Iterative Search for Weakly Supervised Semantic Parsing

Pradeep Dasigi[♡], **Matt Gardner**[♥], **Shikhar Murty**[♢],
Luke Zettlemoyer[♣], and **Eduard Hovy**[♠]
[♡]Allen Institute for Artificial Intelligence, Seattle, Washington
[♥]Allen Institute for Artificial Intelligence, Irvine, California
[♢]Mila, Université de Montréal [♣]University of Washington [♠]Carnegie Mellon University
pradeepd@allenai.org

Abstract

Training semantic parsers from question-answer pairs typically involves searching over an exponentially large space of logical forms, and an unguided search can easily be misled by spurious logical forms that coincidentally evaluate to the correct answer. We propose a novel iterative training algorithm that alternates between searching for consistent logical forms and maximizing the marginal likelihood of the retrieved ones. This training scheme lets us iteratively train models that provide guidance to subsequent ones to search for logical forms of increasing complexity, thus dealing with the problem of spuriousness. We evaluate these techniques on two hard datasets: WIKITABLEQUESTIONS (WTQ) and Cornell Natural Language Visual Reasoning (NLVR), and show that our training algorithm outperforms the previous best systems, on WTQ in a comparable setting, and on NLVR with significantly less supervision.

1 Introduction

Semantic parsing is the task of translating natural language utterances into machine-executable meaning representations, often called *programs* or *logical forms*. These logical forms can be executed against some representation of the context in which the utterance occurs, to produce a *denotation*. This setup allows for complex reasoning over contextual knowledge, and it has been successfully used in several natural language understanding problems such as question answering (Berant et al., 2013), program synthesis (Yin and Neubig, 2017) and building natural language interfaces (Suhr et al., 2018).

Recent work has focused on training semantic parses via *weak supervision* from denotations alone (Liang et al., 2011; Berant et al., 2013). This is because obtaining logical form annotations is generally expensive (although recent work has addressed this issue to some extent (Yih et al., 2016)), and not assuming full supervision lets us be agnostic about the logical form language. The second reason is more important in open-domain semantic parsing tasks where it may not be possible to arrive at a complete set of operators required by the task. However, training semantic parsers with weak supervision requires not only searching over an exponentially large space of logical forms (Berant et al., 2013; Artzi and Zettlemoyer, 2013; Pasupat and Liang, 2015; Guu et al., 2017, *inter alia*) but also dealing with *spurious* logical forms that evaluate to the correct denotation while not being semantically equivalent to the utterance. For example, if the denotations are binary, 50% of all syntactically valid logical forms evaluate to the correct answer, regardless of their semantics. This problem renders the training signal extremely noisy, making it hard for the model to learn anything without some additional guidance during search.

We introduce two innovations to improve learning from denotations. Firstly, we propose an iterative search procedure for gradually increasing the complexity of candidate logical forms for each training instance, leading to better training data and better parsing accuracy. This procedure is implemented via training our model with two interleaving objectives, one that involves searching for logical forms of limited complexity during training (online search), and another that maximizes the marginal likelihood of retrieved logical forms. Second, we include a notion of coverage over the question in the search step to guide the training algorithm towards logical forms that not only evaluate to the correct denotation, but also have some connection to the words in the utterance.

We demonstrate the effectiveness of these two techniques on two difficult reasoning tasks: WIK-

Proceedings of NAACL-HLT 2019, pages 2669–2680
Minneapolis, Minnesota, June 2 - June 7, 2019.

ITABLEQUESTIONS(WTQ) (Pasupat and Liang, 2015), an open domain task with significant lexical variation, and Cornell Natural Language Visual Reasoning (NLVR) (Suhr et al., 2017), a closed domain task with binary denotations, and thus far less supervision. We show that: 1) interleaving online search and MML over retrieved logical forms (§4) is a more effective training algorithm than each of those objectives alone; 2) coverage guidance during search (§3) is helpful for dealing with weak supervision, more so in the case of NLVR where the supervision is weaker; 3) a combination of the two techniques yields 44.3% test accuracy on WTQ, outperforming the previous best single model in a comparable setting, and 82.9% test accuracy on NLVR, outperforming the best prior model, which also relies on greater supervision.

