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! (Huang and Chiang 2005: http://www.cis.upenn.edu/~lhuang3/huang-iwpt.pdf)
  - To all parses, summed probability: the inside algorithm
- Beam search (like in MT)
- Agenda/chart-based search

$\{\text{Mainly useful if just want the best parse}\}$
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.

S:[0,3]

NP:[0,2]

DT:[0,1]

VV:[0,1]

VP:[2,3]

NN:[1,2]

cat:[1,2]

VBD:[2,3]

ran:[2,3]

- A parse triangle groups edges over the same span.

NN

DT

S→NP•VP

NP
Parsing as search: The parsing directed B-hypergraph

[S:payrolls [0,2]]
[S:fell [0,5]]
[VP:fell [2,5]]
[PP:in [3,5]]
[IN:in [3,4]]
[NN:payrolls [1,2]]
[NN:Factory [0,1]]
[VBD:fell [2,3]]
[NN:September [4,5]]

start

[S:payrolls [0,2]]
[S:fell [0,5]]
[VP:fell [2,5]]
[PP:in [3,5]]
[IN:in [3,4]]
[NN:payrolls [1,2]]
[NN:Factory [0,1]]
[VBD:fell [2,3]]
[NN:September [4,5]]

[NN:Factory [0,1]]
[NN:payrolls [1,2]]
[VBD:fell [2,3]]
[IN:in [3,4]]
[NN:September [4,5]]

X:h
[i,j]

[S:payrolls [0,2]]
[S:fell [0,5]]
[VP:fell [2,5]]
[PP:in [3,5]]
[IN:in [3,4]]
[NN:payrolls [1,2]]
[NN:Factory [0,1]]
[VBD:fell [2,3]]
[NN:September [4,5]]

start

[S:payrolls [0,2]]
[S:fell [0,5]]
[VP:fell [2,5]]
[PP:in [3,5]]
[IN:in [3,4]]
[NN:payrolls [1,2]]
[NN:Factory [0,1]]
[VBD:fell [2,3]]
[NN:September [4,5]]

start

[S:payrolls [0,2]]
[S:fell [0,5]]
[VP:fell [2,5]]
[PP:in [3,5]]
[IN:in [3,4]]
[NN:payrolls [1,2]]
[NN:Factory [0,1]]
[VBD:fell [2,3]]
[NN:September [4,5]]

S:payrolls [0,2]
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

- For general grammars
- Start with a table for recording $\delta(X,i,j)$
  - Records the best score of a parse of X over [i,j]
    - If the scores are negative log probabilities, then entries start at $\infty$ and small is good
    - This can be a sparse or a dense map
    - Again, you may want to record backtraces (traversals) as well, like CKY

Step 1: Initialize with the sentence and lexicon:
- For each word w and each tag t
  - Set $\delta(X,i,i) = \text{lex.score}(w,t)$
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 you are 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 \(A \rightarrow X Y\) or \(B \rightarrow Z X\)
  • Build all parent edges \([A,i,k]\) or \([B,k,j]\) 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?
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:
  \[ \beta \leq \beta + \varepsilon \]
  • 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 → NP VP % 0.9
- S → VP % 0.1
- VP → V NP % 0.6
- VP → V % 0.4
- NP → NP NP % 0.3
- NP → N % 0.7
- N → people % 0.8
- N → fish % 0.1
- N → tanks % 0.1
- V → people % 0.1
- V → fish % 0.6
- V → tanks % 0.3

- *people fish tanks*
Example of uniform cost search vs. CKY parsing:
CKY vs. order of agenda pops in chart

