# Weakly Supervised Training of Semantic Parsers

Krishnamurthy and Mitchell (2012)

Presented by Benjamin Striner

# Contents

# Dataset and Task

# Dataset

- Knowledge base
  - 77 relations from Freebase
- Corpus
  - Web crawl for sentences
  - Discard sentences > 10 words
  - Dependency parsed
- String match between KB and parsed corpus
- Identify sentences containing two entities
  - 2.5 million (e1, e2, s) triples
  - 1% of triples are positive examples; subsample 5% of negative

# Task

- 4 inputs, 1 output, 2 constraints

**Input:**

1. A knowledge base $K = (E, R, C, \Delta)$, as defined above.
2. A corpus of dependency-parsed sentences $S$.
3. A CCG lexicon $\Lambda$ that produces logical forms containing predicates from $K$. Section 4.1 describes an approach to generate this lexicon.
4. A procedure for identifying mentions of entities from $K$ in sentences from $S$. (e.g., simple string matching).

**Output**:

1. Parameters $\theta$ for the CCG that produce correct semantic parses $\ell$ for sentences $s \in S$.

1. Every relation instance $r(e_1, e_2) \in \Delta$ is expressed by at least one sentence in $S$ (Riedel et al., 2010; Hoffmann et al., 2011).
2. The correct semantic parse of a sentence $s$ contains a subset of the syntactic dependencies contained in a dependency parse of $s$.

# Natural Language Queries

- Search for “X is a Y” sentences
- Create queries for Y
- Each query annotated with logical form (50 test, 50 val)
- Test recall of correct logical form

| Example Query | Logical Form |
|---|---|
| capital of Russia | $\lambda x.\textsc{cityCapitalOfCountry}(x, \textsc{Russia})$ |
| wife of Abraham | $\lambda x.\textsc{hasHusband}(x, \textsc{Abraham})$ |
| vocalist from London, England | $\lambda x.\textsc{musician}(x) \wedge \textsc{personBornIn}(x, \textsc{London}) \wedge \textsc{cityInCountry}(\textsc{London}, \textsc{England})$ |
| home of ConocoPhillips in Canada | $\lambda x.\textsc{headquarters}(\textsc{ConocoPhillips}, x) \wedge \textsc{cityInCountry}(x, \textsc{Canada})$ |

Table 3: Example natural language queries and their correct annotated logical form.

| | Precision | Recall |
|---|---|---|
| PARSE | 0.80 | 0.56 |
| PARSE-DEP | 0.45 | 0.32 |

Table 4: Precision and recall for predicting logical forms of natural language queries against Freebase. The table compares PARSE, trained with syntactic supervision to PARSE-DEP, trained without syntactic supervision.

# Combinatory Categorical Grammar

# CCG Notation

$$
\begin{aligned}
\text{town} \quad &:= \quad N : \lambda x.\textsc{city}(x) \\
\text{California} \quad &:= \quad N : \lambda x.x = \textsc{California} \\
\text{in} \quad &:= \quad (N\backslash N)/N : \lambda f.\lambda g.\lambda x. \\
&\qquad \exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x, y)
\end{aligned}
$$

$$
\begin{array}{llll}
X/Y : f \quad Y : g & \Longrightarrow & X : f(g) & (>) \\
Y : g \quad X\backslash Y : f & \Longrightarrow & X : f(g) & (<)
\end{array}
$$

# CCG Rule Application

$$\dfrac{\dfrac{\text{town}}{N : \lambda x.\mathrm{CITY}(x)}\text{Lex} \qquad \dfrac{\dfrac{\text{in}}{(N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y)\wedge g(x)\wedge \mathrm{LOCATEDIN}(x,y)}\text{Lex} \quad \dfrac{\text{California}}{N : \lambda x.x = \mathrm{CALIFORNIA}}\text{Lex}}{N\backslash N : \lambda g.\lambda x.\exists y.y=\mathrm{CALIFORNIA}\wedge g(x)\wedge \mathrm{LOCATEDIN}(x,y)}>}{N : \lambda x.\exists y.y=\mathrm{CALIFORNIA}\wedge \mathrm{CITY}(x)\wedge \mathrm{LOCATEDIN}(x,y)}<$$

Figure 1: An example parse of "town in California" using the example CCG lexicon. The first stage in parsing retrieves a category from each word from the lexicon, represented by the "Lex" entries. The second stage applies CCG combination rules, in this case both forms of function application, to combine these categories into a semantic parse.

