Probabilistic Models for Learning a Semantic Parser Lexicon

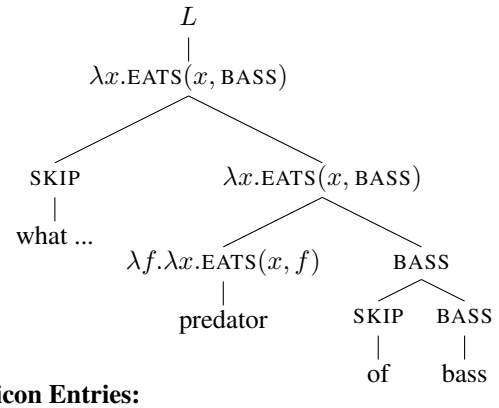
Jayant Krishnamurthy
Allen Institute for Artificial Intelligence
2157 N. Northlake Way, Suite 110
Seattle, WA 98103
jayantk@allenai.org

Abstract

We introduce several probabilistic models for learning the lexicon of a semantic parser. Lexicon learning is the first step of training a semantic parser for a new application domain and the quality of the learned lexicon significantly affects both the accuracy and efficiency of the final semantic parser. Existing work on lexicon learning has focused on heuristic methods that lack convergence guarantees and require significant human input in the form of lexicon templates or annotated logical forms. In contrast, our probabilistic models are trained directly from question/answer pairs using EM and our simplest model has a concave objective that guarantees convergence to a global optimum. An experimental evaluation on a set of 4th grade science questions demonstrates that our models improve semantic parser accuracy (35-70% error reduction) and efficiency (4-25x more sentences per second) relative to prior work despite using less human input. Our models also obtain competitive results on GEO880 without any dataset-specific engineering.

1 Introduction

Semantic parsing has recently gained popularity as a technique for mapping from natural language to a formal meaning representation language, e.g., in order to answer questions against a database (Zelle and Mooney, 1993; Zettlemoyer and Collins, 2005). In order to train a semantic parser, one must first provide a *lexicon*, which is a mapping from words in the language to statements in the meaning representation language. This mapping defines the grammar of the parser and thereby determines the set of meaning representations that can be produced for any given sentence. Therefore, a good lexicon is necessary to achieve both high accuracy and parsing speed. However, the lexicon is unobserved in real semantic parsing applications, leading us to ask: *how do we learn a lexicon for a semantic parser?*



Lexicon Entries:
predator $:= N/NP : \lambda f.\lambda x.\text{EATS}(x, f)$
bass $:= NP : \text{BASS}$

Figure 1: Parse tree of a training example and the lexicon entries derived from it.

This paper presents several novel probabilistic models for learning a semantic parser lexicon. Existing lexicon learning algorithms are heuristic in nature and require either annotated logical forms or manually-specified lexicon entry templates during training. In contrast, our models do not require such templates and can be trained from question/answer pairs and other forms of weak supervision. Training consists of optimizing an objective function with Expectation Maximization (EM), thereby guaran-

Proceedings of NAACL-HLT 2016, pages 606–616,
San Diego, California, June 12-17, 2016.

teeing convergence to a local optimum. Furthermore, the objective function for our simplest model is *concave*, guaranteeing convergence to a global optimum. Our approach generates a probabilistic context-free grammar that represents the space of correct semantic parses for each question; once trained, our approach derives lexicon entries from the most likely parse of each question (Figure 1).

We present an experimental evaluation of our lexicon learning models on a data set of food chain questions from a 4th grade science domain. These questions concern relations between organisms in an ecosystem and have challenging lexical diversity and question length. Our models improve semantic parser accuracy (35-70% error reduction) over prior work despite using less human input. Furthermore, our best model produces a lexicon that contains 40x fewer entries than the most accurate baseline, resulting in a semantic parser that is 4x faster. Our models also obtain competitive results on GEO880 without any dataset-specific engineering.

2 Prior Work

Work on lexicon learning falls into two categories:

Pipelined approaches build a lexicon before training the parser, either by manually defining it (Lee et al., 2014; Angeli et al., 2012) or by using a collection of heuristics. The heuristics often take the form of *lexicon templates*, which are rules that create lexicon entries by pattern-matching training examples (Liang et al., 2011; Krishnamurthy and Mitchell, 2012; Krishnamurthy and Kollar, 2013). These approaches require new lexicon templates for each application. More complex heuristic algorithms have been proposed based on word alignments (Wong and Mooney, 2006; Wong and Mooney, 2007) or common substructures in the meaning representation (Chen and Mooney, 2011); these algorithms all require annotated logical forms.

