Statistical Natural Language Parsing

Gerald Penn
[based on slides by Christopher Manning]
Classical NLP Parsing

- Wrote symbolic grammar and lexicon
- $S \rightarrow NP \ VP$
- $NP \rightarrow (DT) \ NN$
- $NN \rightarrow \text{names}$
- $NP \rightarrow NN \ NN$
- $NN \rightarrow \text{Noun}$
- $VP \rightarrow \text{V} \ NP$
- $VBP \rightarrow \text{V} \ NP$
- Was hamstrung by the 1980s Zeitgeist of encoding this as a deductive proof search.
- Looking for all parses scaled badly and didn’t help coverage
  - Minimal grammar: “Fed raises” sentence: 36 parses
  - Simple 10 rule grammar: 592 parses
  - Real-size broad-coverage grammar: millions of parses

Classical NLP Parsing: The problem and its solution

- Very constrained grammars attempted to limit unlikely/weird parses for sentences.
- But the underlying method made that both difficult and a trade-off relative to coverage, i.e., some sentences can wind up with no parses.
- Solution: There needs to be an explicit mechanism that allows us to rank how likely each of the parses is:
  - Statistical parsing lets us work with very loose grammars that admit millions of parses for sentences but still quickly find the best parses.

The rise of annotated data: The Penn Treebank

```
(15)
(VP (PP (IN The) (NP (NN review)) (VP (VP (VBZ followed) (NP (DT a) (NN movie)) (VBV in) (RB (RBR after) (RB (RBR that) (NP (NN European) (JJ movie) (NN movie))))))))
```

The rise of annotated data

- Starting off, building a treebank seems a lot slower and less useful than building a grammar.
- But a treebank gives us many things
  - Reusability of the labor
  - Broad coverage (up to the corpus, at least)
  - “Analysis in context” - probably a better way to think about grammar anyway.
  - Frequencies and distributional information
  - A way to evaluate systems

Two views of linguistic structure: 1. Con constituency (phrase structure)

- Phrase structure organizes words into nested constituents:
- How do we know what is a constituent?
- Good question: It’s a pretty admixture of observed constraints on word order and semantic interpretability - we often have to ask linguists, and even they don’t always agree
- Distribution: a constituent behaves as a unit that can appear in different places:
  - John raised (to the children) [about apples]
  - John called (about apples) [to the children]
- Suffix taking shape to the children about
- Substitution: expansion in phrase forms
- I eat the (bowl of) soup (at the table)
- Coordination: regular internal structure, no intrusion, fragments, semantics, ...
Attachment ambiguities:
Two possible PP attachments

- The key parsing decision: How do we attach various kinds of constituents – PP, adverbial or participial phrases, coordinations, etc.
- Prepositional phrase attachment:
  - I saw the man with a telescope
  - What does with a telescope modify?
  - The verb saw?
  - The noun man?
  - Is the problem 'AI complete'? Yes, but...

Attachment ambiguities

- Proposed simple structural factors
  - Light association (Kimbol 1973) = 'low' or 'near' attachment = early closure of NP
  - Minimal attachment (Frazier 1978): Effects depend on grammar, but gave 'right' or 'focal' attachment = late closure of NP under the assumed model
- Which is right?
  - Such simple structural factors dominate in early psycholinguistics (and are still widely invoked)
  - In the VNP PP context, right attachment usually gets right 55–67% of cases.
  - But that means it gets wrong 33–45% of cases.

The importance of lexical factors

- Ford, Bresnan, and Kaplan (1982) [promoting 'lexicalist' linguistic theories] argued:
  - Order of grammatical rule processing (by a parser) determines closure effects
  - Ordering is jointly determined by strengths of alternative lexic features, strengths of alternative syntactic rewrite rules, and the sequences of hypotheses in the parsing process.
  - 'It is quite evident, then, that the closure effects in these sentences are induced in some way by the choice of the lexical item.' (Psycholinguistic studies show that this is true independent of discourse context.)

A simple prediction

- Use a likelihood ratio:
  - E.g., \[ \frac{P(vnp)}{P(vp)} \]
  - Plant(agreeing) = 0.15
  - Plant(agreeing) = 0.02
  - \[ \frac{P(vnp)}{P(vp)} \] (chose noun attachment)
A problematic example

- Chrysler confirmed that it would end its troubled venture with Maserati.
- Should be a noun attachment but get wrong answer:
  - $w \sim C(w', C(w, w))$
  - end 5156 607
  - venture 1442 155
  - $P(w) = 0.07, P(w) = 0.18, P(w) = 0.15, P(w) = 0.12$

What might be wrong here?

- If you see a V NP PP sequence, then for the PP to attach to the V, then it must also be the case that the NP doesn’t have a PP (or other postmodifier).
- Since, except in extrapolation cases, such dependencies can’t cross.
- Parsing allows us to factor in and integrate such constraints.