# Model

# Graphical Model

- Random Variables
  - Sentence $S_i = s_i$
  - Semantic Parse $L_i = \ell_i$
  - Constraint Satisfaction $Z_i = z_i$
  - Truth of relation $Y_r = y_r$
- Functions
  - Semantic parser $\Gamma$
  - Weak supervision $\Psi$ $\Phi$

$$p(\mathbf{Y} = \mathbf{y}, \mathbf{Z} = \mathbf{z}, \mathbf{L} = \ell | \mathbf{S} = \mathbf{s}; \theta) =$$
$$\frac{1}{Z_{\mathbf{s}}} \prod_r \Psi(y_r, \ell) \prod_i \Phi(z_i, \ell_i, s_i) \Gamma(s_i, \ell_i; \theta)$$

# Factor Graph

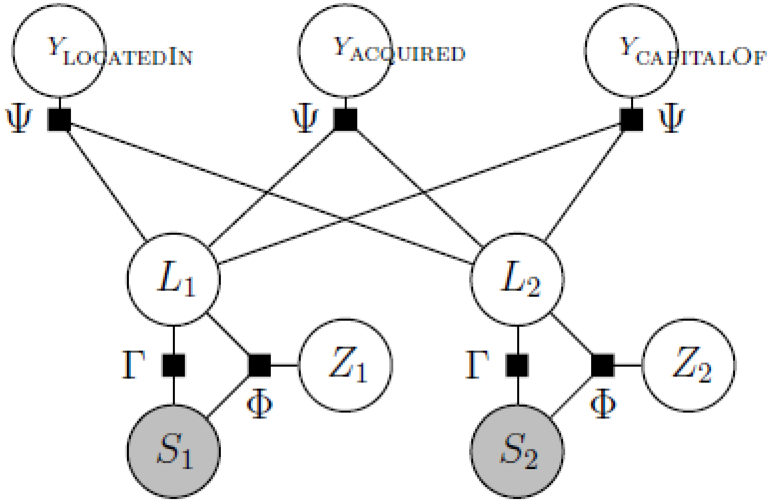


Figure 2: Factor graph containing the semantic parser $\Gamma$ and weak supervision constraints $\Psi$ and $\Phi$, instantiated for an $(e_1, e_2)$ tuple occurring in 2 sentences $S_1$ and $S_2$, with corresponding semantic parses $L_1$ and $L_2$. The knowledge base contains 3 relations, represented by the $Y$ variables.

# Semantic Parser $\Gamma$

- Log-linear probabilistic CCG
  - Count of each entry
  - Number of times each rule applied to each possible argument combination

$$\Gamma(s, \ell; \theta) = \exp\{\theta^T f(\ell, s)\}$$

# Semantic Constraint $\Psi$

- Every relation should be expressed in the sentences
- No semantic parse should be a relation not in the KB
- Yr = is r(e1,e2) expressed in any sentence

$$\Psi(Y_r, \ell) =$$
$$\begin{cases} 1 & \text{if } Y_r = 1 \wedge \exists i.\textsc{Extracts}(\ell_i, r, e_1, e_2) \\ 1 & \text{if } Y_r = 0 \wedge \nexists i.\textsc{Extracts}(\ell_i, r, e_1, e_2) \\ 0 & \text{otherwise} \end{cases}$$

# Syntactic Constraint $\Phi$

- Penalize ungrammatical parses
- Semantic parse should agree with dependency parse
- For every element of parse tree, head words should have a dependency edge

$$\Phi(z, \ell, s) \;=\; \begin{cases} 1 & \text{if } z = \textsc{Agree}(\ell, \textsc{DepParse}(s)) \\ 0 & \text{otherwise} \end{cases}$$