Joint approaches simultaneously learn a lexicon and the parameters of a semantic parser. Typically, these algorithms use lexicon templates to generate a set of lexicon entries for each example, then heuristically select a subset of these entries to include in the global lexicon while training the parser (Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Artzi and Zettlemoyer, 2013b; Artzi et al., 2014). UBL takes a different approach that performs top-down, iterative splits of an annotated logical form for each training example (Kwiatkowski et al., 2010; Kwiatkowski et al., 2011). Artzi et al. (2015) combine templates with top-down splitting. The heuristic search performed by these algorithms can be difficult to control and we empirically found that these algorithms often selected overly-specific lexicon entries (see Section 4.4).

Other work has avoided the lexicon learning problem altogether by searching over all possible meaning representations (Kate and Mooney, 2006; Clarke et al., 2010; Goldwasser et al., 2011; Berant and Liang, 2014; Pasupat and Liang, 2015; Reddy et al., 2014). The challenge of this approach is that the space of meaning representations for a sentence can be very large, making parsing less efficient and learning more difficult. A practical compromise is to combine a (possibly minimal) lexicon with flexible parsing operations (Liang et al., 2011; Zettlemoyer and Collins, 2007; Poon, 2013; Parikh et al., 2015).

Our lexicon learning models are closely related to machine translation word alignment models (Brown et al., 1993) – our key insight is that *lexicon learning is equivalent to word alignment where the tokenization of one of the sentences is unobserved*. Thus, our models simultaneously "tokenize" the logical form – using a splitting process similar to UBL – and align the resulting logical form "tokens" to words.

3 Probabilistic Models for Lexicon Learning

This section describes our lexicon learning models. For concreteness, we focus on learning a lexicon for a Combinatory Categorial Grammar (CCG) semantic parser with lambda calculus logical forms as the meaning representation; however, our models are applicable to other semantic parsing formalisms and meaning representation languages.

Our models learn a CCG lexicon from a data set of question/label pairs $\{(\mathbf{w}^i, L^i)\}_{i=1}^n$. Each question is a sequence of words, $\mathbf{w}^i = [w_1^i, w_2^i, ...]$, and each label is a *set* of logical forms, $L^i = \{\ell_1^i, ...\}$. Labeling each question with a set of logical forms generalizes many weak supervision settings, including ambiguous supervision (Kate and Mooney, 2007) and ques-

Training Example:

$\mathbf{w}$ = What is the predator of bass ?

L = $\{\lambda x.\text{EATS}(x, \text{BASS}),$
$\lambda x.\text{CAUSE}(\text{INCREASE}(\text{BASS}), \text{INCREASE}(x)),$
$...\}$

Generated Grammar:

Unary rules:

$L \rightarrow \lambda x.\text{EATS}(x, \text{BASS})$

$L \rightarrow \lambda x.\text{CAUSE}(\text{INCREASE}(\text{BASS}), \text{INCREASE}(x))$

Nonterminal rules:

$\lambda x.\text{EATS}(x, \text{BASS}) \rightarrow \text{BASS} \;\; \lambda f.\lambda x.\text{EATS}(x, f)$

$\lambda x.\text{EATS}(x, \text{BASS}) \rightarrow \lambda f.\lambda x.\text{EATS}(x, f) \;\; \text{BASS}$

$\text{BASS} \rightarrow \text{SKIP} \;\; \text{BASS}$

...

Terminal rules:

$\lambda x.\text{EATS}(x, \text{BASS}) \rightarrow$ what

$\lambda x.\text{EATS}(x, \text{BASS}) \rightarrow$ is

$\text{SKIP} \rightarrow$ what

...

Figure 2: Training example (top) and several of the rules in its logical form derivation grammar (bottom).

tion/answer pairs (Liang et al., 2011).[1] The output of learning is a collection of lexicon entries $w := C : \ell$ mapping word w to syntactic category C and logical form ℓ.

Our models are generative models of questions given a label, $P(\mathbf{w}|L)$. The key component of each model is a probabilistic context-free grammar (PCFG) over *correct* logical form derivations. A parse tree in this grammar simultaneously represents (1) the choice of a logical form $\ell \in L$, (2) the way ℓ is constructed from smaller parts, and (3) the alignment between these parts and words in the question. Training each model amounts to learning the rule probabilities of this grammar, including which logical forms are likely to generate which words. Parsing an example with the trained grammar produces an alignment between words and logical forms that is used to construct a lexicon.