2 Background

2.1 Weakly supervised semantic parsing

We formally define semantic parsing in a weakly supervised setup as follows. Given a dataset where the i^{th} instance is the triple $\{x_i, w_i, d_i\}$, representing a sentence x_i, the world w_i associated with the sentence, and the corresponding denotation d_i, our goal is to find y_i, the translation of x_i in an appropriate logical form language (see §5.3), such that $[\![y_i]\!]^{w_i} = d_i$; i.e., the execution of y_i in world w_i produces the correct denotation d_i. A semantic parser defines a distribution over logical forms given an input utterance: $p(Y|x_i;\theta)$.

2.2 Training algorithms

In this section we describe prior techniques for training semantic parsers with weak supervision: maximizing marginal likelihood, and reward-based methods.

2.2.1 Maximum marginal likelihood

Most work on training semantic parsers from denotations maximizes the likelihood of the denotation given the utterance. The semantic parsing model itself defines a distribution over *logical forms*, however, not *denotations*, so this maximization must be recast as a *marginalization* over logical forms that evaluate to the correct denotation:

$$\max_{\theta} \prod_{x_i, d_i \in \mathcal{D}} \sum_{y_i \in Y | [\![y_i]\!]^{w_i} = d_i} p(y_i|x_i;\theta) \quad (1)$$

This objective function is called *maximum marginal likelihood* (MML). The inner summation is in general intractable to perform during training, so it is only approximated.

Most prior work (Berant et al., 2013; Goldman et al., 2018, *inter alia*) approximate the intractable marginalization by summing over logical forms obtained via beam search during training. This typically results in frequent search failures early during training when model parameters are close to random, and in general may only yield spurious logical forms in the absence of any guidance. Since modern semantic parsers typically operate without a lexicon, new techniques are essential to provide guidance to the search procedure (Goldman et al., 2018).

One way of providing this guidance during search is to perform some kind of heuristic search up front to find a set of logical forms that evaluate to the correct denotation, and use those logical forms to approximate the inner summation (Liang et al., 2011; Krishnamurthy et al., 2017). The particulars of the heuristic search can have a large impact on performance; a smaller candidate set has lower noise, while a larger set makes it more likely that the correct logical form is in it, and one needs to strike the right balance. In this paper, we refer to the MML that does search during training as *dynamic MML*, and the one that does an offline search as *static MML*.

The main benefit of dynamic MML is that it adapts its training signal over time. As the model learns, it can increasingly focus its probability mass on a small set of very likely logical forms. The main benefit of static MML is that there is no need to search during training, so there is a consistent training signal even at the start of training, and it is typically more computationally efficient than dynamic MML.

2.2.2 Reward-based methods

When training weakly supervised semantic parsers, it is often desirable to inject some prior knowledge into the training procedure by defining arbitrary reward or cost functions. There exists prior work that use such methods, both in a reinforcement learning setting (Liang et al., 2017, 2018), and otherwise (Iyyer et al., 2017; Guu et al., 2017). In our work, we define a customized cost function that includes a coverage term, and use a Minimum Bayes Risk (MBR) (Goodman, 1996; Goel and Byrne, 2000; Smith and Eisner,

2006) training scheme, which we describe in §3.

3 Coverage-guided search

Weakly-supervised training of semantic parsers relies heavily on lexical cues to guide the initial stages of learning to good logical forms. Traditionally, these lexical cues were provided in the parser's lexicon. Neural semantic parsers remove the lexicon, however, and so need another mechanism for obtaining these lexical cues. In this section we introduce the use of coverage to inject lexicon-like information into neural semantic parsers.

Coverage is a measure of relevance of the candidate logical form y_i to the input x_i, in terms of how well the productions in y_i map to parts of x_i. We use a small manually specified lexicon as a mapping from source language to the target language productions, and define coverage of y_i as the number of productions triggered by the input utterance, according to the lexicon, that are included in y_i.

We use this measure of coverage to augment our loss function, and train using an MBR based algorithm as follows. We use beam search to train a model to minimize the expected value of a cost function $\mathcal{C}$:

$$\min_{\theta} \sum_{i=1}^{N} \mathbb{E}_{\tilde{p}(y_i|x_i;\theta)} \mathcal{C}(x_i, y_i, w_i, d_i) \qquad (2)$$

where $\tilde{p}$ is a *re-normalization*[1] of the probabilities assigned to all logical forms on the beam.

