Polynomial time parsing of PCFGs

Gerald Penn
(some slides from Pi-Chuan Chang and Christopher Manning)

0. Chomsky Normal Form

- All rules are of the form $X \rightarrow Y Z$ or $X \rightarrow w$.
- 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 ($n > 2$).
- $VP \rightarrow VPP$ becomes $VP \rightarrow V \cap PP$ and $\cap PP \rightarrow NP$.
- 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 $\theta$,

- $S \rightarrow NP \ VP$
- $NP \rightarrow N N S$
- ...
- $N S \rightarrow N N S$
- $N S \rightarrow N S$

Factory pays fell IN NN in September
1. Cocke-Kasami-Younger (CKY) Constituency Parsing

Factory payrolls fell in September

Viterbi (Max) Scores

\[
\text{NP} \quad \text{NNP} \quad \text{NN} \quad \text{NNS} \\
\text{NP} \quad 1.87 \times 10^3 \\
\text{NN} \quad 0.0023 \\
\text{NNP} \quad 0.003 \\
\text{NNS} \quad 0.0014
\]

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
  - All sorts of optimizations depend on this
  - Binarization may be an explicit transformation or implicit in how the parser works (earliest-style started first), but it’s almost always there

The CKY algorithm (1960/1965) … generalized

The CKY algorithm (1960/1965) … generalized

for \( a \in A \), \( b \in B \) in sentences

\[
\text{score}=\text{score}([\text{begin}, \text{end}])[A][B][a][b]
\]

return buildTree(score, back)
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 clever filtering
- Store chart as (tagged) 3 dimensional array of float (log probabilities)
  - score[(start, end), (category)]
  - For treebank grammar the load is high enough that you don’t really gain from lots of things that work possible
  - 50 words: (50*50)*(log 1000 to 2000)*4 bytes = 5-10MB for parse triangle: Large: Do not store lean for spans [1, 1]
- Use int to represent categories/words (index)

Quiz Question!

<table>
<thead>
<tr>
<th>PP</th>
<th>IN</th>
<th>NNS</th>
<th>VB</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.002</td>
<td>0.01</td>
<td>0.003</td>
<td>0.001</td>
</tr>
<tr>
<td>PP</td>
<td>VB</td>
<td>NNS</td>
<td>NNS</td>
</tr>
<tr>
<td>0.045</td>
<td>0.013</td>
<td>0.0014</td>
<td>0.0001</td>
</tr>
<tr>
<td>runs</td>
<td>down</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 (tagg int) prob of X[i,j] (short 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 CCT
  - Using a version of possible POS for words given a list of constraint rather than utilizing all POS for words
  - Cf. later discussion of figures-of-merit/NP heuristics

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).
  - [level, start, finish]
  - A constituent is a triple, which must be exact in the true parse for the constituent to be marked correct.
  - The Labeled F$_1$ is the micro-averaged harmonic mean of labeled constituent precision and recall
  - This isn’t necessarily a great measure ... many people think dependency accuracy or raw data likelihood 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% LPLR 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 astronaut saw the moon with the telescope*
    - *The astronauts saw the moon with the telescopes*
  - 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] [on the table]
  - Sue put [the book] [on the table]
- *like* usually takes an NP and not a PP
  - Sue likes [the book] [on the table]
  - Sue likes [on the table]
- 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:
  - $S \rightarrow NP \ VP$
  - $NP \rightarrow DT \ NN$
- 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.
- 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
  - 73% accuracy
- Lexicalized PCFGs
  - 88% accuracy

Non-independence II
- Who cares?
  - NL: 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

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

Experimental Setup
- Corpus: Penn Treebank, WSJ
- Training: sections 02-23
- 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.
  - 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.
  - Sophisticated smoothing needed
  - Smarter parsing algorithms needed
  - More data needed
- How necessary is lexicalization?
  - Bilexical 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-branching is not allowed
  - Closed vs. open class words (NP-Schle)
  - Long tradition in linguistics of using function words as features or markers for selection
  - Contrary to the bilexical 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-annoying is a real danger

