Natural Language Processing

Compositional Semantics

Dan Klein – UC Berkeley
Truth-Conditional Semantics
Truth-Conditional Semantics

- Linguistic expressions:
  - “Bob sings”

- Logical translations:
  - sings(bob)
  - Could be p_1218(e_397)

- Denotation:
  - [[bob]] = some specific person (in some context)
  - [[sings(bob)]] = ???

- Types on translations:
  - bob : e (for entity)
  - sings(bob) : t (for truth-value)
Truth-Conditional Semantics

- **Proper names:**
  - Refer directly to some entity in the world
  - Bob : \texttt{bob} \quad \llbracket \texttt{bob} \rrbracket^W \rightarrow ???

- **Sentences:**
  - Are either true or false (given how the world actually is)
  - Bob sings : \texttt{sings(bob)}

- **So what about verbs (and verb phrases)?**
  - \texttt{sings} must combine with \texttt{bob} to produce \texttt{sings(bob)}
  - The $\lambda$-calculus is a notation for functions whose arguments are not yet filled.
  - \texttt{sings} : $\lambda x.\texttt{sings(x)}$
  - This is a *predicate* – a function which takes an entity (type e) and produces a truth value (type t).
    - We can write its type as e$\rightarrow$t.
  - Adjectives?
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:
  - $S : \beta(\alpha) \rightarrow NP : \alpha$  $VP : \beta$  (function application)
  - $VP : \lambda x. \alpha(x) \wedge \beta(x) \rightarrow VP : \alpha$  and  $\emptyset$  $VP : \beta$  (intersection)
- Example:
What do we do with logical translations?

- Translation language (logical form) has fewer ambiguities
- Can check truth value against a database
  - Denotation (“evaluation”) calculated using the database
- More usefully: assert truth and modify a database, either explicitly or implicitly
  - eg prove a consequence from asserted axioms
- Questions: check whether a statement in a corpus entails the (question, answer) pair:
  - “Bob sings and dances” → “Who sings?” + “Bob”
- Chain together facts and use them for comprehension
Other Cases

- **Transitive verbs:**
  - likes : $\lambda x.\lambda y.\text{likes}(y,x)$
  - Two-place predicates of type $e \to (e \to t)$.
  - likes Amy : $\lambda y.\text{likes}(y,Amy)$ 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 likes someone.”
  - This gets tricky quickly!
Indefinites

- First try
  - “Bob ate a waffle” : ate(bob, waffle)
  - “Amy ate a waffle” : ate(amy, waffle)

- Can’t be right!
  - $\exists x : \text{waffle}(x) \land \text{ate}(bob, x)$
  - What does the translation of “a” have to be?
  - What about “the”?
  - What about “every”?
Grounding

- **Grounding**
  - So why does the translation $\text{likes} : \lambda x. \lambda y. \text{likes}(y, x)$ have anything to do with actual liking?
  - It doesn’t (unless the denotation model says so)
  - Sometimes that’s enough: wire up *bought* to the appropriate entry in a database

- **Meaning postulates**
  - Insist, e.g. $\forall x, y. \text{likes}(y, x) \rightarrow \text{knows}(y, x)$
  - This gets into lexical semantics issues

- **Statistical / neural version?**
Tense and Events

- In general, you don’t get far with verbs as predicates
- Better to have event variables e
  - “Alice danced” : danced(alice)
  - $\exists e : dance(e) \land agent(e,alice) \land (time(e) < now)$
- Event variables let you talk about non-trivial tense / aspect structures
  - “Alice had been dancing when Bob sneezed”
  - $\exists e, e' : dance(e) \land agent(e,alice) \land$ 
    sneeze(e’) $\land agent(e’,bob) \land$
    (start(e) < start(e’) $\land$ end(e) = end(e’)) $\land$
    (time(e’) < now)
- Minimal recursion semantics, cf “object oriented” thinking
Adverbs

- What about adverbs?
  - “Bob sings terribly”
  - terribly(sings(bob))?
  - (terribly(sings))(bob)?
  - $\exists e \text{ present}(e) \land \text{type}(e, \text{singing}) \land \text{agent}(e, \text{bob}) \land \text{manner}(e, \text{terrible})$?
  - Gets complex quickly...
Propositional Attitudes

- “Bob thinks that I am a gummi bear”
  - \( \text{thinks}(\text{bob}, \text{gummi}(\text{me})) \) ?
  - \( \text{thinks}(\text{bob}, \text{“I am a gummi bear”}) \) ?
  - \( \text{thinks}(\text{bob}, \text{^gummi}(\text{me})) \) ?