We define the cost function $\mathcal{C}$ as:

$$\mathcal{C}(x_i, y_i, w_i, d_i) = \lambda \mathcal{S}(y_i, x_i) + (1-\lambda)\mathcal{T}(y_i, w_i, d_i) \qquad (3)$$

where the function $\mathcal{S}$ measures the number of items that y_i is missing from the actions (or grammar production rules) triggered by the input utterance x_i given the lexicon; and the function $\mathcal{T}$ measures the consistency of the evaluation of y_i in w_i, meaning that it is 0 if $[\![y_i]\!]^{w_i} = d_i$, or a value e otherwise. We set e as the maximum possible value of the coverage cost for the corresponding instance, to make the two costs comparable in magnitude. λ is a hyperparameter that gives the relative weight of the coverage cost.

[1]Note that without this re-normalization, and with a -1/0 cost function based on denotation accuracy, MBR will maximize the likelihood of correct logical forms on the beam, which is equivalent to dynamic MML.

4 Iterative search

In this section we describe the iterative technique for refining the set of candidate logical forms associated with each training instance.

As discussed in §2.2, most prior work on weakly-supervised training of semantic parsers uses dynamic MML. This is particularly problematic in domains like NLVR, where the supervision signal is binary—it is very hard for dynamic MML to bootstrap its way to finding good logical forms. To solve this problem, we interleave static MML, which has a consistent supervision signal from the start of training, with the coverage-augmented MBR algorithm described in §3.

In order to use static MML, we need an initial set of candidate logical forms. We obtain this candidate set using a bounded-length exhaustive search, filtered using heuristics based on the same lexical mapping used for coverage in §3. A bounded-length search will not find logical forms for the entire training data, so we can only use a subset of the data for initial training. We train a model to convergence using static MML on these logical forms, then use that model to initialize coverage-augmented MBR training. This gives the model a good starting place for the dynamic learning algorithm, and the search at training time can look for logical forms that are longer than could be found with the bounded-length exhaustive search. We train MBR to convergence, then use beam search on the MBR model to find a new set of candidate logical forms for static MML on the training data. This set of logical forms can have a greater length than those in the initial set, because this search uses model scores to not exhaustively explore all possible paths, and thus will likely cover more of the training data. In this way, we can iteratively improve the candidate logical forms used for static training, which in turn improves the starting place for the online search algorithm.

Algorithm 1 concretely describes this process. `Decode` in the algorithm refers to running a beam search decoder that returns a set of consistent logical forms (i.e. $\mathcal{T} = 0$) for each of the input utterances. We start off with a seed dataset $\mathcal{D}^0$ for which consistent logical forms are available.

5 Datasets

We will now describe the two datasets we use in this work to evaluate our methods – Cornell NLVR

Input : Dataset $\mathcal{D} = \{X, W, D\}$; and
seed set $\mathcal{D}^0 = \{X^0, Y^0\}$ such that
$X^0 \subset X$ and $\mathcal{C}(x_i^0, y_i^0, W_i, D_i) = 0$
Output: Model parameters θ^{MBR}
Initialize dataset $\mathcal{D}^{\text{MML}} = \mathcal{D}^0$;
while *Acc*$(\mathcal{D}_{dev})$ *is increasing* **do**
  $\theta^{\text{MML}} = \texttt{MML}(\mathcal{D}^{\text{MML}})$;
  Initialize $\theta^{\text{MBR}} = \theta^{\text{MML}}$;
  Update $\theta^{\text{MBR}} = \texttt{MBR}(\mathcal{D}; \theta^{\text{MBR}})$;
  Update $\mathcal{D}^{\text{MML}} = \texttt{Decode}(\mathcal{D}; \theta^{\text{MBR}})$;
end

