Bottom-up Parsing

Micha Elsner

February 21, 2013
Why parsing?

- As linguistic inquiry
  - What do we need to put in our model to do a good job?
  - Early on, lots of connections between syntactic theorists and comp ling parsing people
  - Now not so much
- To add long-distance dependencies
  - Correct some errors in tagging, LMs
- To aid information extraction, translation
  - Via compositional semantics
  - An active area; results not yet amazing
- To model writing style, information structure, processing
  - Parser as model of processing
Parsing: basics of representation

- Parsing: find syntactic structure for input sentence
- What kind of structure?
- Most popular answer given by Penn Treebank project
  - Marcus, Santorini and Marcinkiewicz ‘93
- Many person-hours of grad student labor
- The PTB uses a phrase structure grammar...
- Loosely based on Government and Binding theory
- Plus some null elements (traces) for movement
- Many researchers use this formalism by default
  - Better than making your own treebank!

To the degree that most grammatical formalisms tend to capture the same regularities this can still be a successful strategy even if no one formalism is widely preferred over the rest.

—Charniak
What PTB trees look like

(S
  (NP-SBJ (NNP Ms.) (NNP Haag))
  (VP (VBZ plays)
    (NP (NNP Elianti)))
  (. .)))
What PTB trees look like

```
S
  NP-SBJ-2
    NP
      DT NN IN NP
      the role of Calimone
    NP
      PP
      VBN
      played
      by
      NNP
      NNP
      Kim Catrall

VP
  VBD
  was
  VBN
  NP
  PP
  ADVP-MNR
  attributed
  -NONE-
  NNP
  NNP
  -NONE-
  TO
  NNP
  NNP
  mistaken
  to
  Christina Haag
```
Most of the influential statistical parsing projects don’t do traces or movement

- Algorithms that actually move things are horribly inefficient
- Real problem (still open) is to get long-distance dependencies

Nor do they do the function annotations (like -SBJ)
Although these can be done fairly easily
Without these, the grammar is context-free

This lecture: context-free grammar parsing and some important ingredients for a good parser
Review: CFGs

- Produce a string of *terminals*
- By expanding *nonterminal symbols*
- Using rewriting *rules*

\[
\begin{align*}
S & \rightarrow NP \ VP \\
NP & \rightarrow DT \ NN \\
NP & \rightarrow NNP \\
DT & \rightarrow \text{the} \\
NN & \rightarrow \text{cat} \\
NN & \rightarrow \text{mat} \\
VP & \rightarrow V \\
VP & \rightarrow V \ PP \\
V & \rightarrow \text{sat} \\
PP & \rightarrow \text{IN} \ NP \\
IN & \rightarrow \text{on}
\end{align*}
\]
Bottom-up Recognizer for CFG

Relies on a data structure called the *chart*

- Like trellis for HMMs

```
<table>
<thead>
<tr>
<th>length 6</th>
<th>@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>length 5</td>
<td>@2</td>
</tr>
<tr>
<td>length 4</td>
<td>@3</td>
</tr>
<tr>
<td>length 3</td>
<td>@4</td>
</tr>
<tr>
<td>length 2</td>
<td>@5</td>
</tr>
<tr>
<td>length 1</td>
<td>@6</td>
</tr>
</tbody>
</table>
```

beginning:

```
the cat sat on the mat
length 1
length 6
```
Step 1: terminal rules

beginning:

D N VP
V
P D N

the cat sat on the mat

length 6

@1  @2  @3  @4  @5  @6

length 1

length 1  length 2  length 3  length 4  length 5  length 6

the  the  cat  sat  on  the  mat
NP (length 2) from 1-2 because NP → D N and D from 1-1 and N from 2-2
A few steps forward

S (length 3) from 1-3 because $S \rightarrow NP \ VP$ and NP from 1-2 and VP from 3-3
## Probabilistic CFGs (PCFGs)

Each rule gets a probability \( P(\text{right side}|\text{left side}) \)