3.1 Logical Form Derivation Grammar

The logical form derivation grammar is a PCFG constructed to represent the set of correct logical form derivations – i.e., correct semantic parses – of each training example. The grammar's nonterminals are logical forms and its binary production rules represent ways that pairs of logical forms can combine in the semantic parser. The grammar's terminals are words and its terminal production rules represent lexicon entries. The complete grammar is a union of many smaller grammars, each of which is constructed to represent the logical form derivations of a single example. Figure 2 shows a training example and a portion of the grammar generated for it, and Figure 1 shows a parse tree in the grammar.

Our algorithm for constructing the PCFG for a training example $(\mathbf{w}, L)$ uses a top-down approach that iteratively splits logical forms in L. Assume we are given a procedure $\text{SPLIT}(f)$ that outputs a list of ways to split f into a pair of logical forms (g, h). Grammar generation performs the following steps:

1. **Model weak supervision.** Add L to the grammar as a nonterminal and add a unary rule $L \rightarrow \ell$ for all $\ell \in L$.

2. **Enumerate logical form splits.** For all $\ell \in L$, perform a depth-first search over logical forms starting at ℓ. To explore a logical form f during the search, use $\text{SPLIT}(f)$ to produce a collection of g, h pairs. For each g, h pair, add the binary rules $f \rightarrow g\ h$ and $f \rightarrow h\ g$ to the grammar, then add g and h to the search queue for later exploration.

3. **Create lexicon entries.** Add a terminal rule $f \rightarrow w$ to G for every word in the question, $w \in \mathbf{w}$, and logical form f encountered during the search above.

4. **Allow word skipping.** Add a special SKIP nonterminal, along with the rules $f \rightarrow f\ \text{SKIP}$, $f \rightarrow \text{SKIP}\ f$ and $\text{SKIP} \rightarrow w$ for all logical forms f and words $w \in \mathbf{w}$.

The SPLIT procedure required above depends on the meaning representation language, but is application-independent. For our lambda calculus representation, $\text{SPLIT}(f)$ returns a list of logical forms g, h such that $f = g(h)$. We use similar constraints as Kwiatkowski et al. (2011) to keep the number of splits manageable. Note that SPLIT could also include composition by returning g, h

[1] In the second case, the set L can be generated by enumerating logical forms and evaluating each one to determine if it produces the correct answer. See Section 5 for a discussion of the benefits and limitations of this process.

pairs such that $f = \lambda x.g(h(x))$; however, we did not explore this possibility in this paper.

An important property of this grammar is that it *excludes many logical form derivations that cannot lead to the label*. For example, the grammar in Figure 2 does not let us apply $\lambda x.\text{EATS}(x, \text{BASS})$ to BASS – even though this operation would be permitted by CCG – because there is no way to reach the label L from the result $\text{EATS}(\text{BASS}, \text{BASS})$. This property reduces the number of possible parses of a question relative to a CCG parser with the same lexicon entries, making parsing more efficient.

The logical form derivation grammar G is constructed by applying the above process to every example in the data set. Let $P(t|L;\theta)$ denote the probability of generating tree t from G given $\text{ROOT}(t) = L$ and parameters θ. This probability factors into a product of production rule probabilities:

$$P(t|L;\theta) = \prod_{(f \to g\ h) \in t} P(f \to g\ h;\theta) \times \prod_{(f \to w) \in t} P(f \to w;\theta)$$

In the above equation, $P(f \to g\ h;\theta)$ and $P(f \to w;\theta)$ represent the conditional probability of selecting a production rule given the nonterminal f. We use $P(\mathbf{w}, t|L;\theta)$ to denote $P(t|L;\theta)$ where the terminals of t are equal to the question $\mathbf{w}$. In the following, sums over trees t are implicitly over all trees permitted by G.

