CS 294-5: Statistical Natural Language Processing

Dan Klein
MF 4 230pm
Soda Hall 310

The Parsing Task

- Parsing marks how words are syntactically organized
- Decisions we make here affect how we process the text
- Big difference between natural languages and programming languages: rampant ambiguity

The Parsing Task

context

new art critics write reviews with computers

Context-Free Grammars

- A context free grammar is a tuple \(<N, T, S, R>\)
  - \(N\): the set of non-terminals
    - Phrasal categories: S, NP, VP, ADJP, etc.
    - Parts-of-speech (pre-terminals): NN, JJ, DT, VB
  - \(T\): the set of terminals (the words)
  - \(S\): the start symbol
    - Often written as ROOT or TOP
    - Not usually the sentence non-terminal \(S\)
  - \(R\): the set of rules
    - Of the form \(X \rightarrow Y_1 Y_2 \ldots Y_k\), with \(X, Y_i \in N\)
    - Examples: \(S \rightarrow NP VP\), \(VP \rightarrow VBP\)
      \(VP \rightarrow VBP NP\), \(VP \rightarrow VP PP\)
    - Also called rewrites, productions, or local trees

Example CFG

- Can just write the grammar (rules with non-terminal LHSs) and lexicon (rules with pre-terminal LHSs)

Grammar

- \(R\rightarrow S\)
- \(S \rightarrow NP VP\)
- \(VP \rightarrow VBP\)
- \(VP \rightarrow VP PP\)
- \(PP \rightarrow IN NP\)

Lexicon

- JJ \rightarrow new
- NN \rightarrow art
- NNS \rightarrow critics
- NNS \rightarrow reviews
- NNS \rightarrow computers
- VBP \rightarrow write
- IN \rightarrow with

Top-Down Generation from CFGs

- A CFG generates a language
- Fix an order: apply rules to leftmost non-terminal

Top-Down Generation from CFGs

- A CFG generates a language
- Fix an order: apply rules to leftmost non-terminal

Grammar

- \(R\rightarrow S\)
- \(S \rightarrow NP VP\)
- \(VP \rightarrow VBP\)
- \(VP \rightarrow VP PP\)
- \(PP \rightarrow IN NP\)

Lexicon

- JJ \rightarrow new
- NN \rightarrow art
- NNS \rightarrow critics
- NNS \rightarrow reviews
- NNS \rightarrow computers
- VBP \rightarrow write
- IN \rightarrow with

INPUT: critics write reviews

Parsing as Search: Top-Down

- Top down parsing: starts with the root and tries to generate the input
How Top-Down Fails

- Big problem 1: search isn’t guided by the input

**INPUT:** critics write reviews

- Big problem 2: separate ambiguities create redundant work

new art critics write reviews with computers

How Bottom-Up Fails

- Big Problem 1: ambiguities still create redundant work

new art critics write reviews with computers

- Little Problem: partial analyses which can’t be completed

art critics write reviews with art computers

Parsing as Search: Bottom-Up

- Bottom-up parsing: input drives the search (shift reduce parsing)

new art critics write reviews with computers

A Simple Chart Parser

- Chart parsers are sparse dynamic programs
- Ingredients:
  - Nodes: positions between words
  - Edges: spans of words with labels, represent the set of trees over those words rooted at x
  - A chart: records which edges we’ve built
  - An agenda: a holding pen for edges (a queue)
- We’re going to figure out:
  - What edges can we build?
  - All the ways we built them.

Word Edges

- An edge found for the first time is called discovered.
  - Edges go into the agenda on discovery.
- To initialize, we discover all word edges.

