Learning to Map Sentences to Logical Form

Luke Zettlemoyer

joint work with Michael Collins
MIT Computer Science and Artificial Intelligence Lab

Mapping Text to Meaning

Natural Language (NL) $\rightarrow$ $\boxed{M}$ $\rightarrow$ Meaning Representation (MR)

Input: (text strings)

- Natural language text

Output: (formal meaning representation)

- A representation of the underlying meaning of the input text

Computation: (an algorithm M)

- Recovers the meaning of the input text

A Challenging Problem

Building the mapping M, in the most general form, requires solving natural language understanding.

There are restricted domains that are still challenging:

- Natural language interfaces to databases
- Dialogue systems

Learning The Mapping

Why learn:

- Difficult to build by hand
- Learned solutions are potentially more robust

We consider a supervised learning problem:

- Given a training set: $\{(NL_i, MR_i) \mid i=1...n\}$
- Find the mapping M that best fits the training set
- Evaluate on unseen test set

The Setup for This Talk

NL: A single sentence

- usually a question

MR: A lambda-calculus expression

- similar to meaning representations used in formal semantics classes in linguistics

M: Weighted combinatory categorical grammar (CCG)

- mildly context-sensitive formalism
- explains a wide range of linguistic phenomena: coordination, long distance dependencies, etc.
- models syntax and semantics
- statistical parsing algorithms exist

A Simple Training Example

Given training examples like:

Input: `What states border Texas?`

Output: $\lambda x.state(x) \wedge borders(x, texas)$

MR: Lambda calculus

- Can think of as first-order logic with functions
- Useful for defining the semantics of questions

Challenge for learning:

- Derivations (parses) are not in training set
- We need to recover this missing information

More Training Examples

Input: `What is the largest state?`

Output: $argmax(\lambda x.state(x), \lambda x.size(x))$

Input: `What states border the largest state?`

Output: $\lambda x.state(x) \wedge borders(x, argmax(\lambda y.state(y), \lambda y.size(y)))$

Input: `What states border states that border states ... that border Texas?`

Output: $\lambda x.state(x) \wedge \exists y.state(y) \wedge \exists z.state(z) \wedge \ldots \wedge borders(x,y) \wedge borders(y,z) \wedge borders(z,texas)$

Outline

- ➤ Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences

CCG

[Steedman 96,00]

Lexicon

- Pairs natural language phrases with syntactic and semantic information
- Relatively complex: contains almost all information used during parsing

Parsing Rules (Combinators)

- Small set of relatively simple rules
- Build parse trees bottom-up
- Construct syntax and semantics in parallel

CCG Lexicon

Words	Category: Syntax : Semantics
Texas	NP : $texas$
Kansas	NP : $kansas$
borders	(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$
state	N : $\lambda x.state(x)$
Kansas City	NP : $kansas\_city\_MO$
...	...

Parsing: Lexical Lookup

What	states	border	Texas
S/(S\NP)/N $\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$	N $\lambda x.state(x)$	(S\NP)/NP $\lambda x.\lambda y.borders(y,x)$	NP $texas$

Parsing Rules (Combinators)

Application

- X/Y : f   Y : a   =>   X : f(a)

  (S\NP)/NP $\lambda x.\lambda y.borders(y,x)$ NP $texas$ S\NP $\lambda y.borders(y,texas)$

- Y : a   X\Y : f   =>   X : f(a)

  NP $kansas$ S\NP $\lambda y.borders(y,texas)$ S $borders(kansas,texas)$

Parsing a Question

What	states	border	Texas
S/(S\NP)/N	N	(S\NP)/NP	NP
$\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$	$\lambda x.state(x)$	$\lambda x.\lambda y.borders(y,x)$	$texas$

S/(S\NP)
$\lambda g.\lambda x.state(x) \wedge g(x)$

S\NP
$\lambda y.borders(y,texas)$

S
$\lambda x.state(x) \wedge borders(x,texas)$

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

Other Combinators

- Type Raising
- Crossed Composition

Features, Weights and Scores

x= What states border Texas

y=

What	states	border	Texas
S/(S\NP)/N	N	(S\NP)/NP	NP
$\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$	$\lambda x.state(x)$	$\lambda x.\lambda y.borders(y,x)$	$texas$

