Lewis & Steedman (2013): Combined Distributional and Logical Semantics

Min Fang

February 3, 2014

Why Combining Distributional and Logical Semantics is a Good Idea

QA-Task: "Did Google buy YouTube?"

1. Google purchased YouTube
2. Google's acquisition of YouTube
3. Google acquired every company
4. YouTube may be sold to Google
5. Google will buy YouTube or Microsoft
6. Google didn't take over YouTube

- lexical semantics (synonymy)
- interpretation of quantifiers, negation, modals and disjunction

Overview of L&S's Approach

Goal

Learn a CCG lexicon that maps equivalent words onto the same logical form.

$$\begin{aligned} \textit{author} &\vdash N/PP[\textit{of}] && : \lambda x.\lambda y.\text{relation37}(x, y) \\ \textit{write} &\vdash (S\backslash NP)/NP && : \lambda x.\lambda y.\text{relation37}(x, y) \end{aligned}$$

(Though this work only focuses on binary relations. N-ary relations are binarised.)

Overview of L&S's Approach

"Shakespeare wrote Macbeth"

"Shakespeare is the author of Macbeth"

$\Downarrow$

Initial semantic analysis

(map CCG-parser output to a deterministic logical form)

$write_{arg0,arg1}(shakespeare, macbeth)$

$author_{arg0,argOf}(shakespeare, macbeth)$

$\Downarrow$

Entity typing

(apply LDA to assign types)

$write_{arg0:PER,arg1:BOOK}(shakespeare{:}PER, macbeth{:}BOOK)$

$author_{arg0:PER,arg1:BOOK}(shakespeare{:}PER, macbeth{:}BOOK)$

$\Downarrow$

Distributional semantic analysis

(cluster typed predicates)

$\text{relation37}(shakespeare{:}PER, macbeth{:}BOOK)$

Initial Semantic Analysis
$\rightarrow$ Initial Lexicon

- syntactic parsing with the C&C parser
- map output to a deterministic logical form, using predicates such as *write*$'$ and *author*$'$
  - post-processing
    - named-entity compounds as one symbol
    - $\lambda x.barack\_obama(x)$ vs. $\lambda x.barack(x) \wedge obama(x)$
  - most lexical entries generated from
    - syntactic category (captures predicate-argument relation)
    - POS tag (distinguishes between noun and verb predicates)
- manually add entries for closed-class function words
  - negatives, e.g. *no*, *not* etc.
  - quantifiers, e.g. *each*, *every*, numbers, etc.

Example Initial Lexical Entries

	Word	**Category**	**Logical form**
Auto.	author	$N/PP[of]$	$\lambda x.\lambda y.author_{arg0,argOf}(y,x)$
	write	$(S\backslash NP)/NP$	$\lambda x.\lambda y.write_{arg0,arg1}(y,x)$
Man.	every	$NP^{\uparrow}/N$	$\lambda p.\lambda q.\forall x[p(x) \rightarrow q(x)]$
	not	$(S\backslash NP)/(S\backslash NP)$	$\lambda p.\lambda q.\neg p(x)$

Entity Typing

Example

Obama was born in Hawaii.	$(born_{argIn}, Hawaii) \Rightarrow$ LOC type
Obama was born in 1961.	$(born_{argIn}, 1961) \Rightarrow$ DAT type