Unary Projection

- When we pop an word edge off the agenda, we check the lexicon to see what tag edges we can build from it
The “Fundamental Rule”

- When we pop edges off of the agenda:
  - Check for unary projections (NNS → critics, NP → NNS)
    \[ \text{Y}[i,j] \text{ with } X \rightarrow Y \text{ forms } X[i,j] \]
  - Combine with edges already in our chart (this is sometimes called the fundamental rule)
    \[ \text{Y}[i,j] \text{ and } Z[i,k] \text{ with } X \rightarrow Y Z \text{ form } X[i,k] \]
  - Enqueue resulting edges (if newly discovered)
  - Stick the popped edge in the chart
- Queries a chart must support:
  - Is edge X[i,j] in the chart?
  - What edges with label Y end at position j?
  - What edges with label Z start at position i?

Exploiting Substructure

- Each edge records all the ways it was built (locally)
  - Can recursively extract trees
  - A chart may represent too many parses to enumerate (how many?)

Order Independence

- A nice property:
  - It doesn’t matter what policy we use to order the agenda (FIFO, LIFO, random).
  - Why? Invariant: before popping an edge:
    - Any edge X[i,j] that can be directly built from chart edges and a single grammar rule is either in the chart or in the agenda.
    - Convince yourselves this invariant holds!
  - This will not be true once we get weighted parsers.

Empty Elements

- Sometimes we want to posit nodes in a parse tree that don’t contain any pronounced words:
  - I want John to parse this sentence
  - I want [ ] to parse this sentence
  - These are easy to add to our chart parser!
    - For each position i, add the “word” edge ε[i,i]
    - Add rules like NP → ε to the grammar
    - That’s it!

(Speech) Lattices

- There was nothing magical about words spanning exactly one position.
- When working with speech, we generally don’t know how many words there are, or where they break.
- We can represent the possibilities as a lattice and parse these just as easily! (cf. HW 1)
N-Ary Rules

- Often we want to write grammar rules like
  \[ VP \rightarrow VBD \ NP \ PP \ PP \]
  which are not binary.
- We can work with these rules by introducing new intermediate symbols into our grammar:
  \[
  [VP \rightarrow VBD \ NP \ PP] \\
  [VP \rightarrow VBD \ NP \ PP] \\
  VP \\
  VBD \\
  \]
- We’ll hear much more about this kind of thing when we talk about grammar representation later in the course.

Runtime: Theory

- How long does it take to parse a sentence?
  - Depends on:
    - Sentence length
    - Grammar size (and structure)
    - Specific input sentences
  - Asymptotically:
    - Do we do constant work per edge pop?
    - No, because one edge may combine with many others.
    - We do constant work per traversal (edge-edge combination).
    - How many traversals?
      - Form of traversal: \( Y[i,j] + Z[j,k] \) form Z[i,k].
      - So there are \( O(n^3) \) traversals – cubic time.

Runtime: Practice

- Let’s take the treebank grammar and go parsing!

Semantic Interpretation

- Back to meaning!
  - A very basic approach to computational semantics
  - Truth-theoretic notion of semantics (Tarskian)
  - Assign a “meaning” to each word
  - Word meanings combine according to the parse structure
  - People can and do spend entire courses on this topic
  - We’ll spend about 15 min!

- Supplemental reading will be on the web page (along with a supplemental syntax reading).

- Question for class: who would attend crash-course linguistics sections if I held them?

Types of Expressions

- Proper names:
  - Refer directly to some entity in the world
    - Have type “e”, or entity
    - Alex : alex
  - Sentences:
    - Are either true or false (given how the world actually is)
    - Have type “t” for “truth value”
    - Alex sings : sing(alex)
    - So what about verbs (and verb phrases)?
      - \( \lambda \)-calculus is a notation for functions whose arguments are not yet filled.
      - \( \lambda \)-calculus is a notation for functions whose arguments are not yet filled.
      - This is predicate – a function which takes an entity (type e) and produces a truth value (type t).
      - We can write its type as e → t.

