Statistical Parsing

Christopher Manning
CS224N

(Head) Lexicalization of PCFGs

- 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:
  - \( p(\text{VP} \rightarrow \text{V NP NP}) = 0.00151 \)
  - \( p(\text{VP} \rightarrow \text{V NP NP} | \text{said}) = 0.00001 \)
  - \( p(\text{VP} \rightarrow \text{V NP NP} | \text{gave}) = 0.01980 \)"

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

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

- a. \( h = \text{profits}; c = \text{NP} \)
- b. \( ph = \text{rose}; pc = S \)
- c. \( P(h|ph,c,pc) \)
- d. \( P(r|h,c,pc) \)
Lexicalization sharpens probabilities: rule expansion

- E.g., probability of different verbal complement frames (often called "subcategorizations")

<table>
<thead>
<tr>
<th>Local Tree</th>
<th>come</th>
<th>take</th>
<th>think</th>
<th>want</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP → V</td>
<td>9.5%</td>
<td>2.6%</td>
<td>4.6%</td>
<td>5.7%</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>3.1%</td>
<td>2.2%</td>
<td>2.0%</td>
<td>2.3%</td>
</tr>
<tr>
<td>VP → V PP</td>
<td>34.5%</td>
<td>2.3%</td>
<td>8.7%</td>
<td>13.1%</td>
</tr>
<tr>
<td>VP → V SBJ</td>
<td>6.6%</td>
<td>5.3%</td>
<td>31.0%</td>
<td>12.2%</td>
</tr>
<tr>
<td>VP → V S</td>
<td>2.4%</td>
<td>1.3%</td>
<td>8.4%</td>
<td>70.8%</td>
</tr>
<tr>
<td>VP → V NP S</td>
<td>0.3%</td>
<td>3.7%</td>
<td>0.0%</td>
<td>0.3%</td>
</tr>
<tr>
<td>VP → V PRT NP</td>
<td>0.3%</td>
<td>1.8%</td>
<td>0.0%</td>
<td>0.0%</td>
</tr>
<tr>
<td>VP → V PRT PP</td>
<td>0.1%</td>
<td>1.5%</td>
<td>0.2%</td>
<td>0.2%</td>
</tr>
</tbody>
</table>

Lexicalization sharpens probabilities: Predicting heads

- "Bilexical probabilities"

- \( p(\text{prices} | \text{n-plural}) = 0.013 \)
- \( p(\text{prices} | \text{n-plural, NP}) = 0.013 \)
- \( p(\text{prices} | \text{n-plural, NP, S}) = 0.025 \)
- \( p(\text{prices} | \text{n-plural, NP, S, v-past}) = 0.052 \)
- \( p(\text{prices} | \text{n-plural, NP, S, v-past, fell}) = 0.146 \)

Charniak (1997) linear interpolation/shrinkage

\[
P(h|c, p) = \lambda_1(p_{MLE}(h|c, p)) + \lambda_2(p_{MLE}(C|p, h, c, p)) + \lambda_3(p_{MLE}(h|c, p)) + \lambda_4(p_{MLE}(h|c))
\]

- \( \lambda_1(p) \) is here a function of how much one would expect to see a certain occurrence, given the amount of training data, word counts, etc.
- \( C(h|ph) \) is semantic class of parent headword
- Techniques like these for dealing with data sparseness are vital to successful model construction

Charniak (1997) shrinkage example

- Allows utilization of rich highly conditioned estimates, but smoothes when sufficient data is unavailable
- One can’t just use MLEs: one commonly sees previously unseen events, which would have probability 0.

Sparseness & the Penn Treebank

- The Penn Treebank – 1 million words of parsed English WSJ – has been a key resource (because of the widespread reliance on supervised learning)
- But 1 million words is like nothing:
  - 965,000 constituents, but only 66 WHADJP, of which only 6 aren’t how much or how many, but there is an infinite space of these
  - How clever/original/incompetent (at risk assessment and evaluation) …
- Most of the probabilities that you would like to compute, you can’t compute

Quiz question!

