Recall: Dependencies

- Dependency syntax: syntactic structure is defined by dependencies
- Head (parent, governor) connected to dependent (child, modifier)
- Each word has exactly one parent except for the ROOT symbol
- Dependencies must form a directed acyclic graph

Recall: Shift-Reduce Parsing

- State: Stack: [ROOT I ate] Buffer: [some spaghetti bolognese]
- Left-arc (reduce operation): Let $\mathfrak{S}$ denote the stack
  - “Pop two elements, add an arc, put them back on the stack”
  $$\begin{align*}
  \mathfrak{S}[w_{-2}, w_{-1}] &\rightarrow \mathfrak{S}[w_{-1}], \quad w_{-2} \text{ is now a child of } w_{-1}
  \end{align*}$$
- Train a classifier to make these decisions sequentially — that classifier can parse sentences for you

Where are we now?

- Early in the class: bags of word (classifiers) => sequences of words (sequence modeling)
- Now we can understand sentences in terms of tree structures as well
- Why is this useful? What does this allow us to do?
- We’re going to see how parsing can be a stepping stone towards more formal representations of language meaning
Today

- Montague semantics:
  - Model theoretic semantics
  - Compositional semantics with first-order logic
- CCG parsing for database queries
- Lambda-DCS for question answering

Model Theoretic Semantics

- Key idea: can ground out natural language expressions in set-theoretic expressions called *models* of those sentences
  - Natural language statement $S \Rightarrow$ interpretation of $S$ that models it
    - *She likes going to that restaurant*
      - Interpretation: defines who *she* and *that restaurant* are, make it able to be concretely evaluated with respect to a *world*
      - Entailment (statement A implies statement B) reduces to: in all worlds where A is true, B is true
  - Our modeling language is *first-order logic*

Model Theoretic Semantics

First-order Logic

- Powerful logic formalism including things like entities, relations, and quantifications
  - *Lady Gaga sings*
  - *sings* is a *predicate* (with one argument), function $f$: entity $\rightarrow$ true/false
  - *sings(Lady Gaga) = true* or false, have to execute this against some database (*world*)
  - $[[\text{sings}]] = \text{denotation}$, set of entities which sing (found by executing this predicate on the world — we’ll come back to this)
Quantification

- Universal quantification: “for all” operator
  - $\forall x \text{sings}(x) \lor \text{dances}(x) \rightarrow \text{performs}(x)$
  - “Everyone who sings or dances performs”
- Existential quantification: “there exists” operator
  - $\exists x \text{sings}(x)$
  - “Someone sings”
- Source of ambiguity! “Everyone is friends with someone”
  - $\forall x \exists y \text{friend}(x,y)$
  - $\exists y \forall x \text{friend}(x,y)$

Logic in NLP

- Question answering:
  - *Who are all the American singers named Amy?*
    - $\lambda x. \text{nationality}(x, \text{USA}) \land \text{sings}(x) \land \text{firstName}(x, \text{Amy})$
  - Function that maps from $x$ to true/false, like `filter`. Execute this on the world to answer the question
  - Lambda calculus: powerful system for expressing these functions
- Information extraction: *Lady Gaga and Eminem are both musicians*
  - musician(Lady Gaga) $\land$ musician(Eminem)
  - Can now do reasoning. Maybe know: $\forall x \text{musician}(x) \Rightarrow \text{performer}(x)$
  - Then: performer(Lady Gaga) $\land$ performer(Eminem)

Compositional Semantics with First-Order Logic

Montague Semantics

- Database containing entities, predicates, etc.
- Sentence expresses something about the world which is either true or false
- Denotation: evaluation of some expression against this database
- $[[\text{Lady Gaga}]] = e470$
  - $[[\text{sings}(e470)]] = \text{True}$
  - denotation of this string is an entity
  - denotation of this expression is T/F
Montague Semantics

\[ \text{sings}(e470) \]

- Function application: apply this to e470

\[ \text{e470} \rightarrow \text{S} \rightarrow \lambda y. \text{sings}(y) \]

\[ \text{NNP} \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \lambda y. \text{sings}(y) \]

- Takes one argument (y, the entity) and returns a logical form \( \text{sings}(y) \)

- We can use the syntactic parse as a bridge to the lambda-calculus representation, build up a logical form (our model) compositionally

Parses to Logical Forms

\[ \text{sings}(e470) \land \text{dances}(e470) \]

\[ \text{e470} \rightarrow \text{S} \rightarrow \lambda y. \text{sings}(y) \land \text{dances}(y) \]

\[ \text{NNP} \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \lambda y. \text{sings}(y) \land \text{dances}(y) \]

- General rules:
  - VP: \( \lambda y. a(y) \land b(y) \rightarrow VP: \lambda y. a(y) \) CC VP: \( \lambda y. b(y) \)
  - S: \( f(x) \rightarrow NP: x \) VP: \( f \)

Parses to Logical Forms

\[ \text{born}(e470, 3/28/1986) \]

\[ \text{e470} \rightarrow \text{S} \rightarrow \lambda y. \text{born}(y, 3/28/1986) \]

\[ \text{NNP} \rightarrow \text{NP} \rightarrow \text{VP} \rightarrow \lambda y. \text{born}(y, 3/28/1986) \]

- Function takes two arguments: first x (date), then y (entity)
- How to handle tense: should we indicate that this happened in the past?