3.2 Independent Model

The independent model assumes that each word w_j of a question $\mathbf{w}$ is generated independently from a parse tree t chosen uniformly at random given the label L. This simple model allows two words in the same question to be generated by different trees. The probability of a question given a label is:

$$P(\mathbf{w}|L;\theta) = \prod_{j=1}^{|\mathbf{w}|} \sum_{f} P(f \to w_j;\theta) \#(f, j, L, |\mathbf{w}|)$$

The final term $\#(f, j, L, |\mathbf{w}|)$ is the fraction of trees with root L and $|\mathbf{w}|$ terminals where the jth terminal symbol is generated by nonterminal f. This term appears due to the assumption that trees are drawn uniformly at random. The parameters θ of this model are the terminal production rule probabilities, which are modeled as a conditional probability table: $P(f \to w;\theta) = \theta_{f,w}$ where $\sum_w \theta_{f,w} = 1$.

The independent model is a generalization of IBM Model 1 (Brown et al., 1993) to the lexicon learning problem, and – like IBM Model 1 – its loglikelihood function is *concave* (see Appendix A). Therefore, the EM algorithm will converge to a global optimum of the data loglikelihood under this model.

3.3 Coupled Model

The coupled model generates the entire question $\mathbf{w}$ from a single parse tree t that is generated given L. This model removes the previous model's naïve assumption that each word is generated independently. The probability of a question given a label under this model is:

$$P(\mathbf{w}|L;\theta) = \sum_{t} P(\mathbf{w}, t|L;\theta)$$

Theoretically, we could learn both of the production rule distributions that compose $P(\mathbf{w}, t|L;\theta)$ in this formulation. However, in practice, the large number of nonterminals makes it challenging to learn a conditional probability table for the binary production rules. Therefore, we again assume the trees are drawn uniformly at random and only learn a conditional probability table for the terminal production rules.

3.4 Coupled Loglinear Model

The coupled loglinear model replaces the conditional probability tables of the coupled model with loglinear models. Loglinear models can share parameters across different – but intuitively similar – production rules. For example, both $\lambda x.\text{EATS}(x, \text{BASS})$ and $\lambda x.\text{EATS}(x, \text{FROG})$ should have similar distributions over production rules when BASS is appropriately replaced by FROG. We can produce this effect with loglinear models by assigning similar feature vectors to these rules.

This model uses *three* locally-normalized loglinear models to parameterize the distribution over production rules. First, a rule model decides whether or not to apply a terminal production rule. Next, given

this model's response, a second loglinear model decides which of the chosen kind of rules to apply. This approach ensures that each nonterminal symbol has a proper conditional probability distribution over rules. The production rule distributions are parameterized as:

$$P(f \to w; \theta) = \frac{\exp(\theta_r^T \phi_r(f) + \theta_t^T \phi_t(f, w))}{Z_r Z_t(f)}$$
$$P(f \to g\ h; \theta) = \frac{\exp(\theta_n^T \phi_n(f, g, h))}{Z_r Z_n(f)}$$

In the above equation, the r, t and n subscripts indicate terms of the rule, terminal and nonterminal models, respectively. The $Z_r, Z_t(f)$ and $Z_n(f)$ terms denote the partition functions of each model, and ϕ_r, ϕ_n and ϕ_t are functions mapping nonterminals and production rules to feature vectors. We use indicator features for logical form patterns, where each pattern is produced by replacing all of a logical form's subexpressions below a certain depth with their types.

3.5 Training with Expectation Maximization

We train all three models by maximizing data log-likelihood with EM (Dempster et al., 1977). Training the independent model is equivalent to training a mixture of multinomials where each word of each question has its own prior over cluster assignments. Let θ^m represent the model parameters on the mth training iteration. The E-step calculates expected count of each terminal production rule:

$$E_{f,w} \leftarrow \sum_{i,j:w_j^i = w} P(f \to w_j^i; \theta^m) \#(f, j, L^i, |\mathbf{w}^i|)$$

The term $\#(f, j, L^i, |\mathbf{w}^i|)$ – representing the fraction of trees where nonterminal f generates the jth word of question i – can be calculated by parsing each example once using the inside/outside algorithm. The M-step re-estimates the terminal production rule probabilities using these expected counts:

$$\theta_{f,w}^{m+1} \leftarrow \frac{E_{f,w}}{\sum_{w'} E_{f,w'}}$$

Training the coupled models is a standard application of EM to learning the parameters of a latent probabilistic CFG. The E-step calculates the expected number of occurrences of each production rule in each example:

$$E_{f,g,h} \leftarrow \sum_{i,t} \#(f \to g\ h, t) P(t|\mathbf{w}^i, L^i; \theta^m)$$
$$E_{f,w} \leftarrow \sum_{i,t} \#(f \to w, t) P(t|\mathbf{w}^i, L^i; \theta^m)$$

