CS388: Natural Language Processing
Lecture 13: Semantics I

Greg Durrett

Slides adapted from Dan Klein, UC Berkeley
Mini 2 due *today* at 5pm
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

I ate some spaghetti bolognese

- **State:** Stack: [ROOT I ate]  Buffer: [some spaghetti bolognese]

- **Left-arc (reduce operation):** Let $\sigma$ denote the stack
  - “Pop two elements, add an arc, put them back on the stack”
    - $\sigma|w_{-2}, w_{-1} \rightarrow \sigma|w_{-1}$, $w_{-2}$ is now a child of $w_{-1}$

- **State:** Stack: [ROOT ate]  Buffer: [some spaghetti bolognese]
Where are we now?

- Early in the class: sentences are just sequences of words

- Now we can understand them 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

- First-order logic
- Compositional semantics with first-order logic
- CCG parsing for database queries
- Lambda-DCS for question answering
First-Order Logic
First-order Logic

- Powerful logic formalism including things like entities, relations, and quantifications

- Propositions: let a = *It is day*, b = *It is night*
  - a ∨ b = either a is true or b is true, a => ¬b = a implies not b

- More complex statements: “*Lady Gaga sings*

- sings is a *predicate* (with one argument), function f: entity => true/false

- sings(Lady Gaga) = true or false, have to execute this against some database (called a *world*)

- [[sings]] = *denotation*, set of entities which sing (sort of like 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 \text{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

  \( \text{musician}(\text{Lady Gaga}) \land \text{musician}(\text{Eminem}) \)

- Can now do reasoning. Maybe know: \( \forall x \text{ musician}(x) \Rightarrow \text{performer}(x) \)

  Then: \( \text{performer}(\text{Lady Gaga}) \land \text{performer}(\text{Eminem}) \)
Compositional Semantics with First-Order Logic
Truth-Conditional Semantics

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>Alias</th>
<th>Birthdate</th>
<th>Sings?</th>
</tr>
</thead>
<tbody>
<tr>
<td>e470</td>
<td>Stefani Germanotta</td>
<td>Lady Gaga</td>
<td>3/28/1986</td>
<td>T</td>
</tr>
<tr>
<td>e728</td>
<td>Marshall Mathers</td>
<td>Eminem</td>
<td>10/17/1972</td>
<td>T</td>
</tr>
</tbody>
</table>

- Database containing entities, predicates, etc.

- Truth-conditional semantics: sentence expresses something about the world which is either true or false

- Denotation: evaluation of some expression against this database

- $[[Lady Gaga]] = e470$
  - denotation of this string is an entity

- $[[\text{sings}(e470)]] = \text{True}$
  - denotation of this expression is T/F
We can use the syntactic parse as a bridge to the lambda-calculus representation, build up a logical form compositionally.
Parses to Logical Forms

\[
sings(e_{470}) \land dances(e_{470})
\]

\[
S \rightarrow e_{470} NP \rightarrow NNP NNP \rightarrow \text{Lady Gaga} VP \rightarrow VBP VBP \rightarrow \text{sings} \land \text{dances} \rightarrow \lambda y. \text{sings}(y) \land \text{dances}(y)
\]

› General rules:
  \[
  \text{VP: } \lambda y. \ a(y) \land b(y) \rightarrow \text{VP: } \lambda y. \ a(y) \text{ CC VP: } \lambda y. \ b(y)
  \]
  \[
  \text{S: } f(x) \rightarrow \text{NP: } x \text{ VP: } f
  \]
Parses to Logical Forms

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

- 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*
  
sings(Lady Gaga, time=yesterday, manner=well)

- “Neo-Davidsonian” view of events: things with many properties:
  
  \[ \exists e. \text{type}(e, \text{sing}) \land \text{agent}(e, e470) \land \text{manner}(e, \text{well}) \land \text{time}(e, \ldots) \]

- Quantification: *Everyone is friends with someone*
  
  \[ \exists y \ \forall x \ \text{friend}(x, y) \ \ \forall x \ \exists y \ \text{friend}(x, y) \]
  
  (one friend) (different friends)

- Same syntactic parse for both! So syntax doesn't resolve all ambiguities

- Indefinite: *Amy ate a waffle*
  
  \[ \exists w. \ \text{waffle}(w) \land \text{ate}(Amy, w) \]

- Generic: *Cats eat mice* (all cats eat mice? most cats? some cats?)
QA from Parsing

\[ \lambda x. \text{born}(e470, x) \]

- Execute this function against a knowledge base to answer the question

- Tricky to parse due to wh-movement...would be easier if we said

  Lady Gaga was born when
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.
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\NP: “if I combine with an NP on my left side, I form a sentence” — verb
- 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 (border Texas)

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

Zettlemoyer and Collins (2005)
CCG Parsing

<table>
<thead>
<tr>
<th>Show me</th>
<th>flights</th>
<th>to</th>
<th>Prague</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/N</td>
<td>N</td>
<td>(N\N) / NP</td>
<td>NP</td>
</tr>
<tr>
<td>(\lambda f. f)</td>
<td>(\lambda x. \text{flight}(x))</td>
<td>(\lambda y. \lambda f. \lambda x. f(y) \land \text{to}(x, y))</td>
<td>NP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(N\N) (\lambda f. \lambda x. f(x) \land \text{to}(x, \text{PRG}))</td>
<td>PRG</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(N) (\lambda x. \text{flight}(x) \land \text{to}(x, \text{PRG}))</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S) (\lambda x. \text{flight}(x) \land \text{to}(x, \text{PRG}))</td>
<td></td>
</tr>
</tbody>
</table>