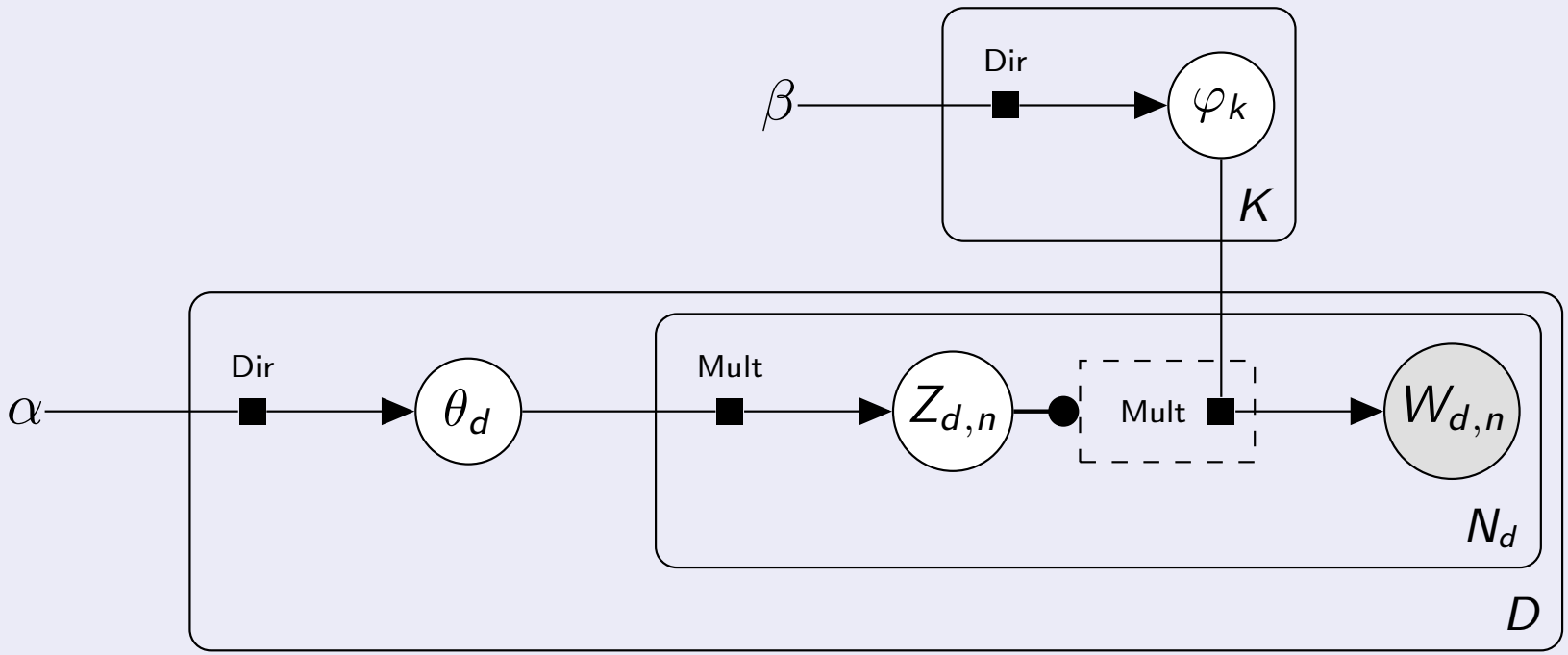
- need a method to model ambiguous predicates
- $\Rightarrow$ subcategorising all terms with a more detailed type (induced from the text)

Following Berant et. al. (2011)

"... we assume that different type signatures of the same predicate have different meanings, but given a type signature a predicate is unambiguous."

