Natural Language Processing

Berkeley N L P

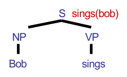
Compositional Semantics

Dan Klein – UC Berkeley

Truth-Conditional Semantics

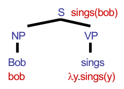
Truth-Conditional Semantics

- 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)



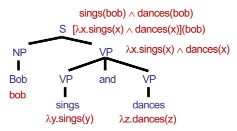
Truth-Conditional Semantics

- Proper names:
  - Refer directly to some entity in the world
  - Bob : bob [[bob]]^W → ???
- Sentences:
  - Are either true or false (given how the world actually is)
  - Bob sings : sings(bob)
- So what about verbs (and verb phrases)?
  - sings must combine with bob to produce sings(bob)
  - The λ-calculus is a notation for functions whose arguments are not yet filled.
  - sings : λx.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→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 : β(α) → NP : α VP : β (function application)
  - VP : λx . α(x) ∧ β(x) → VP : α and : ∅ VP : β (intersection)
- Example:

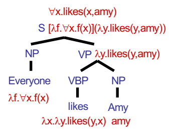


Denotation

- 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
  - Or the opposite: 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

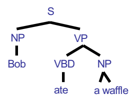
Other Cases

- Transitive verbs:
  - likes : λx.λy.likes(y,x)
  - Two-place predicates of type e→(e→t).
  - likes Amy : λy.likes(y,Amy) is just like a one-place predicate.
- Quantifiers:
  - What does "Everyone" mean here?
  - Everyone : λf.∀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!
  - ∃ x : waffle(x) ∧ ate(bob,x)
  - What does the translation of "a" have to be?
  - What about "the"?
  - What about "every"?



Grounding

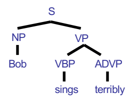
- Grounding
  - So why does the translation likes : λx.λy.likes(y,x) have anything to do with actual liking?
  - It doesn't (unless the denotation model says so)
  - Sometimes that's enough: wire up bought to the appropriate entry in a database
- Meaning postulates
  - Insist, e.g ∀x,y.likes(y,x) → knows(y,x)
  - 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)
  - ∃ e : dance(e) ∧ agent(e,alice) ∧ (time(e) < now)
- Event variables let you talk about non-trivial tense / aspect structures
  - "Alice had been dancing when Bob sneezed"
  - ∃ e, e' : dance(e) ∧ agent(e,alice) ∧
    sneeze(e') ∧ agent(e',bob) ∧
    (start(e) < start(e') ∧ end(e) = end(e')) ∧
    (time(e') < now)
- Minimal recursion semantics, cf "object oriented" thinking

Adverbs

- What about adverbs?
  - "Bob sings terribly"
  - terribly(sings(bob))?
  - (terribly(sings))(bob)?
  - ∃e present(e) ∧ type(e, singing) ∧ agent(e,bob) ∧ manner(e, terrible) ?
  - Gets complex quickly…



Propositional Attitudes

- "Bob thinks that I am a gummi bear"
  - thinks(bob, gummi(me)) ?
  - thinks(bob, "I am a gummi bear") ?
  - thinks(bob, ^gummi(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

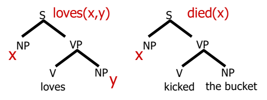
- Non-Intersective Adjectives
  - green ball : λx.[green(x) ∧ ball(x)]
  - fake diamond : λx.[fake(x) ∧ diamond(x)] ? ➡ λx.[fake(diamond(x))
- Generalized Quantifiers
  - the : λf.[*unique-member*(f)]
  - all : λf. λg [∀x.f(x) → g(x)]
  - most?
  - Could do with more general second order predicates, 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)

Classic Implementation, TAG, Idioms

- Add a "sem" feature to each context-free rule
  - S → NP loves NP
  - S[sem=loves(x,y)] → NP[sem=x] loves NP[sem=y]
  - Meaning of S depends on meaning of NPs
- TAG version:



- Template filling: S[sem=showflights(x,y)] →
  I want a flight from NP[sem=x] to NP[sem=y]