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>N[0,1] -&gt; people</td>
<td>0.8</td>
<td></td>
<td>V[0,1] -&gt; people</td>
<td>0.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP[0,1] -&gt; N[0,1]</td>
<td>0.56</td>
<td></td>
<td>VP[0,1] -&gt; V[0,1]</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,1] -&gt; VP[0,1]</td>
<td>0.004</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N[1,2] -&gt; fish</td>
<td>0.1</td>
<td></td>
<td>V[1,2] -&gt; fish</td>
<td>0.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP[1,2] -&gt; V[1,2]</td>
<td>0.07</td>
<td></td>
<td>S[1,2] -&gt; VP[1,2]</td>
<td>0.024</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[2,3] -&gt; VP[2,3]</td>
<td>0.012</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP[0,2] -&gt; NP[0,1]</td>
<td>0.01176</td>
<td></td>
<td>NP[1,2] -&gt; VP[2,3]</td>
<td>0.00042</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP[0,2] -&gt; V[0,1]</td>
<td>0.01176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,2] -&gt; NP[0,1]</td>
<td>0.012</td>
<td></td>
<td>VP[1,2] -&gt; V[0,1]</td>
<td>0.01176</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,2] -&gt; NP[0,2]</td>
<td>0.00042</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NP[1,3] -&gt; NP[1,2]</td>
<td>0.00147</td>
<td></td>
<td>VP[2,3] -&gt; VP[2,3]</td>
<td>0.00147</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP[1,3] -&gt; V[1,2]</td>
<td>0.0252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[1,3] -&gt; VP[2,3]</td>
<td>0.00756</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[1,3] -&gt; VP[1,3]</td>
<td>0.00252</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,3] -&gt; VP[1,3]</td>
<td>0.0127008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,3] -&gt; VP[0,1]</td>
<td>0.00127008</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,3] -&gt; VP[0,2]</td>
<td>0.0021168</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,3] -&gt; VP[1,3]</td>
<td>0.0000882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S[0,3] -&gt; VP[0,3]</td>
<td>0.000000882</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best
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 an 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.**
Speeding up agenda-based parsers

- Two options for doing less work
  - The optimal way: A* parsing
    - Klein and Manning (2003)
  - The ugly but much more practical way: “best-first” parsing
    - Caraballo and Charniak (1998)
    - Charniak, Johnson, and Goldwater (1998)
A* Context Summary Sharpness

Adding local information changes the intercept, but not the slope!
Best-First Parsing

- In best-first, parsing, we visit edges according a figure-of-merit (FOM).
  - A good FOM focuses work on “quality” edges.
    - The good: leads to full parses quickly.
    - The (potential) bad: leads to non-MAP parses.
    - The ugly: propagation
      - If we find a better way to build a parse item, we need to rebuild everything above it
  - In practice, works well!
Search in modern lexicalized statistical parsers

- Klein and Manning (2003b) do optimal A* search
  - Done in a restricted space of lexicalized PCFGs that “factors”, allowing very efficient A* search
- Collins (1999) exploits both the ideas of beams and agenda based parsing
  - He places a separate beam over each span, and then, roughly, does uniform cost search
- Charniak (2000) uses inadmissible heuristics to guide search
  - He uses very good (but inadmissible) heuristics – “best first search” – to find good parses quickly
  - Perhaps unsurprisingly this is the fastest of the 3.
Coarse-to-fine parsing

- Uses grammar projections to guide search
  - VP-VBF, VP-VBG, VP-U-VBN, ... → VP
  - VP[buys/VBZ], VP[drive/VB], VP[drive/VBP], ... → VP
- You can parse much more quickly with a simple grammar because the grammar constant is way smaller
- You restrict the search of the expensive refined model to explore only spans and/or spans with compatible labels that the simple grammar liked

- Very successfully used in several recent parsers
  - Charniak and Johnson (2005)
  - Petrov and Klein (2007)
Coarse-to-fine parsing: A visualization of the span posterior probabilities from Petrov and Klein 2007
Dependency parsing
Dependency Grammar/Parsing

- A sentence is parsed by relating each word to other words in the sentence which depend on it.
- The idea of dependency structure goes back a long way
  - To Pāṇini’s grammar (c. 5th century BCE)
- Constituency is a new-fangled invention
  - 20th century invention
- Modern work often linked to work of L. Tesniere (1959)
  - Dominant approach in “East” (Russia, China, …)
  - Basic approach of 1st millenium Arabic grammarians
- Among the earliest kinds of parsers in NLP, even in US:
  - David Hays, one of the founders of computational linguistics, built early (first?) dependency parser (Hays 1962)
Dependency structure

- Words are linked from head (regent) to dependent
- Warning! Some people do the arrows one way; some the other way (Tesniere has them point from head to dependent…).
- Usually add a fake ROOT so every word is a dependent

*Shaw Publishing acquired 30% of American City in March* $$
Relation between CFG to dependency parse