- Since these are conditional, they sum to 1 for each LHS

<table>
<thead>
<tr>
<th>Rule</th>
<th>Right Side</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>NP VP</td>
<td>1</td>
</tr>
<tr>
<td>NP</td>
<td>DT NN</td>
<td>0.9</td>
</tr>
<tr>
<td>NP</td>
<td>NNP</td>
<td>0.1</td>
</tr>
<tr>
<td>DT</td>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>NN</td>
<td>cat</td>
<td>0.5</td>
</tr>
<tr>
<td>NN</td>
<td>mat</td>
<td>0.5</td>
</tr>
<tr>
<td>VP</td>
<td>V</td>
<td>0.4</td>
</tr>
<tr>
<td>VP</td>
<td>V PP</td>
<td>0.6</td>
</tr>
<tr>
<td>V</td>
<td>sat</td>
<td>1</td>
</tr>
<tr>
<td>PP</td>
<td>IN NP</td>
<td>1</td>
</tr>
<tr>
<td>IN</td>
<td>on</td>
<td>1</td>
</tr>
</tbody>
</table>
The model

A tree $T$ is made by repeatedly applying rules:

- The probability of a tree $T$ and string $W$ is the product of all the rule probs
- These include rules that insert nonterminals
- And rules that insert words

$$P(T, W_{1:n}) = \prod_{A \rightarrow BC \in T} P(BC|A) \prod_{A \rightarrow W \in T} P(W_i|A)$$

$$P(NP \ VP|S)P(D \ N|NP)P(\text{the}|D)P(\text{cat}|N)P(V|VP)P(\text{sat}|V)$$

$$1 \times .9 \times 1 \times .5 \times .4 \times 1$$
CKY algorithm with numbers

Each word gets $P(NT | word)$
When we apply a rule, we multiply the rule prob and the probs of the component parts from the chart

\[ P(NP \rightarrow DN) = 0.9, \quad P(D \text{ at } 1 - 1) = 1 \quad \text{and} \quad P(N \text{ at } 2 - 2) = 0.5 \]
More formally

CKY: Cocke-Kasami-Younger algorithm

A max-product algorithm for PCFGs
- Is a dynamic program
- Quite similar to Viterbi for HMMs
- Works for grammar with binary rules
  - Discuss unary and n-ary rules in a second
- Also has a sum-product version
  - Which computes $P(W_{1:n})$
Definition

We operate by computing $\mu_A(i, l)$, the best way to parse a nonterminal of type $A$ and length $l$ starting at $i$:

$$
\mu_A(i, l) = \max_{\text{tree } T} P(W_{i:i+l}, T, \text{root of } T = A)
$$

- To do so, we must apply a rule to make $A$ from some things $B, C$
- These things must cover the span $(i, l)$
- We divide the span into $(i, j)$ and $(i + j, l - j)$
  - For instance, span $(1, 4)$ can be split: $(1, 2), (2, 4)$
  - Or $(1, 3), (3, 4)$
- We have to search over potential $j$

$$
\mu_A(i, l) = \max_{0 < j < l, A \rightarrow B C} P(A \rightarrow B C) \mu_B(i, j) \mu_C(i + j, l - j)
$$
The solution

As with HMMs, notice that the last chart entry is the answer:

$$\mu_S(1, n) = \max_{\text{tree } T} P(W_{1:n}, T, \text{root of } T = S)$$

Joint probability of the sentence and the best tree rooted at S

- The conditional is proportional to the joint
- By a factor of $P(W_{1:n})$
Efficiency

For an \( n \)-word sentence and \( G \) grammar rules, CKY runs in \( O(Gn^3) \) time

- The chart is (less than) \( n^2 \) size
- To fill in cell \( i - k \), search for split point \( j \)
- No more than \( n \) places to look
- Must look once per each rule
Binarizing the grammar

CKY searches for a single split point:

- So we need the grammar to be binary
- Each rule has (one or) two children
  - Unary (one-child) rules also cause problems, but see Charniak for these
- There is an equivalent binary grammar for any CFG
  - Chomsky Normal Form
Building a lousy parser

We now know how to build a parser:

- Take Penn Treebank, convert to Chomsky Normal (binary) form
- Estimate rule probs by smoothed max likelihood
  \[
  \hat{P}(A \rightarrow BC) = \frac{\#(A \rightarrow BC) + \lambda}{\#(A \rightarrow \bullet) + |B, C| \lambda}
  \]
- Parse with CKY

This parser isn’t very good

- 70% F-score... modern parsers >90%

Why not?
Stupid parser assumptions

This parser makes stupid independence assumptions:

- Decision to produce NT independent of expansion of NT
  - Eg, “expects [S sales to increase]” (non-finite clause)
  - vs “hopes [S sales will increase]” (finite clause)
  - Or “a [NN chair]” (count)
  - vs “*a [NN pork]” (mass)