Tricky things

- Adverbs/temporality: Lady Gaga sang well yesterday
  \[ \text{sings}(\text{Lady Gaga}, \text{time=yesterday}, \text{manner=well}) \]
- "Neo-Davidsonian" view of events: things with many properties:
  \[ \exists e. \text{type}(e, \text{sing}) \land \text{agent}(e,e470) \land \text{manner}(e, \text{well}) \land \text{time}(e, \_ ) \]
- Quantification: Everyone is friends with someone
  \[ \exists y \forall x \text{friend}(x,y) \land \forall x \exists y \text{friend}(x,y) \]
  (one friend) (different friends)
- Same syntactic parse for both! So syntax doesn't resolve all ambiguities
- Indefinite: Amy ate a waffle
  \[ \exists w. \text{waffle}(w) \land \text{ate}(\text{Amy},w) \]
- Generic: Cats eat mice (all cats eat mice? most cats? some cats?)
Semantic Parsing

- For question answering, syntactic parsing doesn’t tell you everything you want to know, but indicates the right structure
- Solution: *semantic parsing*: many forms of this task depending on semantic formalisms
- Two today: CCG (looks like what we’ve been doing) and lambda-DCS
- Applications: database querying/question answer: produce lambda-calculus expressions that can be executed in these contexts

CCG Parsing

Combinatory Categorial Grammar

- Steedman+Szabolcsi (1980s): formalism bridging syntax and semantics
- Parallel derivations of syntactic parse and lambda calculus expression
- Syntactic categories (for this lecture): $S$, $NP$, “slash” categories
- $S\backslash NP$: “if I combine with an NP on my left side, I form a sentence” — verb
- $S\backslash NP$/NP: “I need an NP on my right and then on my left” — verb with a direct object
- When you apply this, there has to be a parallel instance of function application on the semantics side

Combinatory Categorial Grammar

- Steedman+Szabolcsi 1980s: formalism bridging syntax and semantics
- Syntactic categories (for this lecture): $S$, $NP$, “slash” categories
- $S\backslash NP$: “if I combine with an NP on my left side, I form a sentence” — verb
- $S\backslash NP$/NP: “I need an NP on my right and then on my left” — verb with a direct object
"What" is a **very** complex type: needs a noun and needs a \( S \backslash NP \) to form a sentence. \( S \backslash NP \) is basically a verb phrase (\textit{border Texas})

Lexicon is highly ambiguous — all the challenge of CCG parsing is in picking the right lexicon entries

---

Many ways to build these parsers

One approach: run a “supertagger” (tags the sentence with complex labels), then run the parser

```
\begin{align*}
\frac{S/(S/NP)/N}{\lambda f. \lambda g. \lambda x. f(x) \land g(x)} & \quad \text{What} \\
\frac{\lambda x. state(x)}{N} & \quad \text{states} \\
\frac{\lambda x. \lambda y. borders(y, x)}{NP} & \quad \text{border} \\
\frac{\lambda y. borders(y, \text{texas})}{NP} & \quad \text{Texas} \\
\end{align*}
```

Parsing is easy once you have the tags, so we’ve reduced it to a (hard) tagging problem
Building CCG Parsers

- Model: log-linear model over derivations with features on rules:
  \[ P(d|x) \propto \exp w^T \left( \sum_{r \in d} f(r, x) \right) \]
- Training data looks like pairs of sentences and logical forms
  \( \text{What states border Texas} \quad \lambda x. \text{state}(x) \land \text{borders}(x, e89) \)
- Problem: we don’t know the derivation
  - \( \text{Texas} \) corresponds to \( NP \mid e89 \) in the logical form (easy to figure out)
  - \( What \) corresponds to \( (S/(S/NP))/NP \mid \lambda f. \lambda g. \lambda x. \ f(x) \land g(x) \)
  - How do we infer that without being told it?