- A dependency grammar has a notion of a head
- Officially, CFGs don’t
- But modern linguistic theory and all modern statistical parsers (Charniak, Collins, Stanford, …) do, via hand-written phrasal “head rules”:
  - The head of a Noun Phrase is a noun/number/adj/…
  - The head of a Verb Phrase is a verb/modal/….
- The head rules can be used to extract a dependency parse from a CFG parse (follow the heads).
- A phrase structure tree can be got from a dependency tree, but dependents are flat (no VP!)
Propagating head words

- Small set of rules propagate heads
NB. Not all dependencies shown here

- Dependencies are inherently untyped, though some work like Collins (1996) types them using the phrasal categories.
Sources of information:
- bilexical dependencies
- distance of dependencies
- valency of heads (number of dependents)

A word’s dependents (adjuncts, arguments) tend to fall near it in the string.

These next 6 slides are based on slides by Jason Eisner and Noah Smith.
1. Start with left wall $\$
2. Generate root $w_0$
3. Generate left children $w_{-1}$, $w_{-2}$, ..., $w_{-\ell}$ from the FSA $\lambda_{w_0}$
4. Generate right children $w_1$, $w_2$, ..., $w_r$ from the FSA $\rho_{w_0}$
5. Recurse on each $w_i$ for $i \in \{-\ell, ..., -1, 1, ..., r\}$, sampling (steps 2-4)
6. Return $\alpha_{-\ell} ... \alpha_{-1} w_0 \alpha_1 ... \alpha_r$
It takes two to tango.

\[ O(n^5 N^3) \text{ if } N \text{ nonterminals} \]

\[ O(n^5) \text{ combinations} \]

\[ \begin{align*}
\text{goal} & \rightarrow \text{takes} \\
\text{takes} & \rightarrow \text{tango} \\
\text{tango} & \rightarrow \text{to} \\
\text{to} & \rightarrow \text{two} \\
\text{two} & \rightarrow \text{It} \\
\text{It} & \rightarrow \text{takes} \\
\text{takes} & \rightarrow \text{goal} \\
\end{align*} \]
Dependency Grammar Cubic Recognition/Parsing (Eisner & Satta, 1999)

- **Triangles**: span over words, where tall side of triangle is the head, other side is dependent, and no non-head words expecting more dependents.

- **Trapezoids**: span over words, where larger side is head, smaller side is dependent, and smaller side is still looking for dependents on its side of the trapezoid.
Dependency Grammar Cubic Recognition/Parsing (Eisner & Satta, 1999)

A triangle is a head with some left (or right) subtrees.

One trapezoid per dependency.

It takes two to tango.
Cubic Recognition/Parsing (Eisner & Satta, 1999)

- $O(n)$ combinations
- $O(n^3)$ combinations
- $O(n^3)$ combinations

Gives $O(n^3)$ dependency grammar parsing
Evaluation of Dependency Parsing: Simply use (labeled) dependency accuracy

Accuracy = \frac{\text{number of correct dependencies}}{\text{total number of dependencies}}

= \frac{2}{5} = 0.40 = 40\%
McDonald et al. (2005 ACL): Online Large-Margin Training of Dependency Parsers

- Builds a discriminative dependency parser
- Can condition on rich features in that context
  - Best-known recent dependency parser
  - Lots of recent dependency parsing activity connected with CoNLL 2006/2007 shared task
- Doesn’t/can’t report constituent LP/LR, but evaluating dependencies correct:
  - Accuracy is similar to but a fraction below dependencies extracted from Collins:
    - 90.9% vs. 91.4% ... combining them gives 92.2% [all lengths]
  - Stanford parser on length up to 40:
    - Pure generative dependency model: 85.0%
    - Lexicalized factored parser: 91.0%
Score of a parse is the sum of the scores of its dependencies
Each dependency is a linear function of features times weights
Feature weights are learned by MIRA, an online large-margin algorithm
  • But you could think of it as using a perceptron or maxent classifier
Features cover:
  • Head and dependent word and POS separately
  • Head and dependent word and POS bigram features
  • Words between head and dependent
  • Length and direction of dependency
Extracting grammatical relations from statistical constituency parsers

[de Marneffe et al. LREC 2006]