- No subcategories, feature checking, etc

On the other hand, it’s too restrictive:

- Prob of “[ADJP very good]”, “[ADJP very very good]”, “[ADJP very very very good]” unrelated
- Depend on very specific training counts
- No concept of adjunction, productive processes, etc
Making the grammar more specific

Capture subcategorization effects by annotating heads and parent categories:

- \((S \texttt{(NP (DT the) (JJ recent) (NNS sale))})\) ...
- \((\texttt{NP^S-NNS (DT the) (JJ recent) (NN sale)})\)
- “Vertical Markovization”

\[
\begin{array}{c}
S \\
/ \swarrow \searrow \nwarrow \\
\text{NP^S-NNP} & \ldots & \\
/ \swarrow \searrow \nwarrow \\
\text{DT} & \text{JJ} & \text{NN} & \ldots \\
/ \swarrow \searrow \nwarrow \\
\text{the} & \text{recent} & \text{sale} & \\
\end{array}
\]

- \(\texttt{NP^S-NNP}\) (subject proper NP)
- vs \(\texttt{NP^PP-PRO}\) (pronoun in prep. phrase)
Making the grammar less specific

Use X-bar-like structure to allow adjunction:
- “Horizontal Markovization”

```
NP-NNS
 └── @N-+DT-NNS
      └── DT
           └── the
           └── @N-+JJ-NNS
                  └── JJ
                          └── recent

                  └── @N-NNS+PP
                                    └── PP-IN
                                           └── of shares

                  └── @N-NNS
                                    └── NNS
                                            └── of sale

                  └── @N-NNS
                                    └── sale
```

- Any @N symbol can produce adjuncts
- The “+CONTEXT” version represents modifier orders
Lexicalized grammars
Sometimes useful to incorporate words into the grammar rules directly:

- Verbs subcategorize for prepositions
- Collocations
- Crude semantics

For example, might have:

```
VP-VB-deliver
  ┌──────┐
  │ VB   │
  │ deliver │
  │ TO     │
  │ to     │
  │ the    │
  └──────┘
   NP-NN-store
```

Uses grammar rule VP-VB-deliver → VB PP-TO-to
- Checks if “deliver” takes “to” as preposition
Smoothing lexicalized parsers

Lexicalized parsers have very large grammars:
- Grammar is never written out explicitly
- Rules are made up when needed
- Probability of a rule is hard to estimate
  - Rule like “VP-deliver → VB PP-to” is rare

Typical approach: interpolated estimation (like LM)
- Use PCFG without words as backoff

\[ \hat{P}(VP-VB\text{-}deliver \rightarrow VB \ PP\text{-}TO\text{-}to) \propto \#(VP-VB\text{-}deliver \rightarrow VB \ PP\text{-}TO\text{-}to) + \beta P(VP \rightarrow VB \ PP) \]

Exact backoff strategy depends on which parser you look at
Discriminative methods

- Like an HMM, a PCFG is a generative model
- Obvious question: is there a discriminative (CRF-like) model?
- Early answer: yes, but it takes weeks to train
  - And the features we want to use often violate the Markov property
  - For instance, constructions
  - Checking agreement across multiple nodes
- ...in which case you can’t train at all!

Reranking

A combined discriminative/generative system:

- Print out the top 100 parses
- Use max-ent to pick the best one (using arbitrary features)
  - Not necessarily the one with highest original prob
- Max-ent features can violate the Markov property
- Because label set size reduced to 100
Parsing results

- Naive PCFG from Treebank: 70%
- Markovization, lexicalize a few prep/comp/conj: 87% (short ss)
  - Klein+Manning 2003
- Fully lexicalized CFG with backoff smoothing: 87-88% (short ss)
  - Charniak 1997, Collins 1999
- Reranking with large max-ent feature set: 90-91% (all ss)
  - Collins 2000, Charniak+Johnson 2005
- Further improvements to about 92.5%
  - Notably Petrov+Klein Berkeley parser, which we will discuss later
So far

- Recognizer/parser for PCFG via dynamic program
- Modified CFG is capable of parsing accurately
- Smoothing, reranking allow use of many feature types

Next time?

- Other grammar formalisms?
- Incremental algorithms (more cognitively plausible)