S/(S\NP)
$\lambda g.\lambda x.state(x) \wedge g(x)$

S\NP
$\lambda y.borders(y,texas)$

S
$\lambda x.state(x) \wedge borders(x,texas)$

Lexical count features:

$$f(x,y) = [\,0,\ 1,\ 0,\ 1, 1,\ 0,\ 0,\ 1, \ldots, 0]$$

$$w = [-2, 0.1, 0, 2, 1, -3, 0, 0.3, \ldots, 0]$$

$$w \cdot f(x,y) = 3.4$$

Weighted CCG

Weighted linear model (Λ, f, w) :

- CCG lexicon: Λ
- Feature function: $f(x,y) \in \mathbb{R}^m$
- Weights: $w \in \mathbb{R}^m$

Quality of a parse y for sentence x

- Score: $w \cdot f(x,y)$

Weighted CCG Parsing

Two computations: sentence x, parses y, LF z

- Best parse

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

- Best parse with logical form z

$$\hat{y} = \arg \max_{y \ s.t. \ L(y)=z} w \cdot f(x,y)$$

We use a CKY-style dynamic-programming algorithm with pruning

Outline

- Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences

A Supervised Learning Approach

Given a training set: $\{(x_i, z_i) \mid i=1...n\}$

- x_i : a natural language sentence
- z_i : a lambda-calculus expression

Find a weighted CCG that minimizes error

- induce a lexicon Λ
- estimate weights w

Evaluate on unseen test set

Learning: Two Parts

- GENLEX subprocedure
  - Create an overly general lexicon
- A full learning algorithm
  - Prunes the lexicon and estimates parameters w

Lexical Generation

Input Training Example

Sentence: `Texas borders Kansas`
Logic Form: $borders(texas,kansas)$

Output Lexicon

Words	Category
Texas	NP : $texas$
borders	(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$
Kansas	NP : $kansas$
...	...

GENLEX

- Input: a training example (x_i,z_i)
- Computation:
  1. Create all substrings of words in x_i
  2. Create categories from logical form z_i
  3. Create lexical entries that are the cross product of these two sets
- Output: Lexicon Λ

Step 1: GENLEX Words

Input Sentence:

`Texas borders Kansas`

Ouput Substrings:

```
Texas
borders
Kansas
Texas borders
borders Kansas
Texas borders Kansas
```

Step 2: GENLEX Categories

Input Logical Form:

$borders(texas,kansas)$

Output Categories:

...

...

...

Two GENLEX Rules

Input Trigger	Output Category
a constant c	NP : c
an arity two predicate p	(S\NP)/NP : $\lambda x.\lambda y.p(y,x)$

Example Input: $borders(texas,kansas)$

Output Categories:

NP : $texas$
NP : $kansas$
(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$

All of the Category Rules

Input Trigger	Output Category
a constant c	NP : c
arity one predicate p	N : $\lambda x.p(x)$
arity one predicate p	S\NP : $\lambda x.p(x)$
arity two predicate p	(S\NP)/NP : $\lambda x.\lambda y.p(y,x)$
arity two predicate p	(S\NP)/NP : $\lambda x.\lambda y.p(x,y)$
arity one predicate p	N/N : $\lambda g.\lambda x.p(x)\wedge g(x)$
arity two predicate p and constant c	N/N : $\lambda g.\lambda x.p(x,c)\wedge g(x)$
arity two predicate p	(N\N)/NP : $\lambda x.\lambda g.\lambda y.p(y,x)\wedge g(x)$
arity one function f	NP/N : λg.argmax/min(g(x),λx.f(x))
arity one function f	S/NP : $\lambda x.f(x)$

Step 3: GENLEX Cross Product

Input Training Example

Sentence: Texas borders Kansas
Logic Form: $borders(texas,kansas)$

Output Lexicon

Output Substrings:
- Texas
- borders
- Kansas
- Texas borders
- borders Kansas
- Texas borders Kansas

X

Output Categories:
- NP : $texas$
- NP : $kansas$
- (S\NP)/NP : $\lambda x.\lambda y.borders(x,y)$

GENLEX is the cross product in these two output sets

GENLEX: Output Lexicon

Words	Category
Texas	NP : $texas$
Texas	NP : $kansas$
Texas	(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$
borders	NP : $texas$
borders	NP : $kansas$
borders	(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$
...	...
Texas borders Kansas	NP : $texas$
Texas borders Kansas	NP : $kansas$
Texas borders Kansas	(S\NP)/NP : $\lambda x.\lambda y.borders(y,x)$

A Learning Algorithm

The approach is:

- Online: processes data set one example at a time
- Able to Learn Structure: selects a subset of the lexical entries from GENLEX
- Error Driven: uses perceptron-style parameter updates

Inputs: Training set $\{(x_i, z_i) \mid i=1 \ldots n\}$ of sentences and logical forms. Initial lexicon Λ. Initial parameters w. Number of iterations T.

