Learning to Map Sentences to Logical form with PCCG

Zettlemoyer and Collins 2005
21st Conference on Uncertainty in Artificial Intelligence
Overview

Learning mapping from S to L
from text to logical form

For example
What states border Texas?
\( \lambda X. \text{state}(X) \) and \( \text{borders}(X, \text{texas}) \)
Issues

- Needs to find predicate names from words
  - relate “border” to “borders(X,Y)”
- Will use log-linear models
  - Like Conditional Random Fields (CRFs)
- Concentrate of “structure learning”
  - In CCG that means “lexical entries”
GEO880

- 880 utterances (sentences plus logical forms)
- US Geography questions
- Which states border Texas?
- Which states border Texas that border Utah?

History:
- CHAT80 (Pereira)
JOBS640

- 640 queries about jobs
- Which jobs offers more than $60K?
- Which jobs offers more than $80K and is in Tuscon?
Semantics

- **Constants:**
  - texa is of type e
  - state is of type \(<e,t>\)
  - size maps entities to real numbers \(<e,r>\)

- **Logical connectors**
  - and, or, not, implies

- **Quantification**
  - Forall, Exists

- **Lambda expressions:**
  - \(\lambda X.\text{state}(X)\) and borders(X,texas)
**Additional quantifiers**

- `count(\lambda X. state(X) \text{ and } \text{borders}(X, \text{texas})`  
- `argmax(\lambda X. state(X), \lambda X. size(X))`  
- `def(\lambda X. state(X))`  
  - *Unique item for which state(X) is true*
The functional application rules:

a. $A / B \quad B \quad \Rightarrow \quad A$

b. $B \quad A \setminus B \quad \Rightarrow \quad A$

The functional application rules (with semantics):

a. $A / B : f \quad B : g \quad \Rightarrow \quad A : f(g)$

b. $B : g \quad A \setminus B : f \quad \Rightarrow \quad A : f(g)$
Utah := NP
Idaho := NP
borders := (S\NP)/NP
the Mississippi := NP : mississippi_river
CCG example

```

Utah

NP

utah

--------------------- borders ---------------------

(S \ NP) / NP

\lambda x. \lambda y. borders( y, x )

------------> (S \ NP) \lambda y. borders( y, idaho ) <-----------

S

borders( utah, idaho )

Idaho

NP

idaho
```
### CCG example 2

<table>
<thead>
<tr>
<th>What</th>
<th>states</th>
<th>border</th>
<th>Texas</th>
</tr>
</thead>
<tbody>
<tr>
<td>((S/(S\setminus NP))/N)</td>
<td>(N)</td>
<td>((S\setminus NP)/NP)</td>
<td>(NP)</td>
</tr>
<tr>
<td>(\lambda f.\lambda g.\lambda x. f(x) \land g(x))</td>
<td>(\lambda x.\text{state}(x))</td>
<td>(\lambda x.\lambda y.\text{borders}(y, x))</td>
<td>\text{texas})</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(\lambda y.\text{borders}(y, \text{texas}))</td>
</tr>
<tr>
<td>(S/(S\setminus NP))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\lambda g.\lambda x.\text{state}(x) \land g(x))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\lambda x.\text{state}(x) \land \text{borders}(x, \text{texas}))</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CCG (here)

- **CCG five rules**
  - Forward/backward application
  - Forward/bacward composition
  - Type raising

- **Specific lexical entries for special words**
  - “what”

- **Lexical items can be multiword**
Probabilistic CCGs

<L, T>
- L is logical form
- T is sequence of steps to derive L
  - Will be called “parse tree”

S a sentence

A PCCG defines conditional distribution
- P(L, T | S)
Ambiguity

- **Ambiguity**
  - Multiple \(<L,T>\) for \(S\)

- **Multiple lexical entries**
  - New York := NP:new_york_city
  - New York := NP:new_york_state

- **Single L by multiple derivations (T)**
  - If *same* \(L\) for \(S\) then “spurious ambiguity”

- **Usually there is ambiguity**
  - Which is good as that’s what is optimized in learning
Log linear models (like CRFs)
f(L, T, S) maps to feature vectors $R$
  • $f(L, T, S) \rightarrow$
    → $f_1(L, T, S)$, $f_2(L, T, S)$, …
  • Where $f_n$ is count of some substructure in $(L, T, S)$.
Model is parameterized by vector $\theta$
PCCG

- Probability of particular syntax/semantics

\[ P(L, T | S; \bar{\theta}) = \frac{e^{\bar{f}(L, T, S) \cdot \bar{\theta}}}{\sum_{(L, T)} e^{\bar{f}(L, T, S) \cdot \bar{\theta}}} \]