• Exploit the high-quality syntactic analysis done by statistical constituency parsers to get the grammatical relations [typed dependencies]
• Dependencies are generated by pattern-matching rules

Bills on ports and immigration were submitted by Senator Brownback
Discriminative Parsing
Discriminative Parsing as a classification problem

- Classification problem
  - Given a training set of iid samples $T=\{(X_1,Y_1) \ldots (X_n,Y_n)\}$ of input and class variables from an unknown distribution $D(X,Y)$, estimate a function $\hat{h}(X)$ that predicts the class from the input variables
  - The observed $X$'s are the sentences.
  - The class $Y$ of a sentence is its parse tree
  - The model has a large (infinite!) space of classes, but we can still assign them probabilities
    - The way we can do this is by breaking whole parse trees into component parts

1. Distribution-free methods
2. Probabilistic model methods
Motivating discriminative estimation (1)

A training corpus of 108 (imperative) sentences.

Based on an example by Mark Johnson
Motivating discriminative estimation (2)

- In discriminative models, it is easy to incorporate different kinds of features
  - Often just about anything that seems linguistically interesting
- In generative models, it’s often difficult, and the model suffers because of false independence assumptions
- This ability to add informative features is the real power of discriminative models for NLP.
  - Can still do it for parsing, though it’s trickier.
Discriminative Parsers

- Discriminative Dependency Parsing
  - Not as computationally hard (tiny grammar constant)
  - Explored considerably recently. E.g. McDonald et al. 2005
- Make parser action decisions discriminatively
  - E.g. with a shift-reduce parser
- Dynamic-programmed Phrase Structure Parsing
  - Resource intensive! Most work on sentences of length \( \leq 15 \)
  - The need to be able to dynamic program limits the feature types you can use
- Post-Processing: Parse reranking
  - Just work with output of \( k \)-best generative parser
Discriminative models

Shift-reduce parser Ratnaparkhi (98)

• Learns a distribution \(P(T|S)\) of parse trees given sentences using the sequence of actions of a shift-reduce parser

\[
P(T \mid S) = \prod_{i=1}^{n} P(a_i \mid a_1...a_{i-1}S)
\]

• Uses a maximum entropy model to learn conditional distribution of parse action given history

• Suffers from independence assumptions that actions are independent of future observations as with CMM/MEMM

• Higher parameter estimation cost to learn local maximum entropy models

• **Lower** but still good accuracy: 86% - 87% labeled precision/recall
Discriminative dynamic-programmed parsers

- Taskar et al. (2004 EMNLP) show how to do joint discriminative SVM-style (“max margin) parsing building a phrase structure tree also conditioned on words in $O(n^3)$ time
  - In practice, totally impractically slow. Results were never demonstrated on sentences longer than 15 words
- Turian et al. (2006 NIPS) do a decision-tree based discriminative parser
- Research continues….
  - Finkel, Kleeman, and Manning (2008 ACL) feature-based parser is just about practical.
    - We do parse long sentences
Discriminative Models – Distribution
Free Re-ranking (Collins 2000)

- Represent sentence-parse tree pairs by a feature vector $F(X,Y)$
- Learn a linear ranking model with parameters $\alpha$ using the boosting loss

<table>
<thead>
<tr>
<th>Model</th>
<th>LP</th>
<th>LR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins 99</td>
<td>88.3%</td>
<td>88.1%</td>
</tr>
<tr>
<td>(Generative)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collins 00</td>
<td>89.9%</td>
<td>89.6%</td>
</tr>
<tr>
<td>(BoostLoss)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

13% error reduction
Still very close in accuracy to generative model [Charniak 2000]
Charniak and Johnson (2005 ACL):
Coarse-to-fine $n$-best parsing and MaxEnt discriminative reranking

- Builds a maxent discriminative reranker over parses produced by (a slightly bugfixed and improved version of) Charniak (2000).
- Gets 50 best parses from Charniak (2000) parser
  - Doing this exploits the “coarse-to-fine” idea to heuristically find good candidates
- Maxent model for reranking uses heads, etc. as generative model, but also nice linguistic features:
  - Conjunct parallelism
  - Right branching preference
  - Heaviness (length) of constituents factored in
- Gets 91% LP/LR F1 (on all sentences! – up to 80 wd)