Algorithm 1: Iterative coverage-guided search

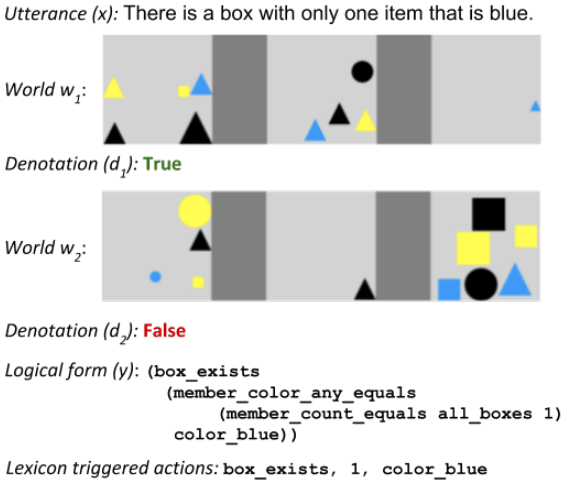

Figure 1: Example from NLVR dataset showing an utterance associated with two worlds and corresponding binary denotations. Also shown are the logical form and the actions triggered by the lexicon from the utterance.

and WIKITABLEQUESTIONS.

5.1 Cornell NLVR

Cornell NLVR is a language-grounding dataset containing natural language sentences provided along with synthetically generated visual contexts, and a label for each sentence-image pair indicating whether the sentence is true or false in the given context. Figure 1 shows two example sentence-image pairs from the dataset (with the same sentence). The dataset also comes with structured representations of images, indicating the color, shape, size, and x- and y-coordinates of each of the objects in the image. While we show images in Figure 1 for ease of exposition, we use the structured representations in this work.

Following the notation introduced in §2.1, x_i in this example is *There is a box with only one item that is blue*. The structured representations associated with the two images shown are two of the worlds (w_i^1 and w_i^2), in which x_i could be evaluated. The corresponding labels are the denotations d_i^1 and d_i^2 that a translation y_i of the sentence x_i is expected to produce, when executed in the two worlds respectively. That the same sentence occurs with multiple worlds is an important property of this dataset, and we make use of it by defining the function $\mathcal{T}$ to be 0 only if $\forall_{w_i^j, d_i^j} [\![y_i]\!]^{w_i^j} = d_i^j$.

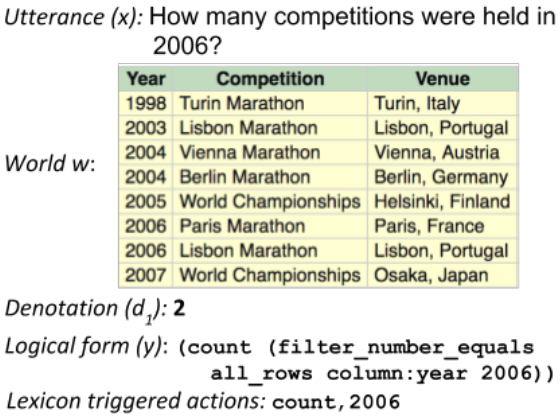

Figure 2: Example from WIKITABLEQUESTIONS dataset showing an utterance, a world, associated denotation, corresponding logical form, and actions triggered by the lexicon.

5.2 WIKITABLEQUESTIONS

WIKITABLEQUESTIONS is a question-answering dataset where the task requires answering complex questions in the context of Wikipedia tables. An example can be seen in Figure 2. Unlike NLVR, the answers are not binary. They can instead be cells in the table or the result of numerical or set-theoretic operations performed on them.

5.3 Logical form languages

For NLVR, we define a typed variable-free functional query language, inspired by the GeoQuery language (Zelle and Mooney, 1996). Our language contains six basic types: `box` (referring to one of the three gray areas in Figure 1), `object` (referring to the circles, triangles and squares in Figure 1), `shape`, `color`, `number` and `boolean`. The constants in our language are color and shape names, the set of all boxes in an image, and the set of all objects in an image. The functions in our language include those for filtering objects and boxes, and making assertions, a higher order function for handling negations, and a function for

querying objects in boxes. This type specification of constants and functions gives us a grammar with 115 productions, of which 101 are terminal productions (see Appendix A.1 for the complete set of rules in our grammar). Figure 1 shows an example of a complete logical form in our language.

For WTQ, we use the functional query language used by (Liang et al., 2018) as the logical form language. Figure 2 shows an example logical form.

5.4 Lexicons for coverage

The lexicon we use for the coverage measure described in §3 contains under 40 rules for each logical form language. They mainly map words and phrases to constants and unary functions in the target language. The complete lexicons are shown in the Appendix. Figures 1 and 2 also show the actions triggered by the corresponding lexicons for the utterances shown. We find that small but precise lexicons are sufficient to guide the search process away from spurious logical forms. Moreover, as shown empirically in §6.4, the model for NLVR does not learn much without this simple but crucial guidance.