Computation: 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 $\boxed{\hat{y} = \arg \max_{y \text{ s.t. } L(y)=z_i} w \cdot f(x_i, y)}$
- Define λ_i to be the lexical entries in $\hat{y}$
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$

Step 3: Update Parameters

- Let $\boxed{y' = \arg\max_y w \cdot f(x_i, y)}$
- If $L(y') \neq z_i$
  - Set $w = w + f(x_i, \boxed{\hat{y}}) - f(x_i, \boxed{y'})$

Output: Lexicon Λ and parameters w.

Initialization

The initial lexicon has two types of entries:

- Domain Independent:

  What | S/(S\NP)/N : $\lambda f.\lambda g.\lambda x.f(x) \wedge g(x)$

- Domain Dependent:

  Texas | NP : *texas*

Initial features and weights

- Features: count the number of times each lexical entry is used in a parse
- Initial Weights for Lexical Entries:
  - From GENLEX: small negative values
  - From initial lexicon: small positive values

Related Work

Learning semantic parsers:

- Inductive Logic Prog. [Zelle, Mooney 1996; Thompson, Mooney 2002]
- Machine Translation [Papineni et al. 1997; Wong, Mooney 2006, 2007]
- Probabilistic CFG Parsing [Miller et. al, 1996; Ge, Mooney 2006]
- Support Vector Mach. [Kate, Mooney 2006; Nguyen et al. 2006]

CCG: [Steedman 1996, 2000]

- Log-linear models [Clark, Curran 2003]
- Multi-modal CCG [Baldridge 2002]
- Wide coverage semantics [Bos et al. 2004]
- CCG Bank [Hockenmaier 2003]

Experimental Related Work

COCKTAIL: Tang and Mooney, 2001 (TM01)

- statistical shift-reduce parser learned with ILP techniques

λ-WASP: Wong and Mooney 2007 (WM07)

- Builds a synchronous CFG with statistical machine translation techniques

Experiments

Two database domains:

- Geo880: (geography)
  - 600 training examples
  - 280 test examples
- Jobs640: (job postings)
  - 500 training examples
  - 140 test examples

Evaluation

Test for completely correct semantics

- Precision:

  \# correct / total # parsed

- Recall:

  \# correct / total # sentences

Results

	Geo 880			Jobs 640		
	Prec.	Rec.	F1	Prec.	Rec.	F1
ZC05[1]	96.25	79.29	**86.95**	97.36	79.29	**87.40**
WM07	93.71	80.00	86.31	---	---	---
TM01[2]	89.92	79.40	84.33	93.25	79.84	86.02

[1] Slightly different algorithm than just presented; performs similarly

[2] Used 10-fold cross validation instead of the fixed test set

Example Learned Lexical Entries

Words	Category
states	N : $\lambda x.state(x)$
major	N/N : $\lambda g.\lambda x.major(x) \wedge g(x)$
population	N : $\lambda x.population(x)$
cities	N : $\lambda x.city(x)$
traverses	(S\NP)/NP : $\lambda x.\lambda y.traverse(y,x)$
run through	(S\NP)/NP : $\lambda x.\lambda y.traverse(y,x)$
the largest	NP/N : $\lambda g.argmax(g,\lambda x.size(x))$
rivers	N : $\lambda x.river(x)$
the highest	NP/N : $\lambda g.argmax(g,\lambda x.elev(x))$
the longest	NP/N : $\lambda g.argmax(g,\lambda x.len(x))$
...	...

Outline

- Combinatory Categorial Grammars (CCG)
- A learning algorithm: structure and parameters
- Extensions for spontaneous, unedited text
- Future Work: Context-dependent sentences

A New Challenge

Learning CCG grammars works well for complex, grammatical sentences:

Input: `Show me flights from Newark and New York to San Francisco or Oakland that are nonstop.`

Output: $\lambda x.flight(x) \wedge nonstop(x) \wedge (from(x,NEW) \vee from(x,NYC)) \wedge (to(x,SFO) \vee to(x,OAK))$

What about sentences that are common given spontaneous, unedited input?