A better predictor would use $n_2$ as well as $v$, $n_1$, $p$

Attachment ambiguities in a real sentence

The board approved [its acquisition by Royal Trustco Ltd.]

- of Toronto
- 337 a share
- [at its monthly meeting]

What is parsing?

- We want to run a grammar backwards to find possible structures for a sentence.
- Parsing can be viewed as a search problem.
- Parsing is a hidden data problem.
- For the moment, we want to examine all structures for a string of words.
- We can do this bottom-up or top-down.
- This distinction is independent of depth-first or breadth-first search – we can do either both ways.
- We search by building a search tree which his distinct from the parse tree.

A phrase structure grammar

- $S \rightarrow NP \ VP$
- $VP \rightarrow V \ NP$
- $VP \rightarrow V \ NP \ PP$
- $NP \rightarrow NP \ PP$
- $NP \rightarrow N$
- $NP \rightarrow e$
- $PP \rightarrow P \ NP$
- $N \rightarrow cats$
- $N \rightarrow claws$
- $N \rightarrow people$
- $N \rightarrow scratch$
- $P \rightarrow with$
- By convention, $S$ is the start symbol, but in the PTB, we have an extra node at the top (ROOT, TOP)
Phrase structure grammars = context-free grammars

- $G = (T, N, S, R)$
- $T$ is set of terminals
- $N$ is set of nonterminals
  - For NLP, we usually distinguish a set $P \subset N$ of preterminals,
    which always rewrite as terminals.
- $S$ is the start symbol (one of the nonterminals)
- $R$ is a collection of productions of the form $X \rightarrow Y$, where $X$ is a
  nonterminal and $Y$ is a sequence of terminals and nonterminals
  (including an empty sequence).
- A grammar $G$ generates a language $L$.

Soundness and completeness

- A parser is sound if every parse it returns is valid/correct.
- A parser terminates if it is guaranteed to not go off
  into an infinite loop.
- A parser is complete if for any given grammar and
  sentence, it produces every valid parse for that
  sentence (on its way to infinity)? Can a sentence
  have infinitely many parses? Are there infinite-length
  sentences?
- For many purposes, we settle for sound but
  incomplete parsers: e.g., probabilistic parsers that
  return a k-best list.

Top-down parsing

- Top-down parsing is goal directed.
- A top-down parser starts with a list of goal
  constituents to be built. The top-down parser
  rewrites the goals in the goal list by matching one
  against the LHS of the grammar rules, and
  expanding it with the RHS, attempting to match the
  sentence to be derived.
- If a goal can be rewritten in several ways, then there
  is a choice of which rule to apply (search problem).
- Can use depth-first or breadth-first search, and goal
  ordering.

Problems with top-down parsing

- Left-recursive rules
- Will do badly if there are many rules for the same LHS.
  - Consider: $A \rightarrow AB$ and $A \rightarrow a$, $B \rightarrow AB$ and $B \rightarrow a$.
- In general, not goal-oriented enough: expands things that
  are possible top-down but not there in the original input.
- Top-down parsers do well if there is useful grammar.
- Driven control: search is directed by the grammar.
- Top-down is hopeless for rewriting parts of speech
  (preterminals) with words (terminals). In practice that is
  always done bottom-up as lexical lookup.
- Repeated words anywhere there is common
  substructure.

Bottom-up parsing

- Bottom-up parsing is data directed.
- The initial goal list of a bottom-up parser is the string to be
  parsed. If a sequence in the goal list matches the RHS of a
  rule, then this sequence may be replaced by the LHS of the
  rule.
- Parsing is finished when the goal list contains just the start
  category.
- If the RHS of several rules match the goal list, then there is a
  choice of which rule to apply (search problem).
- Can use depth-first or breadth-first search, and goal ordering.
- The standard presentation is as shift-reduce parsing.
Shift-reduce parsing: one path

What other search paths are there for parsing this sentence?

Problems with bottom-up parsing

- Empty categories: non-terminals with no words (terminals) under them. This can cause termination problems, unless rewriting empties as constituents is somehow restricted (but then it’s generally incomplete).
- Strictly local goal-orientation: locally possible, but globally impossible.
- Inefficient when there is a great deal of lexical ambiguity (grammar-driven control might help here).
- Conversely, it is data-directed: it attempts to parse the words that are there.
- Repeated work: anywhere there is common substructure

Quiz Question!

- How many distinct parses does the following sentence have due to PP attachment ambiguities?
  - A PP can attach to any preceding N or N within the verb phrase, subject only to the parse still being a tree.
  - (This is equivalent to there being no crossing dependencies, where if d_i is a dependent of d_j, and d_i is a dependent of d_j, then the lines d_i-d_j begin at d_i under the line from d_i to d_j.)

John wrote the book with a pen in the room.
  a) 2  b) 4  c) 6  d) 8

Repeated work...