In the above equation, the function $\#(f \to w; t)$ returns the number of occurrences of $f \to w$ in t. These expected counts can be computed efficiently using the inside/outside algorithm.[2] The M-step of the coupled model is the same as that of the independent model above. The M-step of the coupled loglinear model solves an optimization problem to fit the loglinear models to the computed expectations (Berg-Kirkpatrick et al., 2010):

$$\theta^{m+1} \leftarrow \arg\max_{\theta} \sum_{f,w} E_{f,w} \log P(f \to w; \theta) + \sum_{f,g,h} E_{f,g,h} \log P(f \to g\ h; \theta)$$

This problem factors into three separate optimization problems: a binary logistic regression for the rule model, and estimating two conditional distributions over nonterminal and terminal production rules. We use L-BFGS to solve these problems.

3.6 Producing a Lexicon

Given parameters θ and a data set $\{(\mathbf{w}^i, L^i)\}_{i=1}^n$, we produce a CCG lexicon from the terminal production rules of the most probable parse of each example. First, we parse each question $\mathbf{w}^i$ conditioning on the parse tree root being L^i. Second, we generate lexicon entries from the highest scoring parse tree for each example. We identify the nonterminal f that generates each word $w \in \mathbf{w}$ and, if $f \neq$ SKIP, create a lexicon entry $w := C : f$. As in previous work, we derive the syntactic category C from the semantic type of the logical form (Kwiatkowski et

[2] A further efficiency improvement is to note that, when parsing an example, it is sufficient to use the subset of G that was generated for it.

w:	Which organism in the diagram is eaten by the blackbird?
ℓ:	$\lambda x.\text{EATS}(\text{BLACKBIRD}, x)$
c:	sun → grains → blackbird → hawk → falcon
a:	grains
w:	The __ is neither a producer or a consumer.
ℓ:	$\lambda x.\text{NOT}(\text{ANIMAL}(x) \vee \text{PLANT}(x))$,
c:	sun → grasses → hartebeest → lion
a:	sun
w:	If the sun is blocked by clouds for a long period of time, which animal will be most quickly threatened by starvation?
ℓ:	$\lambda x.\text{CAUSE}(\text{DECREASE}(\text{SUN}), \text{DECREASE}(x))$
c:	sun → algae → shrimp → smelt → salmon
a:	algae
w:	If the seals were killed off, the population of penguin would most likely
ℓ:	$\lambda f.\text{CAUSE}(\text{DECREASE}(\text{SEAL}), f(\text{PENGUIN}))$
c:	sun → algae → squid → penguin → seal
a:	increase

Figure 3: Examples from FOODCHAINS. Each example consists of a question **w**, logical form ℓ, food chain c and answer a.

al., 2011). The argument directions of C are determined by walking up the tree and noting the relative position of each of its arguments. Figure 1 shows an example of a predicted parse tree and the lexicon entries generated from it.

4 Evaluation

We compare our lexicon learning models against several baselines on two data sets: FOODCHAINS, containing 4th grade science food chain questions, and GEO880, containing geography questions. These data sets each present different challenges for lexicon learning: FOODCHAINS has more difficult language – long questions and more lexical variation – while GEO880 has more complex logical forms. Our results demonstrate that our models perform better than several baselines with difficult language while simultaneously performing reasonably well with complex logical forms.

Code, data and other supplementary material for this paper is available at `http://www.allenai.org/paper-appendix/naacl2016-lexicon`.

	FOODCHAINS	GEO880
Examples	774	880
Word types	446	279
Word types w/o entity names	357	157
Tokens per question	11.6	7.56
Predicates in ontology	15	38
Constants per logical form	4.0	7.7

Table 1: Data statistics for FOODCHAINS and GEO880.

4.1 Data

We collected a new data set, FOODCHAINS, that contains 774 food chain questions designed to imitate actual questions from the New York State Grade 4 Regents Exam. Each example in the data set consists of a natural language question and a food chain, which is a list of organisms that eat each other. The questions are multiple choice and the answer options are either animals from the food chain or a direction of change, e.g., "increase." Each question also has a logical form annotated by the first author, which is necessary to train some of the baseline systems. The denotation of each predicate – and therefore logical form – is a deterministic function of the food chain. Figure 3 shows some examples from this data set.