Logical Form Translation

Mapping to LF: Zettlemoyer & Collins 05/07

The task:

```
Input:  List one way flights to Prague.
Output: λx.flight(x)∧ one_way(x)∧ to(x,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)

[Slides from Luke Zettlemoyer]

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?)

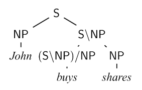
$John \vdash \mathrm{NP} : john'$

$shares \vdash \mathrm{NP} : shares'$

$buys \vdash (\mathrm{S}\backslash\mathrm{NP})/\mathrm{NP} : \lambda x.\lambda y.buys'xy$

$sleeps \vdash \mathrm{S}\backslash\mathrm{NP} : \lambda x.sleeps'x$

$well \vdash (\mathrm{S}\backslash\mathrm{NP})\backslash(\mathrm{S}\backslash\mathrm{NP}) : \lambda f.\lambda x.well'(fx)$



CCG Lexicon

Words	Category
flights	N : λx.flight(x)
to	(N\N)/NP : λx.λf.λy.f(x) ∧ to(y,x)
Prague	NP : PRG
New York city	NP : NYC
…	…

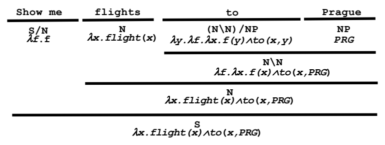
Parsing Rules (Combinators)

Application
- `X/Y : f      Y : a  =>   X : f(a)`
- `  Y : a    X\Y : f  =>   X : f(a)`

Composition
- `X/Y : f   Y/Z : g   =>  X/Z : λx.f(g(x))`
- `Y\Z : f   X\Y : g   =>  X\Z : λx.f(g(x))`

Additional rules:
- Type Raising
- Crossed Composition

CCG Parsing



Weighted CCG

Given a log-linear model with a CCG lexicon Λ, a feature vector f, and weights w.
- The best parse is:

$$y^* = \underset{y}{\operatorname{argmax}}\ w \cdot f(x,y)$$

Where we consider all possible parses y for the sentence x given the lexicon Λ.

Lexical Generation

Input Training Example

Sentence:	Show me flights to Prague.
Logic Form:	λx.flight(x) ∧ to(x,PRG)

Output Lexicon

Words	Category
Show me	S/N : λf.f
flights	N : λx.flight(x)
to	(N\N)/NP : λx.λf.λy.f(x) ∧ to(y,x)
Prague	NP : PRG
...	...

GENLEX: Substrings X Categories

Input Training Example

Sentence:	`Show me flights to Prague.`
Logic Form:	$\lambda x.flight(x) \wedge to(x,PRG)$

Output Lexicon

All possible substrings:

```
Show
me
flights
...
Show me
Show me flights
Show me flights to
...
```

X

Categories created by rules that trigger on the logical form:

```
       NP : PRG
        N : λx.flight(x)
(S\NP)/NP : λx.λy.to(y,x)
(N\N)/NP  : λy.λf.λx. …
        ...
```

[Zettlemoyer & Collins 2005]

Robustness

The lexical entries that work for:

```
Show me the latest flight from Boston to Prague on Friday
 S/NP    NP/N        N         N\N        N\N       N\N
  …       …          …          …          …         …
```

Will not parse:

```
Boston to Prague the latest on Friday
  NP     N\N         NP/N       N\N
  …      …            …          …
```

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

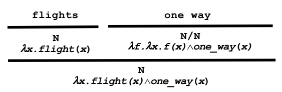
Review: Application

```
X/Y : f     Y : a   =>  X : f(a)
Y : a     X\Y : f   =>  X : f(a)
```

Disharmonic Application

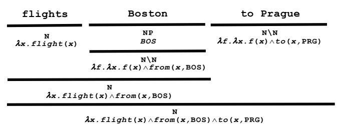
- Reverse the direction of the principal category:

```
X\Y : f     Y : a   =>  X : f(a)
Y : a     X/Y : f   =>  X : f(a)
```



Missing content words

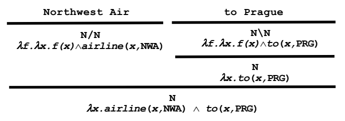
Insert missing semantic content
- `NP : c  =>  N\N : λf.λx.f(x) ∧ p(x,c)`



Missing content-free words

Bypass missing nouns

- `N\N : f =>  N : f(λx.true)`



```
Inputs: Training set {(xi, zi) | i=1...n} of sentences and logical forms.  Initial
    lexicon Λ.  Initial parameters w.  Number of iterations T.
Training: For t = 1...T, i =1...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 $y$^
        • 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 Λ 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:

What states border Texas ⟶ `λ x state( x ) ∧ borders( x , e89 )`

- Syntactic parsing

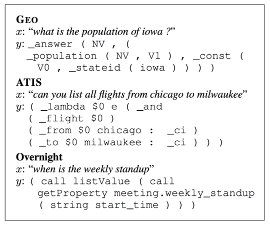
The dog ran ⟶ `(S (NP (DT the) (NN dog) ) (VP (VBD ran) ) )`

