### Statistical Parsing

Christopher Manning
CS224N

---

**(Head) Lexicalization of PCFGs**

Magerman 1995, Collins 1997; Charniak 1997

- The head word of a phrase gives a good representation of the phrase’s structure and meaning
- Puts the properties of words back into a PCFG

---

**Lexicalized Parsing was seen as the breakthrough of the late 90s**

- Eugene Charniak, 2000 JHU workshop: “To do better, it is necessary to condition probabilities on the actual words of the sentence. This makes the probabilities much tighter:
  - \( p(VP \rightarrow V NP NP) = 0.00151 \)
  - \( p(VP \rightarrow V NP NP | said) = 0.000001 \)
  - \( p(VP \rightarrow V NP NP | gave) = 0.01980 \)

- Michael Collins, 2003 COLT tutorial: “Lexicalized Probabilistic Context-Free Grammars … perform vastly better than PCFGs (88% vs. 73% accuracy)”

---

**Michael Collins (2003, COLT)**

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
<th>( F_1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCFGs (Charniak 97)</td>
<td>73.0%</td>
<td></td>
</tr>
<tr>
<td>Conditional Models – Decision Trees (Magerman 95)</td>
<td>84.2%</td>
<td></td>
</tr>
<tr>
<td>Lexical Dependencies (Collins 96)</td>
<td>85.5%</td>
<td></td>
</tr>
<tr>
<td>Conditional Models – Logistic (Ratnaparkhi 97)</td>
<td>86.9%</td>
<td></td>
</tr>
<tr>
<td>Generative Lexicalized Model (Charniak 97)</td>
<td>86.7%</td>
<td></td>
</tr>
<tr>
<td>Generative Lexicalized Model (Collins 97)</td>
<td>88.2%</td>
<td></td>
</tr>
<tr>
<td>Logistic-inspired Model (Charniak 99)</td>
<td>89.6%</td>
<td></td>
</tr>
<tr>
<td>Boosting (Collins 2000)</td>
<td>89.8%</td>
<td></td>
</tr>
</tbody>
</table>

---

**Parsing via classification decisions:**

Charniak (1997)

- A very simple, conservative model of lexicalized PCFG
- Probabilistic conditioning is “top-down” like a regular PCFG (but actual computation is bottom-up)
Charniak (1997) example

Lexicalization sharpens probabilities: Predicting heads

Lexicalization sharpens probabilities: rule expansion

Charniak (1997) linear interpolation/shrinkage

Charniak (1997) shrinkage example

Sparserness & the Penn Treebank
Sparseness & the Penn Treebank (2)

- Many parse preferences depend on bilexical statistics: likelihoods of relationships between pairs of words (compound nouns, PP attachments, ...)
- Extremely sparse, even on topics central to the WSJ:
  - stocks plummeted: 2 occurrences
  - stocks stabilized: 1 occurrence
  - stocks skyrocketed: 0 occurrences
  - # stocks discussed: 0 occurrences
- So far there has been very modest success in augmenting the Penn Treebank with extra unannotated materials or using semantic classes – once there is more than a little annotated training data.
  - Cf. Charniak 1997, Charniak 2000; but see McClosky et al. 2006
    [this recent self-training work is quite successful!]

Complexity of lexicalized PCFG parsing

- Work such as Collins (1997) and Charniak (1997) is $O(n^5)$ – but uses heuristic search to be fast in practice
- Eisner and Satta (2000, etc.) have explored various ways to parse more restricted classes of bilexical grammars in $O(n^4)$ or $O(n^3)$ time
  - Neat algorithmic stuff!!!
  - See example later from dependency parsing

Complexity of exhaustive lexicalized PCFG parsing

- Running time is $O(g^3 \times n^5)$!!