Entity Typing - The Generative Model

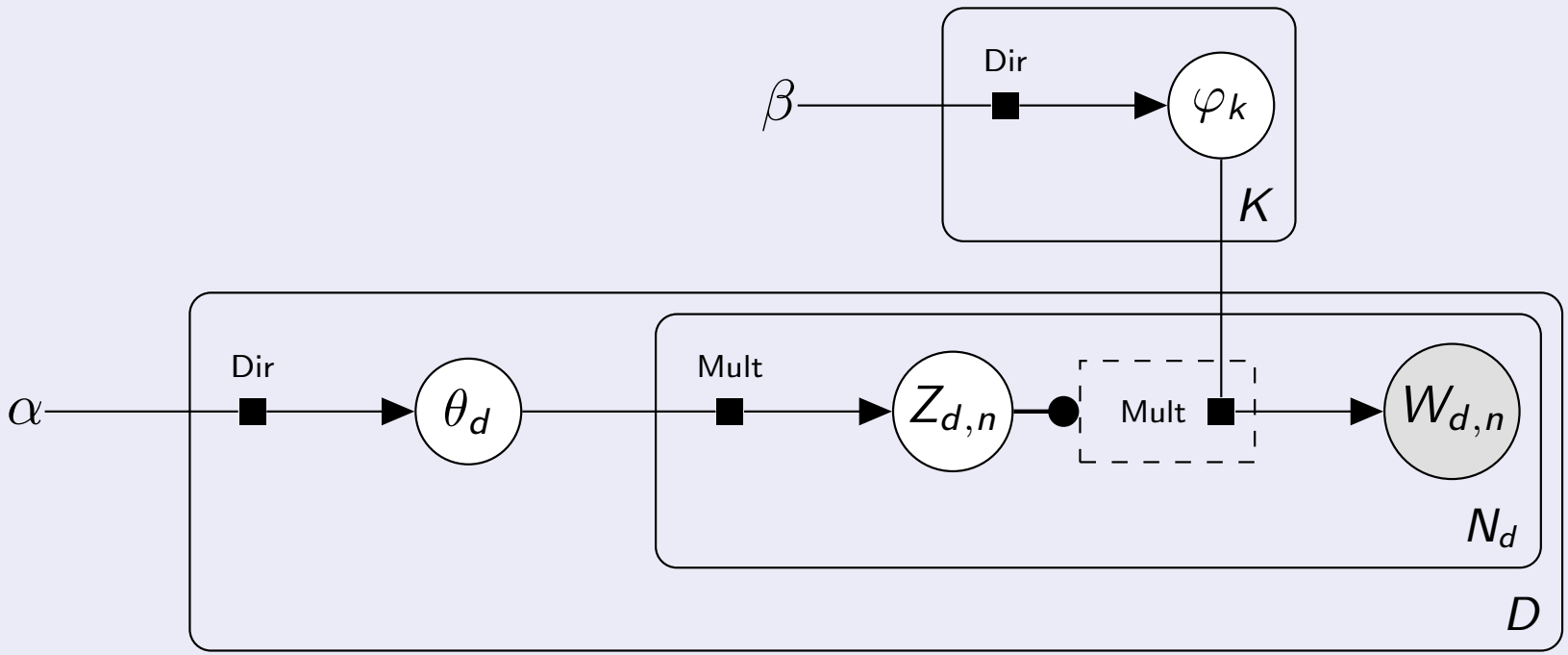
The LDA-Model



- D #documents (unary predicates)
- N_d #words in document d (argument terms to which predicate d applies)
- K #topics (types)

Entity Typing - The Generative Model

The LDA-Model



φ_k $p(arg|k)$ distribution drawn from $Dir(\beta)$

θ_d $p(type|d)$ distribution drawn from $Dir(\alpha)$

$Z_{d,n}$ a type drawn from $Mult(\theta_d)$ for the n-th argument for predicate d

$W_{d,n}$ the observed argument drawn from $Mult(\varphi_{\theta_d})$

Entity Typing - Derivation

- apply the initial type distribution in the lexicon, using estimated distributions from the type model
  - types of nouns: φ distributions (using Bayes' rule)

$$p_x(t) = \frac{p(x|t)p(t)}{p(x)} = \frac{p_{\varphi_t}(x)p(t)}{p(x)}$$

  - type of arguments of binary predicates: θ_d distributions

$$p_x(t) = p_{\theta_d}(t),$$

where x represents the argument-slot of (unary) predicate d

- β-reduction: terms x and y combine to z

$$p_z(t) = \frac{p_x(t)p_y(t)}{\sum_{t'} p_x(t')p_y(t')}$$

Entity Typing - Derivation

Example

- *suit* may be lexically ambiguous between types LEGAL and CLOTHES
- disambiguated when combined with variables *wear* or *file*

$$\dfrac{\dfrac{\text{file}}{\lambda y{:}\left\{\begin{array}{c}DOC=0.5\\LEGAL=0.4\\CLOTHES=0.01\\\dots\end{array}\right\}\lambda x{:}\left\{\begin{array}{c}PER=0.7\\ORG=0.2\\\dots\end{array}\right\}.file_{arg0,arg1}(x,y)}\,(S\backslash NP)/NP \quad \dfrac{\text{a suit}}{\lambda p.\exists y{:}\left\{\begin{array}{c}CLOTHES=0.6\\LEGAL=0.3\\DOC=0.001\\\dots\end{array}\right\}[suit'(y)\wedge p(y)]}\,NP^{\uparrow}}{\lambda x{:}\left\{\begin{array}{c}PER=0.7\\ORG=0.2\\\dots\end{array}\right\}\exists y{:}\left\{\begin{array}{c}LEGAL=0.94\\CLOTHES=0.05\\DOC=0.004\\\dots\end{array}\right\})[suit'(y)\wedge file_{arg0,arg1}(x,y)]}\,S\backslash NP\;<$$

Distributional Relation Clustering Model

- goal: cluster the typed binary predicates $\Rightarrow$ each cluster represents a distinct semantic relation
- cluster based on the *expected* number of times they hold between each argument-pair in the corpus

Example

	write (*PER*, *BOOK*)	*author* (*PER*, *BOOK*)	*born* (*PER*, *DAT*)	*born* (*PER*, *LOC*)	. . .
(Shakespeare, Macbeth)	0.62	0.59	0.01	0.1	
(Dickens, Oliver_Twist)	0.72	0.62	0	0	
(Rowling, Harry_Potter)	0.69	0.68	0	0	
(Obama, 1961)	0	0.02	0.62	0.28	
(Obama, Hawaii)	0.01	0.01	0.27	0.63	
(Bond, Dr._No)	0	0	0	0	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	

Distributional Relation Clustering Model

- type distribution of a binary predicate is the joint distribution of the two argument type distributions
- increase the counts with the corresponding probabilities

Example

$born_{arg0,argIn}(obama, hawaii)$:

$$obama \sim \begin{pmatrix} PER = 0.9 \\ LOC = 0.1 \end{pmatrix} \qquad hawaii \sim \begin{pmatrix} LOC = 0.7 \\ DAT = 0.3 \end{pmatrix}$$

$$born_{arg0,argIn} \sim \begin{pmatrix} arg0 : PER, argIn : LOC = 0.63 \\ arg0 : PER, argIn : DAT = 0.27 \\ arg0 : LOC, argIn : LOC = 0.07 \\ arg0 : LOC, argIn : DAT = 0.03 \end{pmatrix}$$