Lexicon

- GENLEX: takes sentence \( S \) and logical form \( L \). Break up logical form into chunks \( C(L) \), assume any substring of \( S \) might map to any chunk
  \( \text{What states border Texas} \quad \lambda x. \text{state}(x) \land \text{borders}(x, e89) \)
- Chunks inferred from the logic form based on rules:
  - \( NP: e89 \quad (S/NP)/NP: \lambda x. \lambda y. \text{borders}(x, y) \)
- Any substring can parse to any of these in the lexicon
  - \( \text{Texas} \rightarrow NP: e89 \) is correct
  - \( \text{border Texas} \rightarrow NP: e89 \)
  - \( \text{What states border Texas} \rightarrow NP: e89 \)
  - ... Zettlemoyer and Collins (2005)

GENLEX

<table>
<thead>
<tr>
<th>Input Trigger</th>
<th>Rules</th>
<th>Output Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NP} \mid c )</td>
<td>( NP \mid \text{c} )</td>
<td>( NP \mid \text{c} )</td>
</tr>
<tr>
<td>( \text{arity one predicate} p_1 )</td>
<td>( S(NP)/NP: \lambda x. p_1(x) )</td>
<td>( S(NP)/NP: \lambda x. \text{state}(x) )</td>
</tr>
<tr>
<td>( \text{arity one predicate} p_2 )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. p_2(x, y) )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. \text{borders}(x, y) )</td>
</tr>
<tr>
<td>( \text{arity two predicate} p_3 )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. p_3(x, y) )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. \text{borders}(x, y) )</td>
</tr>
<tr>
<td>( \text{arity two predicate} p_4 )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. p_4(x, y) )</td>
<td>( (S/NP)/NP: \lambda x. \lambda y. \text{borders}(x, y) )</td>
</tr>
<tr>
<td>( \text{arity zero sum with shared argument} )</td>
<td>( S/NP: \lambda x. f(x) )</td>
<td>( S/NP: \lambda x. \text{state}(x) )</td>
</tr>
</tbody>
</table>

- Very complex and hand-engineered way of taking lambda calculus expressions and “back-solving” for the derivation
  Zettlemoyer and Collins (2005)
Learning

- Iterative procedure like the EM algorithm: estimate “best” parses that derive each logical form, retrain the parser using these parses with supervised learning
- We’ll talk about a simpler form of this in a few slides

Applications

- GeoQuery: answering questions about states (~80% accuracy)
- Jobs: answering questions about job postings (~80% accuracy)
- ATIS: flight search
- Can do well on all of these tasks if you handcraft systems and use plenty of training data: these domains aren’t that rich
- What about broader QA?

Lambda-DCS

- Dependency-based compositional semantics — original version was less powerful than lambda calculus, lambda-DCS is as powerful
- Designed in the context of building a QA system from Freebase
- Freebase: set of entities and relations
  - Alice Smith
  - Bob Cooper
  - Seattle
  - Washington
  - March 15, 1961
- [[PlaceOfBirth]] = set of pairs of (person, location)

Zettlemoyer and Collins (2005)
Liang et al. (2011), Liang (2013)
Lambda-DCS

Liang et al. (2011), Liang (2013)

\[ \lambda x. x = \text{Seattle} \]
\[ \lambda x. \lambda y. \text{PlaceOfBirth}(x, y) \]
\[ \lambda x. \text{PlaceOfBirth}(x, \text{Seattle}) \]

- Looks like a tree fragment over Freebase

?? \[ \text{PlaceOfBirth} \text{ Seattle} \]

\[ \lambda x. \text{Profession}(x, \text{Scientist}) \wedge \text{PlaceOfBirth}(x, \text{Seattle}) \]

"list of scientists born in Seattle"

- Execute this fragment against Freebase, returns Alice Smith (and others)

Liang et al. (2011), Liang (2013)

Parsing into Lambda-DCS

- Derivation \( d \) on sentence \( x \):
  - No more explicit syntax in these derivations like we had in CCG

- Building the lexicon: more sophisticated process than GENLEX, but can handle thousands of predicates

- Log-linear model with features on rules:
  \[ P(d | x) \propto \exp w^T \left( \sum_{r \in d} f(r, x) \right) \]

- Similar to CRF parsers

Berant et al. (2013)

Parsing with Lambda-DCS

- Learn just from question-answer pairs: maximize the likelihood of the right denotation \( y \) with the derivation \( d \) marginalized out

\[ \mathcal{O}(\theta) = \sum_{i=1}^{n} \log \sum_{d \in D(x); [d, z]_K = y_i} p_d(d | x_i). \]

For each example:

- Run beam search to get a set of derivations
- Let \( d^* \) = highest-scoring derivation in the beam
- Let \( d^* \) = highest-scoring derivation in the beam with correct denotation
- Do a structured perceptron update towards \( d^* \) away from \( d \)

Berant et al. (2013)
Takeaways

- Can represent meaning with first order logic and lambda calculus
- Can bridge syntax and semantics and create semantic parsers that can interpret language into lambda-calculus expressions
- Useful for querying databases, question answering, etc.
- Next time: neural net methods for doing this that rely less on having explicit grammars