Refining the node expansion probabilities

- Charniak (1997) expands each phrase structure tree in a single step.
  - This is good for capturing dependencies between child nodes
  - But it is bad because of data sparseness.
  - A pure dependency, one child at a time, model is worse.
  - But one can do better by in between models, such as generating the children as a Markov process on both sides of the head (Collins 1997; Charniak 2000)
    - Cf. the accurate unlexicalized parsing discussion

Collins (1997, 1999); Bikel (2004)

- Collins (1999): also a generative model
- Underlying lexicalized PCFG has rules of form
  $$ P \rightarrow L_1 L_{j-1} \cdots L_i H R_k \cdots R_{j-1} R_j $$
- A more elaborate set of grammar transforms and factorizations to deal with data sparseness and interesting linguistic properties
  - Each child is generated in turn: given $P$ has been generated, generate $H$, then generate modifying nonterminals from head-adjacent outward with some limited conditioning
Overview of Collins’ Model

$P(t, w_i)$ generated
conditioning on

$L_t \rightarrow L_{t-1} \cdots L_1 (\text{subcat})(H(t, w_i))$

Smoothing for head words of modifying nonterminals

<table>
<thead>
<tr>
<th>Back-off level</th>
<th>$P_{Mx}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$M_x, \text{coord}, \text{punc}, P(H(t), \text{subcat}_w)$</td>
</tr>
<tr>
<td>1</td>
<td>$M_x, \text{coord}, \text{punc}, P(H(t), \text{subcat}_w)$</td>
</tr>
<tr>
<td>2</td>
<td>$i_{\text{tr}}$</td>
</tr>
</tbody>
</table>

- Other parameter classes have similar or more elaborate backoff schemes

Collins model … and linguistics

- Collins had 3 generative models: Models 1 to 3
- Especially as you work up from Model 1 to 3, significant linguistic modeling is present:
  - Distance measure: favors close attachments
  - Model is sensitive to punctuation
  - Distinguish base NP from full NP with post-modifiers
  - Coordination feature
  - Mark gapped subjects
  - Model of subcategorization; arguments vs. adjuncts
  - Slash feature/gap threading treatment of displaced constituents
  - Didn’t really get clear gains from this last one.

Bilexical statistics: Is use of maximal context of $P_{Mx}$ useful?

- Collins (1999): “Most importantly, the model has parameters corresponding to dependencies between pairs of headwords.”
- Gildea (2001) reproduced Collins’ Model 1 (like regular model, but no subcats)
  - Removing maximal back-off level from $P_{Mx}$ resulted in only 0.5% reduction in F-measure
  - Gildea’s experiment somewhat unconvincing to the extent that his model’s performance was lower than Collins’ reported results

Choice of heads

- If not bilexical statistics, then surely choice of heads is important to parser performance...
- Chiang and Bikel (2002): parsers performed decently even when all head rules were of form “if parent is X, choose left/rightmost child”
- Parsing engine in Collins Model 2-emulation mode: LR 88.55% and LP 88.80% on §00 (sent. len. ≤40 words)
  - compared to LR 89.9%, LP 90.1%
### Use of maximal context of $P_{Mw}$

**Back-off level** | **Number of accesses** | **Percentage**
--- | --- | ---
0 | 3,257,309 | 1.40%
1 | 24,294,084 | 11.0%
2 | 191,527,387 | 87.4%
Total | 219,078,780 | 100.0%

Number of times parsing engine was able to deliver a probability for the various back-off levels of the mod-word generation model, $P_{Mw}$, when testing on §00 having trained on §§02–21.

### Bilexical statistics are used often

*The 1.49% use of bilexical dependencies suggests they don’t play much of a role in parsing*
*But the parser pursues many (very) incorrect theories*
*So, instead of asking how often the decoder can use bigram probability on average, ask how often while pursuing its top-scoring theory*
*Answering question by having parser constrain-parse its own output*
*First: parse §00*
*Second: feed parse trees as constraints*
*Percentage of time parser made use of bigram statistics shot up to 28.8%*
*So, used often, but use barely affect overall parsing accuracy*
*Exploratory Data Analysis suggests explanation*
*That is, distributions that include head words are usually sufficiently similar to those that do not as to make almost no difference in terms of accuracy*
POS tag splits, commonest words: effectively a class-based model

- Proper Nouns (NNP):
  - NNP-12: John, Robert, James
  - NNP-2: J., E., L.
  - NNP-1: Bush, Noriega, Peters
  - NNP-15: New, San, Wall
  - NNP-3: York, Francisco, Street

- Personal pronouns (PRP):
  - PRP-0: it, he, I
  - PRP-1: it, he, they
  - PRP-2: it, them, him

The Latest Parsing Results...