6 Experiments

We evaluate both our contributions on NLVR and WIKITABLEQUESTIONS.

6.1 Model

In this work, we use a grammar-constrained encoder-decoder neural semantic parser for our experiments. Of the many variants of this basic architecture (see §7), all of which are essentially seq2seq models with constrained outputs and/or re-parameterizations, we choose to use the parser of Krishnamurthy et al. (2017), as it is particularly well-suited to the WIKITABLEQUESTIONS dataset, which we evaluate on.

The encoder in the model is a bi-directional recurrent neural network with Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) cells, and the decoder is a grammar-constrained decoder also with LSTM cells. Instead of directly outputting tokens in the logical form, the decoder outputs *production rules* from a CFG-like grammar. These production rules sequentially build up an abstract syntax tree, which determines the logical form. The model also has an entity linking component for producing table entities in the logical forms; this component is only applicable to WIKITABLEQUESTIONS, and we remove it when running experiments on NLVR. The particulars of the model are not the focus of this work, so we refer the reader to the original paper for more details.

In addition, we slightly modify the constrained decoding architecture from (Krishnamurthy et al., 2017) to bias the predicted actions towards those that would decrease the value of $\mathcal{S}(y_i, x_i)$. This is done using a coverage vector, $v_i^{\mathcal{S}}$ for each training instance that keeps track of the production rules triggered by x_i, and gets updated whenever one of those desired productions is produced by the decoder. That is, $v_i^{\mathcal{S}}$ is a vector of 1s and 0s, with 1s indicating the triggered productions that are yet to be produced by the decoder. This is similar to the idea of checklists used by Kiddon et al. (2016). The decoder in the original architecture scores output actions at each time step by computing a dot product of the predicted action representation with the embeddings of each of the actions. We add a weighted sum of all the actions that are yet to produced:

$$s_i^a = e^a.(p_i + \gamma * v_i^{\mathcal{S}}.E) \quad (4)$$

where s_i^a is the score of action a at time step i, e^a is the embedding of that action, p_i is the predicted action representation, E is the set of embeddings of all the actions, and γ is a learned parameter for regularizing the bias towards yet-to-be produced triggered actions.

6.2 Experimental setup

NLVR We use the standard train-dev-test split for NLVR, containing 12409, 988 and 989 sentence-image pairs respectively. NLVR contains most of the sentences occurring in multiple worlds (with an average of 3.9 worlds per sentence). We set the word embedding and action embedding sizes to 50, and the hidden layer size of both the encoder and the decoder to 30. We initialized all the parameters, including the word and action embeddings using Glorot uniform initialization (Glorot and Bengio, 2010). We found that using pretrained word representations did not help. We added a dropout (Srivastava et al., 2014) of 0.2 on the outputs of the encoder and the decoder and before predicting the next action, set the beam size to 10 both during training and at test time, and trained the model using ADAM (Kingma and Ba, 2014) with a learning rate of 0.001. All the hyperparameters are tuned on the validation set.

WIKITABLEQUESTIONS This dataset comes with five different cross-validation folds of training data, each containing a different 80/20 split for training and development. We first show results aggregated from all five folds in §6.3, and then show results from controlled experiments on fold 1. We replicated the model presented in Krishnamurthy et al. (2017), and only changed the training algorithm and the language used. We used a beam size of 20 for MBR during training and decoding, and 10 for MML during decoding, and trained the model using Stochastic Gradient Descent (Kiefer et al., 1952) with a learning rate of 0.1, all of which are tuned on the validation sets.

Specifics of iterative search For our iterative search algorithm, we obtain an initial set of candidate logical forms in both domains by exhaustively searching to a depth of 10[2]. During search we retrieve the logical forms that lead to the correct denotations in all the corresponding worlds, and sort them based on their coverage cost using the coverage lexicon described in §5.4, and choose the top-k[3]. At each iteration of the search step in our iterative training algorithm, we increase the maximum depth of our search with a step-size of 2, finding more complex logical forms and covering a larger proportion of the training data. While exhaustive search is prohibitively expensive beyond a fixed number of steps, our training process that uses beam search based approximation can go deeper.

Implementation We implemented our model and training algorithms within the AllenNLP (Gardner et al., 2018) toolkit. The code and models are publicly available at `https://github.com/allenai/iterative-search-semparse`.