FOODCHAINS was created using Mechanical Turk. We first manually created 25 distinct food chains of various lengths containing different organisms and a set of question templates – questions with a blank – based on real Regents questions. In the first task, Turk workers were shown a randomly-selected food chain, question template, and answer, and were asked to complete the question by filling in the blank. In the second task, workers paraphrased questions from the first task, thereby eliminating the templated structure and increasing lexical variation (similar to Wang et al., (2015)). In the third task, a worker validated each question by answering it.

Statistics of FOODCHAINS are presented in Table 1 alongside corresponding statistics of GEO880 (Zelle and Mooney, 1996; Tang and Mooney, 2001; Zettlemoyer and Collins, 2005). FOODCHAINS differs from GEO880 in two significant and interesting ways. First, although the data set contains relatively few predicates, there are many ways to reference each predicate – for example, consider the diversity in references to DECREASE in Figure 3. Second, the questions are long but contain many uninformative

Model	Accuracy
Independent Model (§3.2)	78.7%
Coupled Model (§3.3)	79.0%
Coupled Loglinear Model (§3.4)	**81.7%**

Table 2: Comparison of semantic parser accuracy on FOODCHAINS when trained using our three proposed probabilistic models for lexicon learning.

words that can safely be ignored by the parser.

4.2 Methodology

We compare lexicon learning algorithms by performing an end-to-end evaluation, measuring the question answering accuracy of a CCG semantic parser trained with the learned lexicon. The parser has a rich set of features, including lexicon entry features, dependency features, and dependency distance features. The parser is also permitted to skip words in the question for a learned per-word cost. We train the parser by optimizing data loglikelihood with 100 epochs of stochastic gradient descent.

All experiments on FOODCHAINS are performed using 5-fold cross validation. All questions about a single food chain appear in the same fold, ensuring that the questions in the held-out fold reference unseen food chains.

4.3 Comparing Probabilistic Models

Our initial experiment compares the three probabilistic models proposed in Section 3 on FOODCHAINS. We generated three lexicons by training each model using 10 iterations of EM. We used a smoothing parameter of 0.1 when estimating conditional probability tables, and an L2 regularization parameter of 10^{-6} when estimating loglinear models. We also initialized the coupled model with the optimum of the independent model. All of these models are trained without labeled logical forms, instead using an automatically enumerated set of logical forms that evaluate to the correct answer.

Table 2 presents the result of this evaluation. All three models perform roughly similarly, with the coupled loglinear model slightly outperforming the others. The competitive performance of the independent model is interesting because its concave objective function is easy to optimize. The remaining experiments compare against the coupled loglinear model, which we dub PAL, short for "Probabilistic Alignments for Lexicon learning."

Model	Logical forms?	Lexicon Templates?	Accuracy	PAL % Err. Red.
PAL	**No**	**No**	**81.7%**	–
POS	**No**	Yes	70.5%	38.0%
UBL	Yes	**No**	40.6%	69.1%
ZC2007	Yes	Yes	49.4%	63.8%
ADP2014	**No**	Yes	32.4%	72.9%

Table 3: Semantic parser accuracy comparison for several lexicon learning algorithms on FOODCHAINS. The middle two columns note the human input required by each algorithm and the final column notes the relative error reduction of PAL over each baseline.

4.4 Lexicon Learning Baselines

Our second experiment compares PAL with four baseline lexicon learning algorithms. The first baseline, POS, defines a set of lexicon entries for each word in the training set based on its part-of-speech tag (Liang et al., 2011). We iteratively developed these templates to cover the data set, and the lexicon generated by these templates can correctly parse 96% of the examples in FOODCHAINS. The remaining three baselines, ZC2007 (Zettlemoyer and Collins, 2007), UBL (Kwiatkowski et al., 2011), and ADP2014 (Artzi et al., 2014), are joint lexicon and parameter learning algorithms. We used the UW SPF (Artzi and Zettlemoyer, 2013a) implementations of UBL and ZC2007. For these two models, we trained our parser using the learned lexicon to ensure consistency of implementation details.[3] We initialized UBL's parameters using GIZA++ (Och and Ney, 2003) as in the original paper. The lexicon entry templates for ZC2007 and ADP2014 are derived from the POS templates to ensure coverage of the questions; these algorithms allow each template to apply to 1-4 word phrases.