Horizontal Markovization

- Horizontal Markovization: Merges States

Vertical Markovization

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

Vertical and Horizontal

- Examples:
  - Base treebank: \( x=1, y=0 \)
  - Johnson: \( x=2, y=0 \)
  - Collins: \( x=2, y=0 \)
  - Best \( F1 \): \( x=3, y=2 \)

<table>
<thead>
<tr>
<th>Model</th>
<th>( F1 )</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>72.8</td>
<td>7.5K</td>
</tr>
<tr>
<td>Best ( F1 )</td>
<td>77.8</td>
<td>14K</td>
</tr>
</tbody>
</table>

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>Annotation</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>14K</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>7.0K</td>
</tr>
</tbody>
</table>

Other Tag Splits

- UNARY-DT: mark demonstratives as DT~U ("the X" vs. "those")
- UNARY-RE: mark pronominal adverbs as RB~U ("quickly" vs. "very")
- TAG-PAP: mark tags with non-canonical parents ("not" is an RB~VP)
- SPLIT-SUX: mark auxiliary verbs with ~ALX (cf. Charniak 97)
- SPLIT-CCC: separate "but" and "/" from other conjunctions
- SPLIT-PK: "%" 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</td>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>SPLIT-PK</td>
<td>81.2</td>
<td>8.5K</td>
</tr>
<tr>
<td>SPLIT-SUX</td>
<td>81.6</td>
<td>9.0K</td>
</tr>
<tr>
<td>SPLIT-CCC</td>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>SPLIT-PK</td>
<td>81.6</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>18.1K</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>Magrane 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>85.5</td>
<td>86.0</td>
<td>1.14</td>
<td>59.9</td>
</tr>
<tr>
<td>Klein &amp; Ro 93</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>1.09</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.
An alternative ... memoization

bestScore(X.i,j,s)
if (j == i + 1)
    return tagScore(X, s[i])
else
    return max score(X -> Y Z) *
        bestScore(Y, i, k) * bestScore(Z, k, j)

• Call: bestParse(Start, 1, sent.length(), sent)
• Will this parser work?
• Memory/time requirements?

2. An alternative ... memoization

• A recursive (CNF) parser:
  
  bestParse(X.i,j,s)
  if (j == i + 1)
      return X -> s[i]
  else
      (X -> Y Z, k) = argmax score(X -> Y Z) *
      bestScore(Y, i, k) * bestScore(Z, k, j)
  parse.parent = X
  parse.leftChild = bestParse(Y, i, k)
  parse.rightChild = bestParse(Z, k, j)
  return parse

A memoized parser

• A simple change to record scores you know:

  bestScore(X,i,j)
  if (scores(X,i,j) == null)
      if (j == i + 1)
          score = tagScore(X, s[i])
      else
          score = max score(X -> Y Z) *
            bestScore(Y, i, k) * bestScore(Z, k, j)
      scores(X,i,j] = score
  return scores(X,i,j)

• Memory and time complexity?

Runtime in practice: super-cubic!

Rule State Reachability

• Worse in practice because longer sentences "unlock" more of the grammar
• Many states are more likely to match larger span
• And because of various "systems" issues ... cache misses, etc.

Example: NP CC . NP

Example: NP CC NP . PP
Treebank Splits

- The treebank comes with annotations (e.g., LOC, SUB), etc.
- Whole set together hurt the baseline.
- Some (SUB) were less effective than our equivalents.
- One in particular was very useful (NP-TMP) when pushed down to the head tag.
- We marked gapped S nodes as well.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous</td>
<td>82.0</td>
<td>0.3k</td>
</tr>
<tr>
<td>NP-TMP</td>
<td>82.2</td>
<td>5.4k</td>
</tr>
<tr>
<td>GAPPED-S</td>
<td>82.3</td>
<td>87k</td>
</tr>
</tbody>
</table>

Evaluating Constituent Accuracy:
LP/LR measure

- Let C be the number of correct constituents produced by the parser over the test set, M be the total number of constituents produced, and N be the total in the correct version.
- Precision = CM
- Recall = CN
- It is possible to artificially inflate either one.
- Thus people typically give the F-measure (harmonic mean) of the two. Not a big issue here, like average.
- This isn’t necessarily a great measure … me and many other people think dependency accuracy would be better.