(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



GEO
x: "*what is the population of iowa ?*"
```
y: _answer ( NV , (
  _population ( NV , V1 ) , _const (
    V0 , _stateid ( iowa ) ) ) ) )
```
ATIS
x: "*can you list all flights from chicago to milwaukee*"
```
y: ( _lambda $0 e ( _and
  ( _flight $0 )
  ( _from $0 chicago :  _ci )
  ( _to $0 milwaukee :  _ci ) ) )
```
Overnight
x: "*when is the weekly standup*"
```
y: ( call listValue ( call
    getProperty meeting.weekly_standup
    ( string start_time ) ) )
```

- Prolog
- Lambda calculus
- Other DSLs

Semantic Parsing as Seq2Seq

"what states border Texas"
↓
`lambda x ( state( x ) and 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

Jia and Liang (2016)

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

	GEO	ATIS
No Copying	74.6	69.9
With Copying	85.0	76.3

- 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

lambda $0 e
  ( exists $1 ( and ( from $1 ci0 )
                    ( to $1 ci1 )
                    ( = ( fare $1 ) $0 ) ) )
```

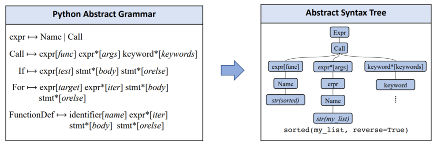
```
name: [
  'D', 'i', 'r', 'e', ' ',
  'W', 'o', 'l', 'f', ' ',
  'A', 'l', 'p', 'h', 'a']
cost: ['2']
type: ['Minion']
rarity: ['Common']
race: ['Beast']
class: ['Neutral']
description: [
  'Adjacent', 'minions', 'have',
  '+', '1', 'Attack', '.']
health: ['2']
attack: ['2']
durability: ['-1']
```

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

[Rabinovich, Stern, Klein, 2017]

Structured Models

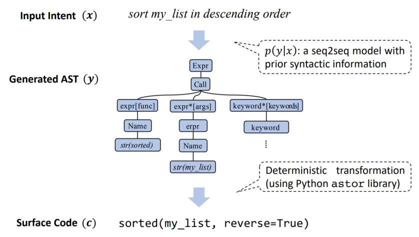
- Meaning representations (e.g., Python) have strong underlying syntax
- How can we **explicitly** model the underlying syntax/grammar of the target meaning representations in the decoding process?


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*]

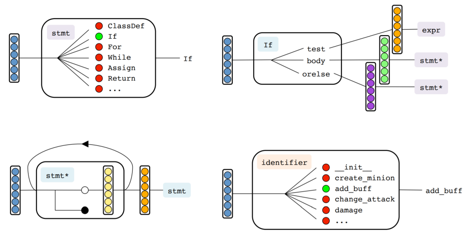
Next section includes slides from Yin / Neubig

Abstract Syntax Trees



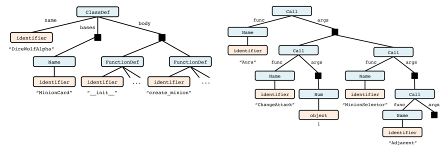
Input Intent (x)
sort my_list in descending order
$p(y|x)$: a seq2seq model with prior syntactic information
Generated AST (y)
Expr
Call
expr[func]
expr*[args]
keyword*[keywords]
Name
erpr
keyword
str(sorted)
Name
str(my_list)
Deterministic transformation (using Python astor library)
Surface Code (c)
sorted(my_list, reverse=True)

AST-Structured Neural Modules



[Rabinovich, Stern, Klein, 2017]

AST-Structured Fragments



Example Results Across Tasks

ATIS		GEO		JOBS	
System	Accuracy	System	Accuracy	System	Accuracy
ZH15	84.2	ZH15	88.9	ZH15	85.0
ZC07	84.6	KCAZ13	89.0	PEK03	88.0
WKZ14	**91.3**	WKZ14	**90.4**	LJK13	90.7
DL16	84.6	DL16	87.1	DL16	90.0
ASN	85.3	ASN	85.7	ASN	**91.4**
+ SUPATT	85.9	+ SUPATT	87.1	+ SUPATT	**92.9**

[Rabinovich, Stern, Klein, 2017]

Copying / Pointer Networks

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

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

Intent *self.plural is an lambda function with an argument n, which returns result of boolean expression n not equal to integer 1*

Pred.
```
self.plural = lambda n: len(n) ✗
```

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

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

Ref.
```
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()))]) ✓
```