- Write down two other possible WHADJP, which have:
  - Different adjective heads
  - Are 3 or more words long
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 rather more successful]

Complexity of lexicalized PCFG parsing

- A[\(d_1\)]
- B[\(d_2\)]
- C[\(d_3\)]
- \(i, k, j \Rightarrow n^3\)
- \(A[\(d_1\)], B[\(d_2\)], C[\(d_3\)] \Rightarrow G^3\)
- Done naively, \(G^3\) is huge (\(G^3 = g^3 V^3\); unworkable)
- \(A, B, C \Rightarrow g^3\)
- \(d_1, d_2 \Rightarrow n^2\)

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

Complexity of exhaustive 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

Collins (1997, 1999); Bikel (2004)

- Collins (1999): also a generative model
- Underlying lexicalized PCFG has rules of form
  \[
P \rightarrow L_i L_{j-1} \cdots L_1 HR_1 \cdots R_{i-1} R_i
\]
- 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

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
Overview of Collins’ Model

Modifying nonterminals generated in two steps

Smoothing for head words of modifying nonterminals

Collins model … and linguistics

Bilexical statistics: Is use of maximal context of \( P_{Mu} \) useful?

Choice of heads

Other parameter classes have similar or more elaborate backoff schemes

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.

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 \( P_{Mu} \) 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

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 $P_{MW}$

<table>
<thead>
<tr>
<th></th>
<th>LR</th>
<th>LP</th>
<th>CBs</th>
<th>0 CBs</th>
<th>≤2 CBs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full model</td>
<td>89.9</td>
<td>90.1</td>
<td>0.78</td>
<td>68.8</td>
<td>89.2</td>
</tr>
<tr>
<td>No bigrams</td>
<td>89.5</td>
<td>90.0</td>
<td>0.80</td>
<td>68.0</td>
<td>88.8</td>
</tr>
</tbody>
</table>

Performance on §00 of Penn Treebank on sentences of length ≤40 words

<table>
<thead>
<tr>
<th>Back-off level</th>
<th>Number of accesses</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3,257,309</td>
<td>1.49</td>
</tr>
<tr>
<td>1</td>
<td>24,294,084</td>
<td>11.0</td>
</tr>
<tr>
<td>2</td>
<td>191,527,387</td>
<td>87.4</td>
</tr>
<tr>
<td>Total</td>
<td>219,078,780</td>
<td>100.0</td>
</tr>
</tbody>
</table>

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-parses its own output
  - parse as normal on §§02–21
  - parse §00
  - 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
  - distributions that include head words are usually sufficiently similar to those that do not as to make almost no difference in terms of accuracy

Charniak (2000) NAACL:
A Maximum-Entropy-Inspired Parser

- There was nothing maximum entropy about it. It was a cleverly smoothed generative model
- Smoothes estimates by smoothing ratio of conditional terms (which are a bit like maxent features):
  \[
  P(t|I, f_{l,t}, f_{g,t}) = \frac{P(t|I, f_{l,t})}{P(t|I, f_{g,t})}
  \]
- Biggest improvement is actually that generative model predicts head tag first and then does $P(w|t, \ldots)$
  - Like Collins (1999)
  - Markovizes rules similarly to Collins (1999)
- Gets 90.1% LP/LR F score on sentences ≤ 40 wds

Petrov and Klein (2006):
Learning Latent Annotations

Can you automatically find good symbols?
- Brackets are known
- Base categories are known
- Induce subcategories
- Clever split/merge category refinement

EM algorithm, like Forward-Backward for HMMs, but constrained by tree.
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, lexicalized ('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 (like in MT)
  - Agenda/chart-based search

  Mainly useful if just want the best parse

Parsing as search definitions

- Grammar symbols: S, NP, @S->NP,
- Parse items/edges represent a grammar symbol over a span:
- Backtraces/traversals represent the combination of adjacent edges into a larger edges:

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**

- Earley dotted rules

**Space and Time Bounds**

- Space = $O(\text{Edges})$
- Time = $O(\text{Traversals})$

**Space**

- N start
- C labels
- N end

**Time**

- S
- C
- N split
- N end

**Earley dotted rules**

- NP → DT NN
- NP → DT · NN
- NP → DT · NN
- DT
- NN
- the
- cat

---

**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!

---

**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 your storing them)

---

**Agenda-Based Parsing**

- The agenda is a holding zone for edges.
- Visit edges by some ordering policy.
  - Combine edge with already-visited edges.
  - Resulting new edges go wait in the agenda.
- We might revisit parse items: A new way to form an edge might be a better way.
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 \(A \rightarrow XY\) or \(B \rightarrow Z\ X\)
- Build all parent edges \([A,i,k]\) or \([B,k,i]\) found
- \(\delta(A,i,k) \leq \delta(X,i,j) + \delta(Y,j,k) + P(A \rightarrow X Y)\)
- 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 \(\delta(A,i,k)\)

When do we know we have a parse for the root?

Uniform-Cost Parsing

- 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.

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

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

Speed: build promising edges first.

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

What can go wrong?

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

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