Table 3 compares the accuracy of semantic parsers trained with these baseline approaches to PAL. Our model outperforms all of the baselines, beating the most accurate baseline, POS, by more

[3] We also postprocessed the lexicon entries to improve performance, for example, by removing lexicon entries that skip words. In both cases, our parser was more accurate than the UW SPF parser.

Model	Lexicon Size	Parse Time (ms)
POS	4,282	7.8
UBL	3,356	14.9
ZC2007	2,949	43.4
ADP2014	401	1.1
Independent Model	410	6.7
Coupled Model	318	3.1
PAL	184	1.8

Table 4: Lexicon size excluding entity names and parse time per question on FOODCHAINS averaged across folds per model. Our models produce smaller lexicons that lead to faster semantic parsers.

Model	Accuracy
UBL (Kwiatkowski et al., 2011)	88.6%
ZC2007 (Zettlemoyer and Collins, 2007)	86.1%
PAL	81.8%
w/ factored lexicon	85.4%

Table 5: Logical form accuracy on GEO880 compared to previously reported CCG parsing results. Both UBL and ZC2007 use special CCG extensions to improve performance; adding one of these to PAL brings its accuracy near that of these systems.

than 10 points. Note that all of these baselines also use more human input than our model, either in the form of lexicon templates or logical forms. Table 4 compares the average number of lexicon entries and semantic parser speed of the baselines with our models. PAL produces the most compact lexicon and second fastest parser, which is 4x faster than POS, the baseline with the highest accuracy. The correlation between lexicon size and parse time is imperfect due to word frequency and co-occurrence effects.

The three joint lexicon and parameter learning algorithms perform poorly on our data set for two reasons. First, the long question length increases the difficulty of finding good lexicon entries. The algorithms with lexicon templates were frequently unable to find a correct parse for long questions, even with a large beam size – we ran ADP2014 with a beam size of 10000. (Note that POS does not suffer from this problem because it only generates lexicon entries for a few parts of speech, so most of the words in a question are ignored by default.) Second, these algorithms' discriminative objectives inherently prefer lexicon entries with highly specific word sequences. This preference interacts poorly with the uninformative words in the data set, leading these algorithms to produce many lexicon entries for long phrases that do not generalize well. The lexicon sizes in Table 4 are suggestive of this problem for ZC2007 and UBL; ADP2014 also has this problem, but its voting mechanism prunes lexicon entries much more aggressively. We tried to solve this problem for ZC2007 and ADP2014 by restricting lexicon templates to apply to at most 1 word, but this change actually reduced accuracy. An investigation of this phenomenon found that reducing the length of the templates made it even more difficult for these models to find correct parses for long questions.

In contrast to these baselines, our models do not suffer from either of these problems because the logical form derivation grammar restricts the search to *correct* derivations and our generative objective prefers frequently-occurring lexicon entries. Our models actually consider a *larger* space of possible lexicon entries than ZC2007 and ADP2014 with 1 word templates, yet find better lexicon entries.

4.5 Geo880 Evaluation

We performed an additional evaluation on GEO880 to demonstrate that our models can work with more complex logical forms. GEO880 is a good data set for this evaluation because its logical forms contain, on average, about twice as many constants as FOODCHAINS. We applied PAL to generate a lexicon for this data set using its included logical form labels, then trained a CCG semantic parser with this lexicon. Table 5 compares the accuracy of this parser with previous CCG lexicon learning results on this data set using the standard 600/280 train/test split. The comparison to PAL is inexact because both prior systems use CCG parsers with special extensions that improve performance on this data set. UBL uses factored lexicon entries that generalize better to certain infrequent entries, and ZC2007 includes relaxed parsing operators that fix common parsing errors. Examining the errors made by our parser, we found many cases where these extensions would help. We therefore trained another CCG parser using a lexicon generated by postprocessing PAL's lexicon to include factored lexicon entries. This parser achieves an accuracy close to previous work.

5 Discussion

We introduce several probabilistic models for learning a semantic parser lexicon that can be trained from question/answer pairs and other forms of weak supervision. Our experimental results demonstrate that our models improve semantic parser accuracy and efficiency relative to prior work on data sets with more challenging language, despite using less human input. Furthermore, we find that our independent model is nearly as effective as more complex models, but has a concave objective function that guarantees training converges to a global optimum.