6.3 Main results

WIKITABLEQUESTIONS Table 1 compares the performance of a single model trained using Iterative Search, with that of previously published single models. We excluded ensemble models since there are differences in the way ensembles are built for this task in previous work, either in terms of size or how the individual models were chosen. We show both best and average (over 5 folds) single model performance from Liang et al. (2018) (Memory Augmented Policy Optimization). The best model was chosen based on performance on the development set. Our single model performances are computed in the same way. Note that Liang et al. (2018) also use a lexicon similar to ours to prune the seed set of logical forms used to initialize their memory buffer.

Approach	Dev	Test
Pasupat and Liang (2015)	37.0	37.1
Neelakantan et al. (2017)	34.1	34.2
Haug et al. (2018)	-	34.8
Zhang et al. (2017)	40.4	43.7
Liang et al. (2018) (MAPO) (avg.)	42.3	43.1
Liang et al. (2018) (MAPO) (best)	42.7	43.8
Iterative Search (avg.)	42.1	43.9
Iterative Search (best)	**43.1**	**44.3**

Table 1: Comparison of single model performances of Iterative Search with previously reported single model performances

Algorithm	Dev acc.	Test acc.
MAPO	42.1	42.7
Static MML	40.0	42.2
Iterative MML	42.5	43.1
Iterative Search	**43.0**	**43.8**

Table 2: Comparison of iterative search with static MML, iterative MML, and the previous best result from (Liang et al., 2018), all trained on the official split 1 of WIKITABLEQUESTIONS and tested on the official test set.

In Table 2, we compare the performance of our iterative search algorithm with three baselines: 1) Static MML, as described in §2.2.1 trained on the candidate set of logical forms obtained through the heuristic search technique described in §6.2; 2) Iterative MML, also an iterative technique but unlike iterative search, we skip MBR and iteratively train static MML models while increasing the number of decoding steps; and 3) MAPO (Liang et al., 2018), the current best published system on WTQ. All four algorithms are trained and evaluated on the first fold, use the same language, and the bottom three use the same model and the same set of logical forms used to train static MML.

[2] It was prohibitively expensive to search beyond depth of 10.

[3] k is a hyperparameter that is chosen on the dev set at each iteration in iterative search, and is typically 10 or 20

NLVR In Table 3, we show a comparison of the performance of our *iterative coverage-guided search* algorithm with the previously published approaches for NLVR. The first two rows correspond to models that are not semantic parsers. This shows that semantic parsing is a promising direction for this task. The closest work to ours is the weakly supervised parser built by (Goldman et al., 2018). They build a lexicon similar to ours for mapping surface forms in input sentences to abstract clusters. But in addition to defining a lexicon, they also manually annotate complete sentences in this abstract space, and use those annotations to perform data augmentation for training a supervised parser, which is then used to initialize a weakly supervised parser. They also explicitly use the abstractions to augment the beam during decoding using caching, and a separately-trained discriminative re-ranker to re-order the logical forms on the beam. As a discriminative re-ranker is orthogonal to our contributions, we show their results with and without it, with "Abs. Sup." being more comparable to our work. Our model, which uses no data augmentation, no caching during decoding, and no discriminative re-ranker, outperforms their variant without reranking on the public test set, and outperforms their best model on the hidden test set, achieving a new state-of-the-art result on this dataset.

6.4 Effect of coverage-guided search

To evaluate the contribution of coverage-guided search, we compare the the performance of the NLVR parser in two different settings: with and without coverage guidance in the cost function. We also compare the performance of the parser in the two settings, when initialized with parameters from an MML model trained to maximize the likelihood of the set of logical forms obtained from exhaustive search. Table 4 shows the results of this comparison. We measure accuracy and consistency of all four models on the publicly available test set, using the official evaluation script. Consistency here refers to the percentage of logical forms that produce the correct denotation in all the corresponding worlds, and is hence a stricter metric than accuracy. The cost weight (λ in Equation 3) was tuned based on validation set performance for the runs with coverage, and we found that $\lambda = 0.4$ worked best.

It can be seen that both with and without initialization, coverage guidance helps by a big margin, with the gap being even more prominent in the case where there is no initialization. When there is neither coverage guidance nor a good initialization, the model does not learn much from unguided search and get a test accuracy not much higher than the majority baseline of 56.2%.