# Training

# Lexicon Generation

- Dependency parse the corpus
- Create entries based on dependency relationships containing entities
  - (relationships on next slide)
- Prune infrequent categories

# Dependency Parse Patterns

| Part of Speech | Dependency Parse Pattern | Lexical Category Template |
|---|---|---|
| Proper Noun | (name of entity $e$)<br>Sacramento | w $:= N : \lambda x.x = e$<br>Sacramento $:= N : \lambda x.x = \textsc{Sacramento}$ |
| Common Noun | $e_1 \xrightarrow{SBJ}$ [is, are, was, ...] $\xleftarrow{OBJ}$ w<br>Sacramento is the capital | w $:= N : \lambda x.c(x)$<br>capital $:= N : \lambda x.\textsc{City}(x)$ |
| Noun Modifier | $e_1 \xleftarrow{NMOD} e_2$<br>Sacramento, California | Type change $N : \lambda x.c(x)$ to $N \vert N : \lambda f.\lambda x.\exists y.c(x) \wedge f(y) \wedge r(x,y)$<br>$N : \lambda x.\textsc{City}(x)$ to $N \vert N : \lambda f.\lambda x.\exists y.\textsc{City}(x) \wedge f(y) \wedge \textsc{LocatedIn}(x,y)$ |
| Preposition | $e_1 \xleftarrow{NMOD}$ w $\xleftarrow{PMOD} e_2$<br>Sacramento in California<br>$e_1 \xrightarrow{SBJ}$ VB* $\xleftarrow{ADV}$ w $\xleftarrow{PMOD} e_2$<br>Sacramento is located in California | w $:= (N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge r(x,y)$<br>in $:= (N\backslash N)/N : \lambda f.\lambda g.\lambda x.\exists y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x,y)$<br>w $:= PP/N : \lambda f.\lambda x.f(x)$<br>in $:= PP/N : \lambda f.\lambda x.f(x)$ |
| Verb | $e_1 \xrightarrow{SBJ}$ w* $\xleftarrow{OBJ} e_2$<br>Sacramento governs California<br>$e_1 \xrightarrow{SBJ}$ w* $\xleftarrow{ADV}$ [IN,TO] $\xleftarrow{PMOD} e_2$<br>Sacramento is located in California<br>$e_1 \xleftarrow{NMOD}$ w* $\xleftarrow{ADV}$ [IN,TO] $\xleftarrow{PMOD} e_2$<br>Sacramento located in California | w* $:= (S\backslash N)/N : \lambda f.\lambda g.\exists x,y.f(y) \wedge g(x) \wedge r(x,y)$<br>governs $:= (S\backslash N)/N : \lambda f.\lambda g.\exists x,y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x,y)$<br>w* $:= (S\backslash N)/PP : \lambda f.\lambda g.\exists x,y.f(y) \wedge g(x) \wedge r(x,y)$<br>is located $:= (S\backslash N)/PP : \lambda f.\lambda g.\exists x,y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x,y)$<br>w* $:= (N\backslash N)/PP : \lambda f.\lambda g.\lambda y.f(y) \wedge g(x) \wedge r(x,y)$<br>located $:= (N\backslash N)/PP : \lambda f.\lambda g.\lambda y.f(y) \wedge g(x) \wedge \textsc{LocatedIn}(x,y)$ |
| Forms of "to be" | (none) | w* $:= (S\backslash N)/N : \lambda f.\lambda g.\exists x.g(x) \wedge f(x)$ |

Table 1: Dependency parse patterns used to instantiate lexical categories for the semantic parser lexicon $\Lambda$. Each pattern is followed by an example phrase that instantiates it. An * indicates a position that may be filled by multiple consecutive words in the sentence. $e_1$ and $e_2$ are the entities identified in the sentence, $r$ represents a relation where $r(e_1, e_2)$, and $c$ represents a category where $c(e_1)$. Each template may be instantiated with multiple values for the variables $e, c, r$.