Input: `Boston to Prague the latest on Friday.`

Output: $argmax(\lambda x.from(x,BOS) \wedge to(x,PRG) \wedge day(x,FRI), \lambda y.time(y))$

We will see an approach that works for both cases.

Spontaneous, unedited input

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
- Integrate into algorithm which should learn to penalize use

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)
```

$$\dfrac{\dfrac{\textbf{flights}}{\mathbf{N}\ \ \lambda x.flight(x)} \quad \dfrac{\textbf{one way}}{\mathbf{N/N}\ \ \lambda f.\lambda x.f(x) \wedge one\_way(x)}}{\mathbf{N}\ \ \lambda x.flight(x) \wedge one\_way(x)}$$

Review: Composition

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

Disharmonic Composition

- Reverse the direction of the principal category:

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

```
to Prague                the latest                          flight
---------------------    ---------------------------------   ---------------
N\N                      NP/N                                N
λf.λx.f(x)∧to(x,PRG)     λf.argmax(λx.f(x),λx.time(x))       λx.flight(x)
-----------------------------------------------------------
                   NP\N
   λf.argmax(λx.f(x)∧to(x,PRG), λx.time(x))
------------------------------------------------------------------------------
                   N
     argmax(λx.flight(x)∧to(x,PRG), λx.time(x))
```

Missing content words

Insert missing semantic content

- NP : c => N\N : λf.λx.f(x) ∧ p(x,c)

```
flights          Boston                       to Prague
-------------    -------------------------    ---------------------
N                NP                           N\N
λx.flight(x)     BOS                          λf.λx.f(x)∧to(x,PRG)
                 -------------------------
                 N\N
                 λf.λx.f(x)∧from(x,BOS)
------------------------------------------
             N
  λx.flight(x)∧from(x,BOS)
--------------------------------------------------------------------
                         N
          λx.flight(x)∧from(x,BOS)∧to(x,PRG)
```

Missing content-free words

Bypass missing nouns

- N\N : f => N : f(λx.true)

```
Northwest Air                      to Prague
-----------------------------      ---------------------------
N/N                                N\N
λf.λx.f(x)∧airline(x,NWA)          λf.λx.f(x)∧to(x,PRG)
                                   ---------------------------
                                   N
                                   λx.to(x,PRG)
--------------------------------------------------------------
                     N
         λx.airline(x,NWA) ∧ to(x,PRG)
```

A Complete Parse

```
Boston    to Prague               the latest                        on Friday
-------   ---------------------   -------------------------------   ---------------------
NP        N\N                     NP/N                              N\N
BOS       λf.λx.f(x)∧to(x,PRG)    λf.argmax(λx.f(x),λx.time(x))     λf.λx.f(x)∧day(x,FRI)
-------                                                             ---------------------
N\N                                                                 N
λf.λx.f(x)∧from(x,BOS)                                              λx.day(x,FRI)
-------------------------------
N\N
λf.λx.f(x)∧from(x,BOS) ∧to(x,PRG)
-----------------------------------------------------------------
                              NP\N
  λf.argmax(λx.f(x)∧from(x,BOS)∧to(x,PRG), λx.time(x))
------------------------------------------------------------------------------------------
                                       N
     argmax(λx.from(x,BOS)∧to(x,PRG)∧day(x,FRI), λx.time(x))
```

Inputs: Training set $\{(x_i, z_i) \mid i=1 \ldots n\}$ of sentences and logical forms.
Initial lexicon Λ. Initial parameters w. Number of iterations T.

Computation: For $t = 1 \ldots T$, $i = 1 \ldots n$:

Step 1: Check Correctness

- Let $\boxed{y^* = \arg\max_y w \cdot \boxed{f(x_i, y)}}$
- 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 $\boxed{\hat{y} = \arg\max_{y \text{ s.t. } L(y)=z_i} w \cdot \boxed{f(x_i, y)}}$
- Define λ_i to be the lexical entries in $\hat{y}$
- Set lexicon to $\Lambda = \Lambda \cup \lambda_i$

Step 3: Update Parameters

- Let $\boxed{y' = \arg\max_y w \cdot \boxed{f(x_i, y)}}$
- If $L(y') \neq z_i$
  - Set $w = w + \boxed{f(x_i, \hat{y})} - \boxed{f(x_i, y')}$

Output: Lexicon Λ and parameters w.

Related Work for Evaluation

Hidden Vector State Model: He and Young 2006 (HY06)

- Learns a probabilistic push-down automaton with EM

λ-WASP: Wong & Mooney 2007 (WM07)

- Builds a synchronous CFG with statistical machine translation techniques

Zettlemoyer and Collins 2005 (ZC05)

- Uses GENLEX without relaxed grammar

Two Natural Language Interfaces

ATIS (travel planning)

- Manually-transcribed speech queries
- 4500 training examples
- 500 example development set
- 500 test examples

Geo880 (geography)

- Edited sentences
- 600 training examples
- 280 test examples

Evaluation Metrics

Precision, Recall, and F-measure for:

- Completely correct logical forms
- Attribute / value partial credit