We found that coverage guidance was not as useful for WTQ. The average value of the best performing λ was around 0.2, and higher values neither helped nor hurt performance.

6.5 Effect of iterative search

To evaluate the effect of *iterative search*, we present the accuracy numbers from the search (S) and maximization (M) steps from different iterations in Tables 5 and 6, showing results on NLVR and WTQ, respectively. Additionally, we also show number of decoding steps used at each iterations, and the percentage of sentences in the training data for which we were able to obtain consistent logical forms from the S step, the set that was used in the M step of the same iteration. It can be seen in both tables that a better MML model gives a better initialization for MBR, and a better MBR model results in a larger set of utterances for which we can retrieve consistent logical forms, thus improving the subsequent MML model. The improvement for NLVR is more pronounced (a gain of 21% absolute) than for WTQ (a gain of 3% absolute), likely because the initial exhaustive search provides a much higher percentage of spurious logical forms for NLVR, and thus the starting place is relatively worse.

Complexity of Logical Forms We analyzed the logical forms produced by our iterative search algorithm at different iterations to see how they differ. As expected, for NLVR, allowing greater depths lets the parser explore more complex logical forms. Table 7 shows examples from the validation set that indicate this trend.

7 Related Work

Most of the early methods used for training semantic parsers required the training data to come with annotated logical forms (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005). The primary limitation of such methods is that manually producing these logical forms is expensive, making it hard to scale these methods across domains.

Approach	Dev.		Test-P		Test-H	
	Acc.	Cons.	Acc.	Cons.	Acc.	Cons.
MaxEnt (Suhr et al., 2017)	68.0	-	67.7	-	67.8	-
BiATT-Pointer (Tan and Bansal, 2018)	74.6	-	73.9	-	71.8	-
Abs. Sup. (Goldman et al., 2018)	84.3	66.3	81.7	60.1	-	-
Abs. Sup. + ReRank (Goldman et al., 2018)	85.7	67.4	84.0	65.0	82.5	63.9
Iterative Search	85.4	64.8	82.4	61.3	**82.9**	**64.3**

Table 3: Comparison of our approach with previously published approaches. We show accuracy and consistency on the development set, and public (Test-P) and hidden (Test-H) test sets.

	No coverage		+ coverage	
	Acc.	Cons.	Acc.	Cons.
No init.	56.4	12.0	73.9	43.6
MML init.	77.7	51.1	80.7	56.4

Table 4: Effect of coverage guidance on NLVR parsers trained with and without initialization from an MML model. Metrics shown are accuracy and consistency on the public test set.

Iter.	Length	% cov.	Step	Dev. Acc
0	10	51	M	64.0
1	12	65	S	81.6
			M	76.5
2	14	65	S	82.7
			M	81.8
3	16	73	S	85.4
			M	83.1
4	18	75	S	84.7
			M	81.2

Table 5: Effect of iterative search (S) and maximization (M) on NLVR. % cov. is the percentage of training data for which the S step retrieves consistent logical forms.

Iter.	Length	% cov.	Step	Dev. Acc
0	10	83.3	M	40.0
1	12	70.2	S	42.5
			M	42.5
2	14	71.3	S	43.1
			M	42.7
3	16	71.0	S	42.8
			M	42.5
4	18	71.0	S	43.0
			M	42.7

Table 6: Iterative search on WIKITABLEQUESTIONS. M and S refer to Maximization and Search steps.

More recent research has focused on training semantic parsers with *weak supervision* (Liang et al., 2011; Berant et al., 2013), or trying to automatically infer logical forms from denotations (Pasupat and Liang, 2016). However, matching the performance of a fully supervised semantic parser with only weak supervision remains a significant challenge (Yih et al., 2016).

The main contributions of this work deal with training semantic parsers with weak supervision, and we gave a detailed discussion of related training methods in §2.2.