Compositional Semantics

- So now we have meanings for the words
- How do we know how to combine words?
  - Associate a combination rule with each grammar rule:
    - \( S : \beta(x) \rightarrow NP : \alpha \rightarrow VP : \beta \) (function application)
    - \( VP : \lambda x . \alpha(x) \rightarrow \beta \rightarrow VP : \alpha \rightarrow VP : \beta \) (intersection)
  - Example:
    - sing(John) \rightarrow dances(John)
    - sing(John) \rightarrow dances(John)
    - sing(John) \rightarrow dances(John)
    - So there are \( O(n^3) \) traversals – cubic time.
Other Cases

- Transitive verbs:
  - likes : \( \lambda x. \lambda y. \text{likes}(y, x) \)
  - Two-place predicates of type \( e \rightarrow (e \rightarrow t) \):
  - \( \text{likes} : \lambda y. \text{likes}(y, \text{Amy}) \) is just like a one-place predicate.

- Quantifiers:
  - What does "Everyone" mean here?
    - Everyone : \( \lambda f. \forall x. f(x) \)
    - Mostly works, but some problems
      - Have to change our NP/VP rule.
      - Won’t work for "Amy likes everyone."
      - "Everyone like someone."
    - This gets tricky quickly!

Parsing with Semantics

- Let’s say we get that all sorted out…
- Can we augment our parser edges with meanings?
- Yes, but then edge labels declare the whole parse!

What have we accomplished?

- One big unified approach!
  - Sentences in, compact syntactico-semantic representations out
  - Can get parts-of-speech (for, e.g., text-to-speech)
  - Can get phrase structure (for, e.g., translation)
  - Can translate sentences into first-order logic statements (for, e.g., reasoning)

- The Achilles heel:
  - Still now way to decide which analysis is right!
  - Rest of the course focuses on this problem.

Trickier Stuff

- Non-Intersective Adjectives
  - green ball : \( \lambda x. (\text{green}(x) \land \text{ball}(x)) \)
  - fake diamond : \( \lambda x. (\text{fake}(x) \land \text{diamond}(x)) \)

- Generalized Quantifiers
  - the : \( \lambda f. \text{unique-member}(f) \) (1,4)

- Pronouns
- Plural NPs
- Tense
- Ellipsis
- Propositional Attitudes
- Questions!

- … the list goes on and on!

Parsing with Semantics

- Instead: augment non-word edges with types
- Assemble meanings the same way we assemble parses
- The double N’[1,4] goes away

Modeling Uncertainty

- Gapping hole warning!
- Big difference between the syntax and semantics models presented here.

- The scout saw the enemy soldiers with night goggles.
  - If we get good probabilistic tools, we’ll be able to say things like "72% belief that the PP attaches to the NP."
  - That means that probably the enemy has night vision goggles.
  - However, you can’t throw a logical assertion into a theorem prover with 72% confidence.
  - Not clear humans really extract and process logical statements symbolically anyway.

- In short, we need probabilistic reasoning, not just probabilistic disambiguation followed by symbol reasoning!
Parsing vs. Recognition

- **Parsers:** find the derivations of a sentence in a grammar
- **Recognizers:** tell whether a sentence can be generated
  - Need this to judge grammaticality
  - Only "I saw a van" and "I saw Ivan" are really grammatical

Who Cares about Recognition?

- **Analysis is only a part of NLP!**
  - In general we analyze (parse) input language
  - But we have to generate in the output language
  - **Translation:** Analyze some French, generate some English (or whatever source and target)
  - **Speech Recognition:** Analyze some acoustic data, generate some English
  - **Text Summarization:** Read some English, generate less English
  - Need to make sure the results are well formed
  - **Transduction = Transfer + Fluency**

How’s our Symbolic Parser at Recognition?

- **Using the treebank grammar:**
  - I saw a van
  - I saw of an
  - the the the of
  - In fact, it accepts $T^*$ (all terminal strings).
  - We’ll obviously have to raise our game here!

What’s Next?

- **Modeling uncertainty from now on!**
  - Probabilistic recognition: language modeling
  - Probabilistic parsing: disambiguation

- **Next class:** building Markov language models

- **Readings:** Skim M+S 3-4, J+M 9,15

- **Assignment 0:** Try it out – little work, lots of fun.