- “to” needs an NP (destination) and N (parent)
Building CCG Parsers

- Model: log-linear model over derivations with features on rules:

\[ P(d|x) \propto \exp w^\top \left( \sum_{r \in d} f(r, x) \right) \]

- Can parse with a variant of CKY

Eminem sings

Zettlemoyer and Collins (2005)
Building CCG Parsers

- Training data looks like pairs of sentences and logical forms

What states border Texas $\lambda x. \text{state}(x) \land \text{borders}(x, e89)$

- Problem: we don’t know the derivation

  - Texas corresponds to NP | e89 in the logical form (easy to figure out)
  - What corresponds to $(S/(S\backslash NP))/N | \lambda f.\lambda g.\lambda x. f(x) \land g(x)$
  - How do we infer that without being told it?

Zettlemoyer and Collins (2005)
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

$What \text{ states border Texas} \quad \lambda x. \text{ state}(x) \land \text{borders}(x, \text{e89})$

- Chunks inferred from the logic form based on rules:
  - NP: e89
  - $\langle \text{S}\backslash\text{NP} \rangle/\text{NP}: \lambda x. \lambda y. \text{borders}(x, y)$

- Any substring can parse to any of these in the lexicon
  - $Texas \rightarrow \text{NP: e89 is correct}$
  - $\text{border Texas} \rightarrow \text{NP: e89}$
  - $What \text{ states border Texas} \rightarrow \text{NP: e89}$

  ...  

Zeitelmoyer and Collins (2005)
<table>
<thead>
<tr>
<th>Input Trigger</th>
<th>Rules</th>
<th>Output Category</th>
<th>Categories produced from logical form</th>
</tr>
</thead>
<tbody>
<tr>
<td>constant c</td>
<td>$NP : c$</td>
<td></td>
<td>$NP : \text{texas}$</td>
</tr>
<tr>
<td>arity one predicate $p_1$</td>
<td>$N : \lambda x. p_1(x)$</td>
<td></td>
<td>$N : \lambda x. \text{state}(x)$</td>
</tr>
<tr>
<td>arity one predicate $p_1$</td>
<td>$S \setminus NP : \lambda x. p_1(x)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(S \setminus NP) / NP : \lambda x. \lambda y. p_2(y, x)$</td>
<td>$N : \lambda x. \text{state}(x)$</td>
<td></td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(S \setminus NP) / NP : \lambda x. \lambda y. p_2(x, y)$</td>
<td>$N : \lambda x. \text{state}(x)$</td>
<td></td>
</tr>
<tr>
<td>literal with arity two predicate $p_2$ and constant second argument $c$</td>
<td>$N / N : \lambda g. \lambda x. p_2(x, c) \wedge g(x)$</td>
<td></td>
<td>$N / N : \lambda g. \lambda x. \text{state}(x) \wedge g(x)$</td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(N \setminus N) / NP : \lambda x. \lambda y. p_2(x, y) \wedge g(x)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>an arg max / min with second argument arity one function $f$</td>
<td>$NP / N : \lambda g. \text{arg max} / \text{min}(g, \lambda x. f(x))$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>an arity one numeric-ranged function $f$</td>
<td>$S / NP : \lambda x. f(x)$</td>
<td></td>
<td>$S / NP : \lambda x. \text{size}(x)$</td>
</tr>
</tbody>
</table>

- Very complex and hand-engineered way of taking lambda calculus expressions and “backsolving” for the derivation

Zettlemoyer and Collins (2005)
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

Zettlemoyer and Collins (2005)
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
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

- [[PlaceOfBirth]] = set of pairs of (person, location)

Liang et al. (2011), Liang (2013)
Lambda-DCS

Lambda-DCS

Seattle

PlaceOfBirth

PlaceOfBirth.Seattle

Lambda calculus

\[ \lambda x. \ x = \text{Seattle} \]

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

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

Looks like a tree fragment over Freebase

???

— PlaceOfBirth – Seattle

Profession.Scientist \land

PlaceOfBirth.Seattle

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

\land \text{PlaceOfBirth}(x,\text{Seattle})

\[ \text{Liang et al. (2011), Liang (2013)} \]
Lambda-DCS

March 15, 1961

DateOfBirth

Alice Smith

PlaceOfBirth

Seattle

Scientist

Profession

Professor

PlaceOfBirth

Seattle

CapitalOf

Washington

Bob Cooper

PlaceOfBirth

Seattle

Professor

“list of scientists born in Seattle”

Profession.Scientist ∧ PlaceOfBirth.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 \( x \):
  
  \[
P(d|x) \propto \exp w^\top \left( \sum_{r \in d} f(r, x) \right)
\]

- 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:

- Similar to CRF parsers

Berant et al. (2013)
Learn just from question-answer pairs: maximize the likelihood of the right denotation $y$ with the derivation $d$ marginalized out

$$
\mathcal{L}(\theta) = \sum_{i=1}^{n} \log \sum_{d \in D(x) : [d.z]_K = y_i} p_\theta(d \mid x_i).
$$

For each example:

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

Berant et al. (2013)
Learning

- Each vertical slice is the beam for one example. Green = correct denotation

- Only a small number of questions are even reachable by beam search initially (but some questions are very easy so even a totally untrained model can answer them)

- During training, more and more “good” derivations surface and will result in model updates

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