- Usual solution involves intensions (\(^X\)) which are, roughly, the set of possible worlds (or conditions) in which \( X \) is true

- Hard to deal with computationally
  - Modeling other agents’ models, etc
  - Can come up in even simple dialog scenarios, e.g., if you want to talk about what your bill claims you bought vs. what you actually bought
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: $\lambda f. [\text{unique-member}(f)]$
  - all: $\lambda f. \lambda g \forall x. f(x) \rightarrow g(x)$
  - most?
  - Could do with more general second order predicates, too (why worse?)
    - the(cat, meows), all(cat, meows)

- **Generics**
  - “Cats like naps”
  - “The players scored a goal”

- **Pronouns (and bound anaphora)**
  - “If you have a dime, put it in the meter.”

- ... the list goes on and on!
Scope Ambiguities

- **Quantifier scope**
  - “All majors take a data science class”
  - “Someone took each of the electives”
  - “Everyone didn’t hand in their exam”

- **Deciding between readings**
  - Multiple ways to work this out
    - Make it syntactic (movement)
    - Make it lexical (type-shifting)
Add a “sem” feature to each context-free rule

- \( S \rightarrow NP \) loves \( NP \)
- \( S[sem=loves(x,y)] \rightarrow NP[sem=x] \) loves \( NP[sem=y] \)
- Meaning of \( S \) depends on meaning of \( NPs \)

TAG version:

- Template filling: \( S[sem=showflights(x,y)] \rightarrow I \) want a flight from \( NP[sem=x] \) to \( NP[sem=y] \)
Logical Form Translation
The task:

Input: List one way flights to Prague.
Output: $\lambda x. \text{flight}(x) \land \text{one\_way}(x) \land \text{to}(x, \text{PRG})$

Challenging learning problem:

- Derivations (or parses) are not annotated
- Approach: [Zettlemoyer & Collins 2005]
- Learn a lexicon and parameters for a weighted Combinatory Categorial Grammar (CCG)
Background

- Combinatory Categorial Grammar (CCG)
- Weighted CCGs
- Learning lexical entries: GENLEX
CCG Parsing

- **Combinatory Categorial Grammar**
  - Fully (mono-) lexicalized grammar
  - Categories encode argument sequences
  - Very closely related to the lambda calculus
  - Can have spurious ambiguities (why?)

```
John ⊢ NP : john'
shares ⊢ NP : shares'
buys ⊢ (S\NP)/NP : λx.λy.buys'xy
sleeps ⊢ S\NP : λx.sleeps'x
well ⊢ (S\NP)\(S\NP) : λf.λx.well'(fx)
```
<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>flights</td>
<td>N : ( \lambda x.\text{flight}(x) )</td>
</tr>
<tr>
<td>to</td>
<td>((\text{N/N})/\text{NP} : \lambda x.\lambda f.\lambda y.f(x) \land \text{to}(y,x))</td>
</tr>
<tr>
<td>Prague</td>
<td>NP : PRG</td>
</tr>
<tr>
<td>New York city</td>
<td>NP : NYC</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Parsing Rules (Combinators)

Application
- $X/Y : f$ $Y : a$ $\Rightarrow$ $X : f(a)$
- $Y : a$ $X\backslash Y : f$ $\Rightarrow$ $X : f(a)$

Composition
- $X/Y : f$ $Y/Z : g$ $\Rightarrow$ $X/Z : \lambda x. f(g(x))$
- $Y\backslash Z : f$ $X\backslash Y : g$ $\Rightarrow$ $X\backslash Z : \lambda x. f(g(x))$

Additional rules:
- Type Raising
- Crossed Composition
CCG Parsing

Show me  flights  to  Prague

S/N  λf.f  N  λx.flight(x)  (N\N)/NP  λy.λf.λx.f(y)∧to(x,y)  NP  PRG

N\N  λf.λx.f(x)∧to(x,PRG)

N  λx.flight(x)∧to(x,PRG)

S  λx.flight(x)∧to(x,PRG)
Given a log-linear model with a CCG lexicon \( \Lambda \), a feature vector \( f \), and weights \( w \).