- Denominator is sum over all valid parses in S
Features (fn)

- Lexical features only
- Count number times lexical entry used in $T$

Over simplified but seems to work
• Parsing given a PCCG

\[
\arg \max_L P(L|S; \bar{\theta}) = \arg \max_L \sum_T P(L, T|S; \bar{\theta})
\]
Parameter Estimation

• Log-likelihood of the training set

\[
O(\bar{\theta}) = \sum_{i=1}^{n} \log P(L_i | S_i; \bar{\theta})
= \sum_{i=1}^{n} \log \left( \sum_{T} P(L_i, T | S_i; \bar{\theta}) \right)
\]

• Differentiate to get:

\[
\frac{\partial O}{\partial \theta_j} = \sum_{i=1}^{n} \sum_{T} f_j(L_i, T, S_i) P(T | S_i, L_i; \bar{\theta})
- \sum_{i=1}^{n} \sum_{L,T} f_j(L, T, S_i) P(L, T | S_i; \bar{\theta})
\]
Parameter Estimation

- Estimate using dynamic programming
  - Use inside outside algorithm
- Maximize likelihood using
  - (EM) or
  - Stochastic gradient ascent

Set \( \bar{\theta} \) to some initial value

\[
\text{for } k = 0 \ldots N - 1 \\
\text{for } i = 1 \ldots n \\
\bar{\theta} = \bar{\theta} + \frac{\alpha_0}{(1 + ct)} \frac{\partial \log P(L_i | S_i; \bar{\theta})}{\partial \theta}
\]
Learning

- A Induction of a lexicon (structure learning)
- Parameter estimation
Lexical Entry Generation

GENLEX

- Generate entries give S and L that allow sentence to be parsed as L

  - Good examples
    - Utah := NP:utah
    - Idaho := NP:idaho
    - Borders := (S\NP)/NP:\X.\Y.borders(X,Y)

  - Bad examples
    - Utah := NP:idaho
    - Idaho := (S\NP)/NP:\X.\Y.borders(X,Y)
## GENLEX

### Triggers

<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>$NP : texas$</td>
<td>$\arg \max(\lambda x.\text{state}(x) \land \text{borders}(x, \text{texas}), \lambda x.\text{size}(x))$</td>
</tr>
<tr>
<td>arity one predicate $p_1$</td>
<td>$N : \lambda x.\text{p}_1(x)$</td>
<td>$N : \lambda x.\text{state}(x)$</td>
<td></td>
</tr>
<tr>
<td>arity one predicate $p_1$</td>
<td>$S\backslash NP : \lambda x.\text{p}_1(x)$</td>
<td>$S\backslash NP : \lambda x.\text{state}(x)$</td>
<td></td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(S\backslash NP)/NP : \lambda x.\lambda y.\text{p}_2(y, x)$</td>
<td>$(S\backslash NP)/NP : \lambda x.\lambda y.\text{borders}(y, x)$</td>
<td></td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(S\backslash NP)/NP : \lambda x.\lambda y.\text{p}_2(x, y)$</td>
<td>$(S\backslash NP)/NP : \lambda x.\lambda y.\text{borders}(x, y)$</td>
<td></td>
</tr>
<tr>
<td>literal with arity two predicate $p_2$ and constant second argument $c$</td>
<td>$N/N : \lambda g.\lambda x.\text{p}_1(x) \land g(x)$</td>
<td>$N/N : \lambda g.\lambda x.\text{borders}(x, \text{texas}) \land g(x)$</td>
<td></td>
</tr>
<tr>
<td>arity two predicate $p_2$</td>
<td>$(N\backslash N)/NP : \lambda x.\lambda y.\text{p}_2(x, y) \land g(x)$</td>
<td>$(N\backslash N)/NP : \lambda g.\lambda x.\text{borders}(x, y) \land g(x)$</td>
<td></td>
</tr>
<tr>
<td>an arg max / min with second argument arity one function $f$</td>
<td>$NP/N : \lambda g.\text{arg max} / \text{min}(g, \lambda x.\text{f}(x))$</td>
<td>$NP/N : \lambda g.\text{arg max}(g, \lambda x.\text{size}(x))$</td>
<td></td>
</tr>
<tr>
<td>an arity one numeric-ranged function $f$</td>
<td>$S/NP : \lambda x.\text{f}(x)$</td>
<td>$S/NP : \lambda x.\text{size}(x)$</td>
<td></td>
</tr>
</tbody>
</table>
PCCG: training

Inputs:

- Training examples $E = \{(S_i, L_i) : i = 1 \ldots n\}$ where each $S_i$ is a sentence, each $L_i$ is a logical form.
- An initial lexicon $\Lambda_0$
Procedures:

- $\text{PARSE}(S, L, \Lambda, \tilde{\theta})$: takes as input a sentence $S$, a logical form $L$, a lexicon $\Lambda$, and a parameter vector $\tilde{\theta}$. Returns the highest probability parse for $S$ with logical form $L$, when $S$ is parsed by a PCCG with lexicon $\Lambda$ and parameters $\tilde{\theta}$. If there is more than one parse with the same highest probability, the entire set of highest probability parses is returned. Dynamic programming methods are used when implementing PARSE, see section 2.4 of this paper.