A possible complaint about our approach is that, when training from question/answer pairs, it is not practical to enumerate all logical forms that produce the correct answer. We believe this complaint is misguided because enumerating logical forms is unavoidable in the question/answer setting. *Every* algorithm uses an enumerate-and-test approach to identify correct logical forms; this process occurs in the gradient computation of semantic parser training and in template-based lexicon learning algorithms such as ADP2014. The critical question is not *whether* enumeration is used, but rather *how* logical forms are enumerated. Many strategies are possible and different strategies are likely to be effective on different data sets. In fact, the failure of template-based algorithms on FOODCHAINS is largely a failure of their template-guided enumeration strategy to find correct logical forms. Choosing an enumeration strategy and related questions – e.g., does semantic parser parameter learning affect the enumeration? – are empirical questions that must be decided on a task-specific basis.

A recent trend in semantic parsing has been to avoid lexicon learning, instead directly searching the space of all possible logical forms. However, we think that lexicon learning still serves a valuable purpose. Fundamentally, a lexicon constrains the space of logical forms searched by a semantic parser; these constraints improve efficiency and can improve accuracy as long as they do not exclude correct logical forms. Thus, a promising approach to building a semantic parser is to automatically learn a lexicon using one of our models, then (perhaps) manually specify new parsing operations to correct any problems with the learned lexicon. We plan to apply this approach in the future to construct semantic parsers for more challenging tasks.

Acknowledgments

We gratefully acknowledge Oyvind Tafjord, Matt Gardner, Peter Turney, Oren Etzioni and the anonymous reviewers for their helpful comments.

References

Gabor Angeli, Christopher D. Manning, and Daniel Jurafsky. 2012. Parsing time: Learning to interpret time expressions. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Yoav Artzi and Luke Zettlemoyer. 2013a. UW SPF: The University of Washington Semantic Parsing Framework.

Yoav Artzi and Luke Zettlemoyer. 2013b. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics*.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*.

Yoav Artzi, Kenton Lee, and Luke Zettlemoyer. 2015. Broad-coverage CCG semantic parsing with AMR. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*.

Taylor Berg-Kirkpatrick, Alexandre Bouchard-Côté, John DeNero, and Dan Klein. 2010. Painless unsupervised learning with features. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. 2010. Driving semantic parsing from

the world's response. In *Proceedings of the Fourteenth Conference on Computational Natural Language Learning*.

Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

Dan Goldwasser, Roi Reichart, James Clarke, and Dan Roth. 2011. Confidence driven unsupervised semantic parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*.

Rohit J. Kate and Raymond J. Mooney. 2006. Using string-kernels for learning semantic parsers. In *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*.

Rohit J. Kate and Raymond J. Mooney. 2007. Learning language semantics from ambiguous supervision. In *Proceedings of the 22nd Conference on Artificial Intelligence*.

Jayant Krishnamurthy and Thomas Kollar. 2013. Jointly learning to parse and perceive: Connecting natural language to the physical world. *Transactions of the Association of Computational Linguistics – Volume 1*.

Jayant Krishnamurthy and Tom M. Mitchell. 2012. Weakly supervised training of semantic parsers. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*.

Kenton Lee, Yoav Artzi, Jesse Dodge, and Luke Zettlemoyer. 2014. Context-dependent semantic parsing for time expressions. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the Association for Computational Linguistics*.

Franz Josef Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*.

Ankur P. Parikh, Hoifung Poon, and Kristina Toutanova. 2015. Grounded semantic parsing for complex knowledge extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Hoifung Poon. 2013. Grounded unsupervised semantic parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*.

Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*.

Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Human Language Technology Conference of the NAACL*.

Yuk Wah Wong and Raymond J. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*.

John M. Zelle and Raymond J. Mooney. 1993. Learning semantic grammars with constructive inductive logic programming. In *Proceedings of the 11th National Conference on Artificial Intelligence*.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*.

Luke S. Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference*

on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.

Appendix A: Proof of Concavity

The loglikelihood $O(\theta)$ of the independent model is:

$$\begin{aligned} O(\theta) &= \sum_{i=1}^{n} \log P(\mathbf{w}^i | L^i; \theta) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{|\mathbf{w}^i|} \log \sum_{f} \theta_{f,w_j^i} \#(f, j, L^i, |\mathbf{w}^i|) \end{aligned}$$

Each $\log$ term above is concave in θ because $\log$ is a concave function applied to an affine function of θ. (Note that the $\#(f, j, L^i, |\mathbf{w}^i|)$ terms do not depend on θ.) Finally, $O(\theta)$ is concave because it is a sum of concave functions.