- The best parse is:

\[
y^* = \arg\max_y w \cdot f(x, y)
\]

Where we consider all possible parses \( y \) for the sentence \( x \) given the lexicon \( \Lambda \).
**Lexical Generation**

**Input Training Example**

| Sentence: | Show me flights to Prague. |
| Logic Form: | $\lambda x.\text{flight}(x) \land \text{to}(x, PRG)$ |

**Output Lexicon**

<table>
<thead>
<tr>
<th>Words</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me</td>
<td>$S/N : \lambda f.f$</td>
</tr>
<tr>
<td>flights</td>
<td>$N : \lambda x.\text{flight}(x)$</td>
</tr>
<tr>
<td>to</td>
<td>$(N\backslash N)/NP : \lambda x.\lambda f.\lambda y.\text{flight}(x) \land \text{to}(y, x)$</td>
</tr>
<tr>
<td>Prague</td>
<td>$NP : PRG$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>
### Input Training Example

<table>
<thead>
<tr>
<th>Sentence</th>
<th>Logic Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me flights to Prague.</td>
<td>$\lambda x. \text{flight}(x) \land \text{to}(x, \text{PRG})$</td>
</tr>
</tbody>
</table>

### Output Lexicon

<table>
<thead>
<tr>
<th>All possible substrings:</th>
<th>Categories created by rules that trigger on the logical form:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me flights</td>
<td>NP : PRG</td>
</tr>
<tr>
<td>...</td>
<td>N : $\lambda x. \text{flight}(x)$</td>
</tr>
<tr>
<td>Show me flights to</td>
<td>(S\NP)/NP : $\lambda x. \lambda y. \text{to}(y, x)$</td>
</tr>
<tr>
<td>...</td>
<td>(N\N)/NP : $\lambda y. \lambda f. \lambda x. \ldots$</td>
</tr>
</tbody>
</table>

[Zettlemoyer & Collins 2005]
Robustness

The lexical entries that work for:

Show me the latest flight from Boston to Prague on Friday

Will not parse:

Boston to Prague the latest on Friday
Relaxed Parsing Rules

Two changes

- Add application and composition rules that relax word order
- Add type shifting rules to recover missing words

These rules significantly relax the grammar

- Introduce features to count the number of times each new rule is used in a parse
Review: Application

\[
\begin{align*}
  X/Y : f & \quad Y : a \quad \Rightarrow \quad X : f(a) \\
  Y : a & \quad X\backslash Y : f \quad \Rightarrow \quad X : f(a)
\end{align*}
\]
Disharmonic Application

- Reverse the direction of the principal category:

  \[ X \setminus Y : f \quad Y : a \quad \Rightarrow \quad X : f(a) \]
  \[ Y : a \quad X / Y : f \quad \Rightarrow \quad X : f(a) \]

<table>
<thead>
<tr>
<th>flights</th>
<th>one way</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N )</td>
<td>( N/N )</td>
</tr>
<tr>
<td>( \lambda x.\text{flight}(x) )</td>
<td>( \lambda f.\lambda x.f(x) \wedge \text{one_way}(x) )</td>
</tr>
</tbody>
</table>

\( N \)

\( \lambda x.\text{flight}(x) \wedge \text{one\_way}(x) \)
Insert missing semantic content

- NP : c  =>  N\N : \( \lambda f. \lambda x. f(x) \land p(x, c) \)

<table>
<thead>
<tr>
<th>flights</th>
<th>Boston</th>
<th>to Prague</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>NP</td>
<td>N\N</td>
</tr>
<tr>
<td>( \lambda x. \text{flight}(x) )</td>
<td>BOS</td>
<td>( \lambda f. \lambda x. f(x) \land \text{to}(x, \text{PRG}) )</td>
</tr>
<tr>
<td>N\N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lambda f. \lambda x. f(x) \land \text{from}(x, \text{BOS}) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lambda x. \text{flight}(x) \land \text{from}(x, \text{BOS}) )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \lambda x. \text{flight}(x) \land \text{from}(x, \text{BOS}) \land \text{to}(x, \text{PRG}) )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bypass missing nouns

- \( N \setminus N : f \rightarrow N : f(\lambda x. \text{true}) \)

\[
\begin{align*}
\text{Northwest Air} & \quad \text{to Prague} \\
\text{N/N} & \quad \text{N/N} \\
\lambda f. \lambda x. f(x) \land \text{airline}(x, \text{NWA}) & \quad \lambda f. \lambda x. f(x) \land \text{to}(x, \text{PRG}) \\
\lambda x. \text{to}(x, \text{PRG}) & \\
\lambda x. \text{airline}(x, \text{NWA}) \land \text{to}(x, \text{PRG}) &
\end{align*}
\]
**Inputs:** Training set \( \{(x_i, z_i) \mid i=1\ldots n\} \) of sentences and logical forms. Initial lexicon \( \Lambda \). Initial parameters \( w \). Number of iterations \( T \).