<table>
<thead>
<tr>
<th>Parser</th>
<th>F1 ≤ 40 words</th>
<th>F1 all words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Klein &amp; Manning unlexicalized 2003</td>
<td>86.3</td>
<td>85.7</td>
</tr>
<tr>
<td>Matsuzaki et al. simple EM latent states 2005</td>
<td>86.7</td>
<td>86.1</td>
</tr>
<tr>
<td>Charniak generative (“maxent inspired”) 2000</td>
<td>90.1</td>
<td>89.5</td>
</tr>
<tr>
<td>Petrov and Klein NAACL 2007</td>
<td>90.6</td>
<td>90.1</td>
</tr>
<tr>
<td>Charniak &amp; Johnson discriminative reranker 2005</td>
<td>92.0</td>
<td>91.4</td>
</tr>
</tbody>
</table>

Statistical parsing inference: The General Problem

- Someone gives you a PCFG G
- For any given sentence, you might want to:
  - Find the best parse according to G
  - Find a bunch of reasonably good parses
  - Find the total probability of all parses licensed by G
- Techniques:
  - CKY, for best parse: can extend it:
    - To k-best: naively done, at high space and time cost - k^2 time/k space cost, but there are cleverer algorithms!
    - To all parses, summed probability: the inside algorithm
  - Beam search
  - Agenda/chart-based search

Parse as search definitions

- Grammar symbols: S, NP, @S->NP,
- Parse items/edges represent a grammar symbol over a span:
  - the[0,1]
  - NP:[0,2]
- Backtraces/traversals represent the combination of adjacent edges into a larger edges:
  - S:[0,3]
    - NP:[0,2]
    - VP:[2,3]

Parse trees and parse triangles

- A parse tree can be viewed as a collection of edges and traversals.
- A parse triangle groups edges over the same span

Parsing as search: The parsing directed B-hypergraph

[Klein and Manning 2001]
Chart example: classic picture

Chart example: classic picture

Space and Time Bounds

Space = $O(\text{Edges})$

Time = $O(\text{Traversals})$

CKY Parsing

- In CKY parsing, we visit edges tier by tier:
  - Guarantees correctness by working inside-out.
  - Build all small bits before any larger bits that could possibly require them.
  - Exhaustive: the goal is in the last tier!

Beam Search

- State space search
- States are partial parses with an associated probability
  - Keep only the top scoring elements at each stage of the beam search
  - Find a way to ensure that all parses of a sentence have the same number of steps
    - Or at least are roughly comparable
  - Leftmost top-down CFG derivations in true CNF
  - Shift-reduce derivations in true CNF

Beam Search

- Time-synchronous beam search
- Beam at time $i$: successors of beam elements
- Beam at time $i + 1$: successors of beam elements

Kinds of beam search

- Constant beam size $k$
- Constant beam width relative to best item
  - Defined either additively or multiplicatively
  - Sometimes combination of the above two
  - Sometimes do fancier stuff like trying to keep the beam elements diverse
- Beam search can be made very fast
- No measure of how often you find model optimal answer
  - But can track correct answer to see how often/far gold standard optimal answer remains in the beam
Beam search treebank parsers?