Clustering Algorithm

- Chinese Whispers algorithm (non-parametric, scalable)
- collection of predicates converted to a graph
  - nodes represent predicates
  - edge weights represent the similarity between predicates (0 if argument types don't match)
  - prune infrequent predicates

1. Each predicate p is assigned to a different semantic relation r_p
2. Iterate over the predicates p in a random order
3. Set $r_p = \arg\max_r \sum_{p'} \mathbb{1}_{r=r_{p'}} sim(p, p')$, where $sim(p, p')$ is the distributional similarity between p and p', and $\mathbb{1}_{r=r'}$ is 1 iff r=r' and 0 otherwise.
4. Repeat (2.) to convergence.

Semantic Parsing Using Relation Clusters

- incorporate the relation cluster identifiers into the lexical entries of the CCG semantic derivation
- introduce *packed predicates*: a function from argument types to relations

Example

$$born \vdash (S \backslash NP)/PP[in] :$$
$$\lambda y.\lambda x. \left\{ \begin{array}{l} (x : PER, y : LOC) \Rightarrow \text{rel49} \\ (x : PER, y : DAT) \Rightarrow \text{rel53} \end{array} \right\} (x, y)$$

Semantic Parsing Using Relation Clusters

Example

$born_{arg0,argIn}(obama, hawaii)$

$$obama \sim \begin{pmatrix} PER = 0.9 \\ LOC = 0.1 \end{pmatrix} \qquad hawaii \sim \begin{pmatrix} LOC = 0.7 \\ DAT = 0.3 \end{pmatrix}$$

$$born_{arg0,argIn} \sim \begin{pmatrix} \text{rel49} = 0.63 \\ \text{rel53} = 0.27 \end{pmatrix}$$

Obama was born in Hawaii in 1961.

$$1961 \sim \begin{pmatrix} LOC = 0.1 \\ DAT = 0.9 \end{pmatrix}$$

$$\begin{Bmatrix} \text{rel49} = 0.63 \\ \text{rel53} = 0.27 \end{Bmatrix} (ob, hw) \quad \wedge \quad \begin{Bmatrix} \text{rel49} = 0.09 \\ \text{rel53} = 0.81 \end{Bmatrix} (ob, 1961)$$

Experiments - Setup

- corpus: Gigaword, ~4 billion words
- trained using 15 types, 5000 iterations of Gibbs sampling
- clustering only with proper nouns (improved precision)

Type	Top Words
1	suspect, assailant, fugitive, accomplice
2	author, singer, actress, actor, dad
5	city, area, country, region, town, capital
8	subsidiary, automaker, airline, Co., GM
10	musical, thriller, sequel, special

Question-Answering Experiments

- automatically create questions by sampling from text
- evaluate the answers found in a different corpus

Example

Google bought YouTube.
$\Rightarrow$ What did Google buy?
$\Rightarrow$ What bought YouTube?

- CCG models: test whether question predicate subsumes answer predicate and arguments match
  - CCG-WordNet: question predicate hypernym of answer predicate?
  - CCG-Distributional: probability of predicates being in the same cluster

QA-Experiments Results

System	Answers	Correct
Relational-LDA	7046	11.6%
Reverb	180	89.4%
CCG-Baseline	203	95.8%
CCG-WordNet	211	94.8%
CCG-Distributional@250	250	94.1%
CCG-Distributional@500	500	82.0%

CCG-Distributional ranks question/answer pairs by confidence.
@250 ... top-250 answers are evaluated.

FraCaS-Suite Experiments

- test understanding of quantifiers
- knowledge of lexical semantics unnecessary

Premises:	Every European has the right to live in Europe. Every European is a person. Every person who has the right to live in Europe can travel freely within Europe.
Hypothesis:	Every European can travel freely within Europe
Solution:	Yes

- manually corrected parse errors
- output of CCG-Distributional: distribution over logical forms
- theorem prover $\Rightarrow$ license inference if it holds in any interpretation with non-zero probability

FraCaS Experiments Results

System	Single Premise	Multiple Premises
MacCartney&Manning 07	84%	-
MacCartney&Manning 08	98%	-
CCG-Dist (parser syntax)	70%	50%
CCG-Dist (gold syntax)	89%	80%

- first approach that allows multiple-premise inferences
- however, less robust (e.g. to parsing errors)

Summary & Future Work

- first work to combine a distributionally induced lexicon with formal semantics
- binary predicates are typed and clustered according to mutual similarity
- packed predicates: map to relations given argument types
- doing inference with underlying relations

- introduce hierarchical clustering
- intergrate ontologies, WordNet to improve clustering
- better method to map to predicate-argument structure