**Training:** For \( t = 1\ldots T, i =1\ldots n \):

**Step 1: Check Correctness**
- Let \( y^* = \arg\max_y w \cdot f(x_i, y) \)
- If \( L(y^*) = z_i \), go to the next example

**Step 2: Lexical Generation**
- Set \( \lambda = \Lambda \cup \text{GENLEX}(x_i, z_i) \)
- Let \( \hat{y} = \arg\max_{y \text{ s.t. } L(y)=z_i} w \cdot f(x_i, y) \)
- Define \( \lambda_i \) to be the lexical entries in \( y^\hat{ } \)
- Set lexicon to \( \Lambda = \Lambda \cup \lambda_i \)

**Step 3: Update Parameters**
- Let \( y' = \arg\max_y w \cdot f(x_i, y) \)
- If \( L(y') \neq z_i \)
  - Set \( w = w + f(x_i, \hat{y}) - f(x_i, y') \)

**Output:** Lexicon \( \Lambda \) and parameters \( w \).
Neural Encoder-Decoder Approaches
Encoder-Decoder Models

- Can view many tasks as mapping from an input sequence of tokens to an output sequence of tokens

- Semantic parsing:
  
  $\text{What states border Texas} \rightarrow \lambda x \text{state}(x) \land \text{borders}(x, e89)$

- Syntactic parsing
  
  $\text{The dog ran} \rightarrow (S (NP (DT the) (NN dog)) (VP (VBD ran)))$

  (but what if we produce an invalid tree or one with different words?) 😐

- Machine translation, summarization, dialogue can all be viewed in this framework as well — our examples will be MT for now

Next slides from Greg Durrett
Semantic Parsing as Translation

- Prolog
- Lambda calculus
- Other DSLs

**GEO**

$x$: “what is the population of iowa ?”

$y$: 

```prolog
_\_answer ( NV , ( 
  _\_population ( NV , V1 ) , _\_const ( 
    V0 , _\_stateid ( iowa ) ) ) ) 
```

**ATIS**

$x$: “can you list all flights from chicago to milwaukee”

$y$: 

```prolog
( _\_lambda $0 e ( _\_and 
  ( _\_flight $0 ) 
  ( _\_from $0 chicago : _\_ci ) 
  ( _\_to $0 milwaukee : _\_ci ) ) )
```

**Overnight**

$x$: “when is the weekly standup”

$y$: 

```prolog
( call listValue ( call 
  getProperty meeting.weekly_standup 
  ( string start_time ) ) )
```
Semantic Parsing as Seq2Seq

“what states border Texas”

\[
\text{lambda } x \ ( \ \text{state}( \ x \ ) \ \text{and} \ \text{border}( \ x, \ e89 \ ) \ )
\]

- Write down a linearized form of the semantic parse, train seq2seq models to directly translate into this representation.

- What are some benefits of this approach compared to grammar-based?

- What might be some concerns about this approach? How do we mitigate them?

Jia and Liang (2016)
Problem: Lack of Inductive Bias

“what states border Texas”  “what states border Ohio”

- Parsing-based approaches handle these the same way
  - Possible divergences: features, different weights in the lexicon
- Can we get seq2seq semantic parsers to handle these the same way?
- Key idea: don’t change the model, change the data
- “Data augmentation”: encode invariances by automatically generating new training examples
Possible Solution: Data Augmentation

Examples
("what states border texas ?",
answer(NV, (state(V0), next_to(V0, NV), const(V0, stateid(texas)))))

Rules created by AbsEntities
ROOT → {"what states border STATEID ?",
  answer(NV, (state(V0), next_to(V0, NV), const(V0, stateid(STATEID ))))}
STATEID → {"texas", texas}
STATEID → {"ohio", ohio}

- Lets us synthesize a "what states border ohio ?" example
- Abstract out entities: now we can “remix” examples and encode invariance to entity ID. More complicated remixes too
Possible Solution: Copying

<table>
<thead>
<tr>
<th></th>
<th>GEO</th>
<th>ATIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Copying</td>
<td>74.6</td>
<td>69.9</td>
</tr>
<tr>
<td>With Copying</td>
<td>85.0</td>
<td>76.3</td>
</tr>
</tbody>
</table>

- For semantic parsing, copying tokens from the input (texas) can be very useful.
- Copying typically helps a bit, but attention captures most of the benefit. However, vocabulary expansion is critical for some tasks (machine translation).

