - Rewrites get used too often or too rarely.

Breaking Up the Symbols

- We can relax independence assumptions by encoding dependencies into the PCFG symbols:

  Parent annotation [Johnson 98]

  Marking possessive NPs

  Parent annotation [Johnson 98]

  Marking possessive NPs

- What are the most useful features to encode?

Annotations

- Annotations split the grammar categories into subcategories.
- Conditioning on history vs. annotating
  - (NP^S → PRP) is a lot like (NP → PRP | S)
  - (PNP-POS → NNP POS) isn’t history conditioning.
- Feature grammars vs. annotation
  - Can think of a symbol like NNP^POS as NP [parent:NP, +POS]
- After parsing with an annotated grammar, the annotations are then stripped for evaluation.

Experimental Setup

- Corpus: Penn Treebank, WSJ

  Training: sections 02-21
  Development: section 22 (first 20 files)
  Test: section 23

- Accuracy - F1: harmonic mean of per-node labeled precision and recall.
- Size – number of symbols in grammar.
  - Passive / complete symbols: NP, NP^S
  - Active / incomplete symbols: NP → NP CC

Experimental Process

- We’ll take a highly conservative approach:
  - Annotate as sparingly as possible
  - Highest accuracy with fewest symbols
  - Error-driven, manual hill-climb, adding one annotation type at a time
Lexicalization

- Lexical heads are important for certain classes of ambiguities (e.g., PP attachment):
- Lexicalizing grammar creates a much larger grammar.
- More data needed
- How necessary is lexicalization?
  - Biliteral vs. monolexical selection
  - Closed vs. open class lexicalization

Unlexicalized PCFGs

- What do we mean by an “unlexicalized” PCFG?
  - Grammar rules are not systematically specified down to the level of lexical items
  - NP-stocks is not allowed
  - NP-S-CC is fine
  - Closed vs. open class words (NP-S-the)
  - Long tradition in linguistics of using function words as features or markers for selection
  - Contrary to the biliteral idea of semantic heads
  - Open-class selection really a proxy for semantics

- Honesty checks:
  - Number of symbols: keep the grammar very small
  - No smoothing: over-annotating is a real danger

Horizontal Markovization

- Horizontal Markovization: Merges States

Vertical Markovization

- Vertical Markov order: rewrites depend on past $k$ ancestor nodes.
  (cf. parent annotation)

Unary Splits

- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.

- Solution: Mark unary rewrite sites with -U

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base: v+2v</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>Annotation</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>77.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>UNARY</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
</tbody>
</table>
Tag Splits

- Problem: Treebank tags are too coarse.
- Example: Sentential, PP, and other prepositions are all marked IN.
- Partial Solution: Subdivide the IN tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>78.3</td>
<td>8.0K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.3</td>
<td>8.1K</td>
</tr>
</tbody>
</table>

Other Tag Splits

- UNARY-DT: mark demonstratives as DT^AU ("the X" vs. "those")
- UNARY-RB: mark phrasal adverbs as RB^AU ("quickly" vs. "very")
- TAG-PA: mark tags with non-canonical parents ("not" is an RB^VP)
- SPLIT-AUX: mark auxiliary verbs with -AUX [cf. Charniak 97]
- SPLIT-CC: separate "but" and "&" from other conjunctions
- SPLIT-%: "%" gets its own tag.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>SPLIT-IN</td>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>SPLIT-AX</td>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>SPLIT-AUX</td>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>SPLIT-%</td>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>SPLIT-CC</td>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>

Yield Splits

- Problem: sometimes the behavior of a category depends on something inside its future yield.
- Examples: Possessive NPs, Finite vs. infinite VPs, Lexical heads!
- Solution: annotate future elements into nodes.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>82.3</td>
<td>9.7K</td>
</tr>
<tr>
<td>POSS-NP</td>
<td>83.1</td>
<td>9.8K</td>
</tr>
<tr>
<td>SPLIT-VP</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
</tbody>
</table>

Distance / Recursion Splits

- Problem: vanilla PCFGs cannot distinguish attachment heights.
- Solution: mark a property of higher or lower sites:
  - Contains a verb.
  - Is (non)-recursive.
  - Base NPs [cf. Collins 99]
  - Right-recursive NPs

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>85.7</td>
<td>10.5K</td>
</tr>
<tr>
<td>BASE-NP</td>
<td>86.0</td>
<td>11.7K</td>
</tr>
<tr>
<td>DOMINATES-V</td>
<td>86.9</td>
<td>14.1K</td>
</tr>
<tr>
<td>RIGHT-REC-NP</td>
<td>87.0</td>
<td>15.2K</td>
</tr>
</tbody>
</table>

A Fully Annotated Tree

Final Test Set Results

<table>
<thead>
<tr>
<th>Parser</th>
<th>LP</th>
<th>LR</th>
<th>F1</th>
<th>CB</th>
<th>0 CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Magerman 95</td>
<td>84.9</td>
<td>84.6</td>
<td>84.7</td>
<td>1.26</td>
<td>56.6</td>
</tr>
<tr>
<td>Collins 96</td>
<td>86.3</td>
<td>85.8</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Klein &amp; M 03</td>
<td>86.9</td>
<td>85.7</td>
<td>86.3</td>
<td>1.10</td>
<td>60.3</td>
</tr>
<tr>
<td>Charniak 97</td>
<td>87.4</td>
<td>87.5</td>
<td>87.4</td>
<td>1.00</td>
<td>62.1</td>
</tr>
<tr>
<td>Collins 99</td>
<td>88.7</td>
<td>88.6</td>
<td>88.6</td>
<td>0.90</td>
<td>67.1</td>
</tr>
</tbody>
</table>

- Beats “first generation” lexicalized parsers.