# Most Frequent Relations

| Relation Name | Relation Instances | Sentences |
|---|---|---|
| cityLocatedInState | 2951 | 13422 |
| cityLocatedInCountry | 1696 | 7904 |
| cityOfPersonBirth | 397 | 440 |
| companiesHeadquarteredHere | 326 | 432 |
| musicArtistMusician | 251 | 291 |
| cityUniversities | 239 | 338 |
| cityCapitalOfCountry | 123 | 2529 |
| hasHusband | 103 | 367 |
| parentOfPerson | 85 | 356 |
| hasSpouse | 81 | 461 |

Table 2: Occurrence statistics for the 10 most frequent relations in the training data. "Relation Instances" shows the number of entity tuples $(e_1, e_2)$ that appear as positive examples for each relation, and "Sentences" shows the total number of sentences in which these tuples appear.

# Training

- Structured perceptron learning rule
- Each train example is two entities and every sentence containing both
- First optimization is easy because y and z are functions of parse
- Second optimization requires beam search over parses
  - Generate 300 parses
  - Eliminate using constraints

$$
\begin{aligned}
\ell^{predicted} &\leftarrow \arg\max_{\boldsymbol{\ell}} \max_{\mathbf{y},\mathbf{z}} p(\boldsymbol{\ell}, \mathbf{y}, \mathbf{z} | \mathbf{s}^j; \theta^t) \\
\ell^{actual} &\leftarrow \arg\max_{\boldsymbol{\ell}} p(\boldsymbol{\ell} | \mathbf{y}^j, \mathbf{z}^j, \mathbf{s}^j; \theta^t) \\
\theta^{t+1} &\leftarrow \theta^t + \sum_i f(\ell_i^{actual}, s_i) \\
&\quad - \sum_i f(\ell_i^{predicted}, s_i)
\end{aligned}
$$

# Results

# Three models

- PARSE: semantic parser
- PARSE+DEP: observes correct dependency parse at test time
- PARSE-DEP: no syntactic constraint

# Results

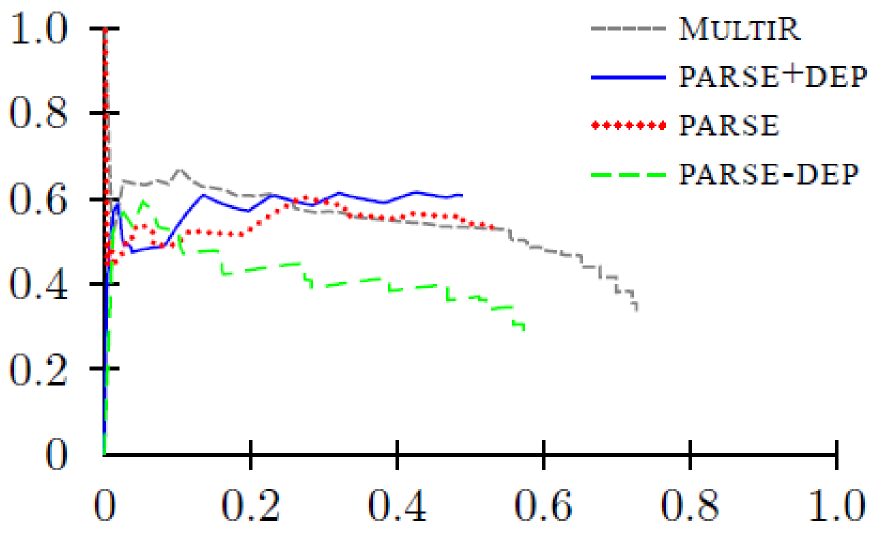


Figure 3: Aggregate precision as a function of recall, for MULTIR (Hoffman et al., 2011) and our three semantic parser variants.

# Discussion

- Supervising a semantic parser directly requires annotating sentences with logical forms
- This model uses more readily-available supervision
  - Knowledge base
  - Dependency parses