- $\text{ESTIMATE}(\Lambda, E, \tilde{\theta})$: takes as input a lexicon $\Lambda$, a training set $E$, and a parameter vector $\tilde{\theta}$. Returns parameter values $\hat{\theta}$ that are the output of stochastic gradient descent on the training set $E$ under the grammar defined by $\Lambda$. The input $\theta$ is the initial setting for the parameters in the stochastic gradient descent algorithm. Dynamic programming methods are used when implementing ESTIMATE, see section 2.4.

- $\text{GENLEX}(S, L)$: takes as input a sentence $S$ and a logical form $L$. Returns a set of lexical items. See section 3.1 for a description of GENLEX.
PCCG: algorithm

- **Step 1**
  - Search small number of sufficient lexical entries to parse S
  - Find highest scoring parse
  - Identify the entries that were used in best parse

- **Step 2**
  - Re-estimate the parameters
  - Using stochastic gradient ascent
Initialization: Define $\bar{\theta}$ to be a real-valued vector of arity $|\Lambda^*|$, where $\Lambda^* = \Lambda_0 \cup \bigcup_{i=1}^{n} \text{GENLEX}(S_i, L_i)$. $\bar{\theta}$ stores a parameter value for each potential lexical item. The initial parameters $\bar{\theta}^0$ are taken to be 0.1 for any member of $\Lambda_0$, and 0.01 for all other lexical items.

Algorithm:

- For $t = 1 \ldots T$

Step 1: (Lexical generation)
  - For $i = 1 \ldots n$:
    - Set $\lambda = \Lambda_0 \cup \text{GENLEX}(S_i, L_i)$.
    - Calculate $\pi = \text{PARSE}(S_i, L_i, \lambda, \bar{\theta}^{t-1})$.
    - Define $\lambda_i$ to be the set of lexical entries in $\pi$.
  - Set $\Lambda_t = \Lambda_0 \cup \bigcup_{i=1}^{n} \lambda_i$

Step 2: (Parameter Estimation)
  - Set $\bar{\theta}^t = \text{ESTIMATE}(\Lambda_t, E, \bar{\theta}^{t-1})$

Output: Lexicon $\Lambda_T$ together with parameters $\bar{\theta}^T$. 
Related Work

- **Older NL interfaces to Databases**
- **X and Mooney work**
  - Cocktail
  - *Had explicit lexical entries, not using CCG*
- **ATIS (flight information)**
  - BBN work (non-CCG)
## Experiments

<table>
<thead>
<tr>
<th></th>
<th>Geo880</th>
<th></th>
<th>Jobs640</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Our Method</td>
<td>96.25</td>
<td>79.29</td>
<td>97.36</td>
<td>79.29</td>
</tr>
<tr>
<td>COCKTAIL</td>
<td>89.92</td>
<td>79.40</td>
<td>93.25</td>
<td>79.84</td>
</tr>
</tbody>
</table>
Best Learned Entries

\[
\begin{align*}
\text{states} & := \quad N : \lambda x.\text{state}(x) \\
\text{major} & := \quad N/N : \lambda f.\lambda x.\text{major}(x) \land f(x) \\
\text{population} & := \quad N : \lambda x.\text{population}(x) \\
\text{cities} & := \quad N : \lambda x.\text{city}(x) \\
\text{rivers} & := \quad N : \lambda x.\text{river}(x) \\
\text{run through} & := \quad (S\backslash NP)/NP : \lambda x.\lambda y.\text{traverse}(y, x) \\
\text{the largest} & := \quad NP/N : \lambda f.\text{arg max}(f, \lambda x.\text{size}(x)) \\
\text{river} & := \quad N : \lambda x.\text{river}(x) \\
\text{the highest} & := \quad NP/N : \lambda f.\text{arg max}(f, \lambda x.\text{elev}(x)) \\
\text{the longest} & := \quad NP/N : \lambda f.\text{arg max}(f, \lambda x.\text{len}(x))
\end{align*}
\]