```
λx.flight(x) ∧ from(x,BOS) ∧ to(x,PRG)
```

is represented as:

```
{flight, from = BOS, to = PRG }
```

Two-Pass Parsing

Simple method to improve recall:

- For each test sentence that can not be parsed:
  - Reparse with word skipping
  - Every skipped word adds a constant penalty
  - Output the highest scoring new parse

We report results with and without this two-pass parsing strategy

ATIS Test Set

Exact Match Accuracy:

	Precision	Recall	F1
Single-Pass	90.61	81.92	**86.05**
Two-Pass	85.75	84.60	85.16

ATIS Test Set

Partial Credit Accuracy:

	Precision	Recall	F1
Single-Pass	96.76	86.89	91.56
Two-Pass	95.11	96.71	**95.9**
HY 2006	---	---	90.3

Geo880 Test Set

Exact Match Accuracy:

	Precision	Recall	F1
Single-Pass	95.49	83.20	**88.93**
Two-Pass	91.63	86.07	88.76
ZC 05	96.25	79.29	86.95
WM 07	93.72	80.00	86.31

ATIS Development Set

Exact Match Accuracy:

	Precision	Recall	F1
Full method	87.26	74.44	**80.35**
Without features for new rules	70.33	42.45	52.95
Without relaxed word order rules	82.81	63.98	72.19
Without missing word rules	77.31	56.94	65.58

Summary

We presented an algorithm that:

- Learns the lexicon and parameters for a weighted CCG
- Uses online, error-driven updates

We extended it to parse spontaneous, unedited sentences

- Improves accuracy while maintaining the advantages of using a detailed grammatical formalism

We are currently working on learning context-dependent parsers

Future Work: Meaning is context dependent

Input: `Show me flights to Pittsburgh`
Output: $\lambda x.flight(x) \wedge to(x, PIT)$

Input: `from Boston nonstop`
Output: $\lambda x.flight(x) \wedge nonstop(x) \wedge to(x, PIT) \wedge from(x, BOS)$

Input: `Give me the cheapest one`
Output: $\mathrm{argmin}(\lambda x.flight(x) \wedge nonstop(x) \wedge to(x, PIT) \wedge from(x, BOS), \lambda x.cost(x))$

Context-dependent data

Modified ATIS dialogues

- Extract user statements
- Label each statement with context-dependent meaning (by converting original SQL)
- 400 dialogues ($\simeq$3000 queries)
  - average 7.5 per dialogue, min 2, max 55
- All of the challenges from previous work still apply but must also model context

The End

Thanks

Can correct previous statements

Input: `Show me flights to Pittsburgh on thursday night`

Output: $\lambda x.flight(x) \wedge to(x, PIT) \wedge day(x, THU) \wedge during(x, PM)$

Input: `friday before 10am`

Output: $\lambda x.flight(x) \wedge to(x, PIT) \wedge day(x, FRI) \wedge time(x) < 1000$

When do we copy content from previous statements, what do we change?

Can refer to sets

Input: `What airlines fly to Pittsburgh.`

Output: $\lambda x.airline(x) \wedge \exists y.flight(y) \wedge to(y, PIT) \wedge operates(y, x)$

Input: `Which of these flights are nonstop.`

Output: $\lambda x.flight(x) \wedge to(x, PIT) \wedge nonstop(x)$

Which variables define the sets?

We may need world knowledge

Input: `Show me flights from Boston to Pittsburgh`

Output: $\lambda x.flight(x) \wedge from(x, BOS) \wedge to(y, PIT)$

Input: `List return flights`

Output: $\lambda x.flight(x) \wedge from(x, PIT) \wedge to(y, BOS)$

These types of sequences are challenging but relatively rare.