Principles for success: take 1

- If you are going to do parsing-as-search with a grammar as is:
  - Evidence for left recursive structures must be found, not predicted.
  - Presence of empty categories must be predicted, not found.
- Doing these things helps with termination but doesn’t fix the repeated work problem:
  - Both TD (LL) and BU (LR) parsers can (and frequently do) do work exponentially in the sentence length on NLP problems.

Principles for success: take 2

- Grammar transformations can fix both left recursion and epsilon productions.
- Then you parse the same language but with different trees.
- Your linguist-friends will hate you:
  - But they don’t need to know – you can fix up the trees later so that no one notices what you did.
  - But the big problem is the global ambiguities – this creates no problems with termination, but can lead to exponentially many parses.
Principles for success: take 3

- Rather than doing parsing-as-enumeration, we do parsing as dynamic programming
- This is the most standard way to do things
  - E.g., CYK parsing
- It solves the problem of doing repeated work
- But there are also other ways of solving the problem of doing repeated work
  - E.g., doing graph-search rather than tree-search.

Human parsing

- Humans often do ambiguity maintenance
  - Have the police ... eat their supper?
  - Come in and look around.
  - Taken out and shot.
- But humans also commit early and are "garden pathed":
  - The man who hunts ducks out on weekends.
  - The cotton shirts are made from gowns in Mississippi.
  - The horse raced past the barn door.

Probabilistic or stochastic context-free grammars (PCFGs)

- \( G = (T, \mathcal{N}, S, R, P) \)
- \( T \) is set of terminals
- \( \mathcal{N} \) is set of nonterminals
  - For NLP, we usually distinguish out a set \( \mathcal{P} \) of preterminals, which always rewrite as terminals
  - \( S \) is the start symbol (one of the nonterminals)
  - \( R \) is a rule set, productions of the form \( X \rightarrow \gamma \), where \( X \) is a nonterminal, and \( \gamma \) is a sequence of terminals and nonterminals (possibly an empty sequence)
  - \( P(\cdot) \) gives the probability of each rule
    - For each non-terminal, \( \sum_{\gamma} P(\gamma | X) = 1 \)
  - A grammar \( G \) generates a language \( L = \{ w \mid \exists \gamma \in \mathcal{P}^* \text{ s.t. } G \vdash \gamma \Rightarrow w \} \)
  - \( \sum_{\gamma \in \mathcal{P}^*} P(\gamma | X) = 1 \)

PCFGs – Notation

- \( w_1 \ldots w_n = w \) is the word sequence from 1 to \( n \) (sentence of length \( n \))
- \( W_1 \ldots W_k = w \) is the subsequence \( w_1 \ldots w_k \)
- \( N \) is the nonterminal \( N \) dominating \( w_1 \ldots w_k \)

\[ \begin{array}{c}
  W_1 \\
  W_2 \\
  \ldots \\
  W_k \\
\end{array} \]

- We’ll write \( P(W \rightarrow \emptyset) \) to mean \( P(W \rightarrow X | N) \)
- We’ll want to calculate max \( P(t \Rightarrow w) \)

The probability of trees and strings

- \( P(t) \) – The probability of a tree is the product of the probabilities of the rules used to create it.
- \( P(w_{1:n}) \) – The probability of the string is the sum of the probabilities of the trees which have that string as their yield

\[ P(t) = \prod_{t_i \in \text{parse of } w_{1:n}} P(t_i) \]

A Simple PCFG (in CNF)

- \( S \rightarrow NP VP \) 1.0
  - \( S \rightarrow NP PP \) 0.4
  - \( S \rightarrow VP NP \) 0.3
  - \( S \rightarrow VP PP \) 0.3
- \( VP \rightarrow V NP \) 0.7
  - \( VP \rightarrow V PP \) 0.3
  - \( VP \rightarrow P NP \) 1.0
  - \( VP \rightarrow with PP \) 1.0
  - \( VP \rightarrow VPP \) 1.0
- \( PP \rightarrow P NP \) 1.0
  - \( PP \rightarrow V PP \) 0.9
  - \( PP \rightarrow PP PP \) 0.3
- \( P \rightarrow \emptyset \) 0.4
  - \( P \rightarrow stars \) 0.1
  - \( P \rightarrow telescope \) 0.1
- \( V \rightarrow stars \) 0.9
Tree and String Probabilities

- \( w_{10} = \) astronomers saw stars with ears
- \( P(t_{1}) = 1.0 \times 0.1 \times 0.7 \times 1.0 \times 0.4 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \)
  \( = 0.0009072 \)
- \( P(t_{2}) = 1.0 \times 0.1 \times 0.3 \times 0.7 \times 1.0 \times 0.18 \times 1.0 \times 1.0 \times 0.18 \)
  \( = 0.0006804 \)
- \( P(w_{10}) = P(t_{1}) + P(t_{2}) \)
  \( = 0.0009072 + 0.0006804 \)
  \( = 0.0015876 \)