Polynomial time parsing of PCFGs

Christopher Manning

(some slides from Pi-Chuan Chang)

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 ($n > 2$).
- 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 $\theta$, $\begin{align*}
\theta_0: & S \rightarrow NP \ VP \\
\theta_1: & NP \rightarrow NN \ NNS \\
\theta_4: & NN \rightarrow \text{Factory} \\
\theta_{43}: & NNS \rightarrow \text{payrolls} \\
\end{align*}$
1. Cocke-Kasami-Younger (CKY) Constituency Parsing

Factory payrolls fell in September

Viterbi (Max) Scores

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

NP—NNP NNS 0.056
i_{NN} = (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

The CKY algorithm (1960/1965) … generalized

For span = 2 to #(words):
  for begin = 0 to #(words) - span:
    end = begin + span
    for split = begin+1 to end-1:
      for A, B, C in nonterms:
        prob = score[begin][split][B] \cdot score[split][end][C] \cdot P(A \rightarrow BC)
        if prob > score[begin][end][A]:
          score[begin][end][A] = prob
          back[begin][end][A] = new Triple(split, B, C)
      //handle unaries
      boolean added = true
      while added:
        added = false
        for A, B in nonterms:
          prob = P(A \rightarrow B) \cdot score[begin][end][B];
          if prob > score[begin][end][A]:
            score[begin][end][A] = prob
            back[begin][end][A] = B
            added = true
      return buildTree(score, back)
For each A, only keep the "A->BC" with highest prob.

\[
\text{prob} = \text{score}[\text{begin}] \cdot \text{score}[\text{split}] \cdot P(\text{A} \rightarrow \text{BC})
\]

@PP->P

NP

→

cats

cats

cats

cats

\[
\text{score}[i][i+1][\text{A}] = P(\text{A} \rightarrow \text{words}[i]);
\]

÷

NP

@VP->V

NP

→

cats

cats

cats

cats

÷

NP

VP

→

cats

cats

cats

÷

NP

VP

→

scratch

scratch

scratch

scratch

÷

NP

VP

→

scratch

scratch

scratch

scratch

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

claws

claws

claws

claws

÷

NP

VP

→

claws

claws

claws

claws

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with

with

÷

NP

VP

→

with

with

with
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).
- 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.

2. An alternative ... memoization

- A recursive (CNF) parser:
  - bestParse(X,i,j,s):
  - if j = i + 1
  - return X \rightarrow s[j]
  - (X\rightarrow Y Z, k) = \arg\max score(X \rightarrow Y Z) \ast
    - bestScore(Y,i,k,s) \ast bestScore(Z,k,j,s)
  - parse.parent = X
  - parse.leftChild = bestParse(Y,i,k,s)
  - parse.rightChild = bestParse(Z,k,j,s)
  - return parse

An alternative ... memoization

- bestScore(X,i,j,s):
  - if j = i + 1
    - return tagScore(X, s[j])
  - else
    - return max score(X \rightarrow Y Z) \ast
      - bestScore(Y, i, k) \ast bestScore(Z,k,j)

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

- A simple change to record scores you know:
  
  ```
  bestScore(X,i,j,s)
  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?

3. Evaluating Parsing Accuracy

- Most sentences are not given a completely correct parse by any currently existing parsers.
- Standardly for Penn Treebank parsing, evaluation is done in terms of the percentage of correct constituents (labeled spans).
- A constituent is a triple, all of which must be in the true parse for the constituent to be marked correct.

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 (microaveraged).
- Precision = C/M
- Recall = C/N
- 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.
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] on the table
  - * Sue put [the book] on the table
- like usually takes an NP and not a PP
  - Sue likes [the book]
  - * 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 → 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.

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

Michael Collins (2003, COLT)
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:

Annotations

- Annotations split the grammar categories into sub-categories.
- Conditioning on history vs. annotating
  - $\text{PNP} \rightarrow \text{PRP}$ is a lot like $\text{PNP} \rightarrow \text{PRP} | \text{S}$
  - $\text{PNP-POS} \rightarrow \text{NNP POS}$ isn’t history conditioning.
- Feature grammars vs. annotation
  - Can think of a symbol like $\text{NP} \rightarrow \text{NP-POS}$ as $\text{NP} \left[ \text{parent:NP}, \text{+POS} \right]$
- 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 $\rightarrow$ 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+SS-CC is fine
    - Closed vs. open class words (NP+SS-the)
    - 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-annotating is a real danger
  - What do we mean by an "unlexicalized" PCFG?
  - Grammar rules are not systematically specified down to the level of lexical items
  - NP+SS-CC is fine
  - Closed vs. open class words (NP+SS-the)
  - 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

Horizontal Markovization

- Horizontal Markovization: Merges States

Vertical Markovization

- Vertical Markov order: rewrites depend on past k ancestor nodes.
  - Examples:
    - Raw treebank: \(v=1, h=\)
    - Johnson 98: \(v=2, h=\)
    - Collins 99: \(v=2, h=2\)
    - Best F1: \(v=3, h=2v\)

Vertical and Horizontal

- Examples:
  - Raw treebank: \(v=1, h=\)
  - Johnson 98: \(v=2, h=\)
  - Collins 99: \(v=2, h=2\)
  - Best F1: \(v=3, h=2v\)

Tag Splits

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

Unary Splits

- Problem: unary rewrites used to transmute categories so a high-probability rule can be used.
- Solution: Mark unary rewrite sites with `-U`
Other Tag Splits

- **UNARY-DT**: mark demonstratives as DT^U ("the X" vs. "those")
- **UNARY-RB**: mark phrasal adverbs as RB^U ("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></th>
<th>F1</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNARY-DT</td>
<td>80.4</td>
<td>8.1K</td>
</tr>
<tr>
<td>UNARY-RB</td>
<td>80.5</td>
<td>8.1K</td>
</tr>
<tr>
<td>TAG-PA</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-CC</td>
<td>81.7</td>
<td>9.1K</td>
</tr>
<tr>
<td>SPLIT-%</td>
<td>81.8</td>
<td>9.3K</td>
</tr>
</tbody>
</table>

Treebank Splits

- The treebank comes with annotations (e.g., -LOC, -SUBJ), etc.
- Whole set together hurt the baseline.
- Some (-SUBJ) 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>81.8</td>
<td>9.3K</td>
</tr>
<tr>
<td>NP-TMP</td>
<td>82.2</td>
<td>7.6K</td>
</tr>
<tr>
<td>GAPPED-S</td>
<td>82.3</td>
<td>9.7K</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></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>

Annotation

<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>

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></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>O 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.