Jia and Liang (2016)
Mapping to Programs

show me the fare from ci0 to ci1

\[
\text{lambda } \$0 \ e
\quad (\text{exists } \$1 \ (\text{and} (\text{from } \$1 \ \text{ci0} )
\quad (\text{to } \$1 \ \text{ci1} )
\quad (\text{= (fare } \$1 \ ) \ \$0 ) ) )
\]

```
name: [ 'D', 'i', 'r', 'e', 'h', 'A', 'l', 'p', 'h', 'a' ]
cost: [2]
type: ['Minion']
rarity: ['Common']
race: ['beast']
class: ['Neutral']
description: [ 'Adjacent', 'minions', 'have', '+', '1', 'Attack', 'hr' ]
health: [2]
attack: [2]
durability: [-1]
```

class DireWolfAlpha(MinionCard):
    def __init__(self):
        super().__init__(
            "Dire Wolf Alpha", 2, CHARACTER_CLASS.ALL,
            CARD_RARITY.COMMON, minion_type=MIGNION_TYPE.BEAST)
    def create_minion(self, player):
        return Minion(2, 2, auras=[
            Aura(ChangeAttack(1), MinionSelector(Adjacent()))
        ])
Structured Models

- Meaning representations (e.g., Python) have strong underlying syntax
- How can we explicitly model the underlying syntax/grammar of the target meaning representations in the decoding process?

Python Abstract Grammar

```
expr → Name | Call
Call → expr[func] expr*[args] keyword*[keywords]
   If → expr[test] stmt*[body] stmt*[orelse]
For → expr[target] expr*[iter] stmt*[body] stmt*[orelse]
FunctionDef → identifier[name] expr*[iter]
   stmt*[body] stmt*[orelse]
```

Abstract Syntax Tree

```
Sorted
```

Next section includes slides from Yin / Neubig
Abstract Syntax Trees

Input Intent ($x$)  
*sort my_list in descending order*

Generated AST ($y$)

Surface Code ($c$)  
*sorted(my_list, reverse=True)*
AST-Structured Neural Modules

[Image of a diagram showing various statements and conditions in an AST structure, with annotations and labels such as "stmt", "ClassDef", "If", "For", "While", "Assign", "Return", "expr", "stmt*", "test", "body", "orelse", "identifier", "__init__", "create_minion", "add_buff", "change_attack", "damage", "..."

[Rabinovich, Stern, Klein, 2017]
AST-Structured Fragments
Example Results Across Tasks

<table>
<thead>
<tr>
<th>ATIS</th>
<th>Accuracy</th>
<th>GEO</th>
<th>Accuracy</th>
<th>JOBS</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>ZH15</td>
<td>84.2</td>
<td>ZH15</td>
<td>88.9</td>
<td>ZH15</td>
<td>85.0</td>
</tr>
<tr>
<td>ZC07</td>
<td>84.6</td>
<td>KCAZ13</td>
<td>89.0</td>
<td>PEK03</td>
<td>88.0</td>
</tr>
<tr>
<td>WKZ14</td>
<td><strong>91.3</strong></td>
<td>WKZ14</td>
<td><strong>90.4</strong></td>
<td>LJK13</td>
<td>90.7</td>
</tr>
<tr>
<td>DL16</td>
<td>84.6</td>
<td>DL16</td>
<td>87.1</td>
<td>DL16</td>
<td>90.0</td>
</tr>
<tr>
<td>ASN</td>
<td>85.3</td>
<td>ASN</td>
<td>85.7</td>
<td>ASN</td>
<td><strong>91.4</strong></td>
</tr>
<tr>
<td>+ SUPATT</td>
<td>85.9</td>
<td>+ SUPATT</td>
<td>87.1</td>
<td>+ SUPATT</td>
<td><strong>92.9</strong></td>
</tr>
</tbody>
</table>

[Rabinovich, Stern, Klein, 2017]
## Copying / Pointer Networks

### Intent: join app_config.path and string 'locale' into a file path, substitute it for localedir.

**Pred.**

```python
localedir = os.path.join(app_config.path, 'locale')
```

### Intent: `self.plural` is a lambda function with an argument `n`, which returns the result of boolean expression `n` not equal to integer 1.

**Pred.**

```python
self.plural = lambda n: len(n)
```

**Ref.**

```python
self.plural = lambda n: int(n!=1)
```

### Intent: `name` Burly Rockjaw Trogg `<cost> 5 </cost> `<attack> 3 </attack> `<defense> 5 </defense> `<desc> Whenever your opponent casts a spell, gain 2 Attack. `<rarity> Common `<rarity> ...

**Ref.**

```python
class BurlyRockjawTrogg(MinionCard):
    def __init__(self):
        super().__init__('Burly Rockjaw Trogg', 4, CHARACTER_CLASS.ALL, CARD_RARITY.COMMON)
    def create minion(self, player):
        return Minion(3, 5, effects=[Effect(SpellCast(player=EnemyPlayer()),
                                         ActionTag(Give(ChangeAttack(2)), SelfSelector()))])
```