- Most people do bottom up parsing (shift-reduce parsing or a version of left-corner parsing)
  - For treebank grammars, not much grammar constraint, so want to use data driven constraint
  - Adwait Ratnaparkhi 1996 [maxent shift-reduce parser]
  - Manning and Carpenter 1998 and Henderson 2004 left-corner parsers
- But top-down with rich conditioning is possible
  - Cf. Brian Roark 2001
- Don’t actually want to store states as partial parses
  - Store them as the last rule applied, with backpointers to the previous states that built those constituents (and a probability)
  - You get a linear time parser … but you may not find the best parses according to your model (things “fall off the beam”)

Agenda-based parsing

- Agenda-based parsing
  - Keep a list of edges called an agenda
    - Edges are triples \([X, i, j]\)
    - The agenda is a priority queue
  - Every time the score of some \(\delta(X, i, j)\) improves (i.e. gets lower):
    - Stick the edge \([X, i, j]\)-score into the agenda
    - (Update the backtrace for \(\delta(X, i, j)\) if yours storing them)

Agenda-based parsing

- Agenda-based parsing
  - Step II: While agenda not empty
    - Get the “next” edge \([X, i, j]\) from the agenda
    - Fetch all compatible neighbors \([Y, j, k]\) or \([Z, k, i]\)
      - Compatible means that there are rules \(A \rightarrow X Y\) or \(B \rightarrow Z X\)
    - Build all parent edges \([A, i, k]\) or \([B, k, j]\) found
      - \(\delta(A, i, k) = \delta(X, i, j) + \delta(Y, j, k) + \text{P}(A \rightarrow X Y)\)
    - If we’ve improved \(\delta(A, i, k)\), then stick it on the agenda
    - Also project unary rules:
      - Fetch all unary rules \(A \rightarrow X\), score \([A, i, j]\) built from this rule on \([X, i, j]\) and put on agenda if you’ve improved \(\delta(A, i, j)\)
  - When do we know we have a parse for the root?

Agenda-based parsing

- Agenda-based parsing
  - Open questions:
    - Agenda priority: What did “next” mean?
    - Efficiency: how do we do as little work as possible?
    - Optimality: how do we know when we find the best parse of a sentence?
    - If we use \(\delta(X, i, j)\) as the priority:
      - Each edge goes on the agenda at most once
      - When an edge pops off the agenda, its best parse is known (why?)
      - This is basically uniform cost search (i.e., Dijkstra’s algorithm).
        - [Cormen, Leiserson, and Rivest 1990; Knuth 1970]
We want to work on good parses inside-out.

- CKY does this synchronously, by tiers.
- Uniform-cost does it asynchronously, ordering edges by their best known parse score.
- Why best parse is known:
  - Adding structure incurs probability cost.
  - Trees have lower probability than their sub-parts.
  - The best-scored edge in the agenda cannot be waiting on any of its sub-edges.
  - We never have to propagate. We don’t explore truly useless edges.

\[ \beta \leq \beta + \epsilon \]

Example of uniform cost search vs. CKY parsing:
The grammar, lexicon, and sentence

- \( S \rightarrow NP\ VP \ % 0.9 \)
- \( S \rightarrow VP \ % 0.1 \)
- \( VP \rightarrow V\ NP \ % 0.6 \)
- \( VP \rightarrow V \ % 0.4 \)
- \( NP \rightarrow NP\ NP \ % 0.3 \)
- \( NP \rightarrow N \ % 0.7 \)
- \( N \rightarrow people \ % 0.8 \)
- \( N \rightarrow fish \ % 0.1 \)
- \( V \rightarrow people \ % 0.1 \)
- \( V \rightarrow fish \ % 0.6 \)
- \( V \rightarrow tanks \ % 0.3 \)
- \( people \ fish \ tanks \)

What can go wrong?

- We can build too many edges.
  - Most edges that can be built, shouldn’t.
  - CKY builds them all!

  **Speed: build promising edges first.**

- We can build in a bad order.
  - Might find bad parses for parse item before good parses.
  - Will trigger best-first propagation.

**Correctness: keep edges on the agenda until you’re sure you’ve seen their best parse.**