We evaluate our contributions on the NLVR and WIKITABLEQUESTIONS datasets. Other work that evaluates on on these datasets include Goldman et al. (2018), Tan and Bansal (2018), Neelakantan et al. (2017), Krishnamurthy et al. (2017), Haug et al. (2018), and (Liang et al., 2018). These prior works generally present modeling contributions that are orthogonal (and in some cases complementary) to the contributions of this paper. There has also been a lot of recent work on neural semantic parsing, most of which is also orthogonal to (and could probably benefit from) our contributions (Dong and Lapata, 2016; Jia and Liang, 2016; Yin and Neubig, 2017; Krishnamurthy et al., 2017; Rabinovich et al., 2017). Recent attempts at dealing with the problem of spuriousness include Misra et al. (2018) and Guu et al. (2017).

Coverage has recently been used in machine translation (Tu et al., 2016) and summarization (See et al., 2017). There have also been many methods that use coverage-like mechanisms to give lexical cues to semantic parsers. Goldman et al. (2018)'s abstract examples is the most recent and related work, but the idea is also related to lexicons in pre-neural semantic parsers (Kwiatkowski et al., 2011).

0	*There is a tower with four blocks* `(box_exists (member_count_equals all_boxes 4))`
1	*Atleast one black triangle is not touching the edge* `(object_exists (black (triangle ((negate_filter touch_wall) all_objects))))`
2	*There is a yellow block as the top of a tower with exactly three blocks.* `(object_exists (yellow (top (object_in_box (member_count_equals all_boxes 3)))))`
3	*The tower with three blocks has a yellow block over a black block* `(object_count_greater_equals (yellow (above (black (object_in_box (member_count_equals all_boxes 3))))) 1)`

Table 7: Complexity of logical forms produced at different iterations, from iteration 0 to iteration 3; each logical form could not be produced at the previous iterations

8 Conclusion

We have presented a new technique for training semantic parsers with weak supervision. Our key insights are that lexical cues are crucial for guiding search during the early stages of training, and that the particulars of the approximate marginalization in maximum marginal likelihood have a large impact on performance. To address the first issue, we used a simple coverage mechanism for including lexicon-like information in neural semantic parsers that do not have lexicons. For the second issue, we developed an iterative procedure that alternates between statically-computed and dynamically-computed training signals. Together these two contributions greatly improve semantic parsing performance, leading to new state-of-the-art results on NLVR and WIKITABLEQUESTIONS. As these contributions are to the learning algorithm, they are broadly applicable to many models trained with weak supervision. One potential future work direction is investigating whether they extend to other structured prediction problems beyond semantic parsing.

Acknowledgments

We would like to thank Jonathan Berant and Noah Smith for comments on earlier drafts and Chen Liang for helping us with implementation details of MAPO. Computations on beaker.org were supported in part by credits from Google Cloud.

References

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *ACL'16*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Vaibhava Goel and William J Byrne. 2000. Minimum bayes-risk automatic speech recognition. *Computer Speech & Language*, 14(2):115–135.

Omer Goldman, Veronica Latcinnik, Udi Naveh, Amir Globerson, and Jonathan Berant. 2018. Weakly-supervised semantic parsing with abstract examples. In *ACL*.

Joshua Goodman. 1996. Parsing algorithms and metrics. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 177–183. Association for Computational Linguistics.

Kelvin Guu, Panupong Pasupat, Evan Zheran Liu, and Percy Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Association for Computational Linguistics (ACL)*.

Till Haug, Octavian-Eugen Ganea, and Paulina Grnarova. 2018. Neural multi-step reasoning for question answering on semi-structured tables. In *ECIR*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Mohit Iyyer, Wen tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Association for Computational Linguistics*.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *ACL'16*.

Chloé Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally coherent text generation with neural checklist models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339.

Jack Kiefer, Jacob Wolfowitz, et al. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526.

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*, pages 1512–1523. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 23–33.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis with generalization. *arXiv preprint arXiv:1807.02322*.

Percy Liang, Michael I Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *ACL*.

Dipendra Misra, Ming-Wei Chang, Xiaodong He, and Wen tau Yih. 2018. Policy shaping and generalized update equations for semantic parsing from denotations. In *EMNLP*.

Arvind Neelakantan, Quoc V Le, Martin Abadi, Andrew McCallum, and Dario Amodei. 2017. Learning a natural language interface with neural programmer. In *ICLR*.

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)*, volume 1, pages 1470–1480.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 23–32.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *ACL*.

Abigail See, Peter J Liu, and Christopher D Manning. 2017. Get to the point: Summarization with pointer-generator networks. *ACL*.

David A Smith and Jason Eisner. 2006. Minimum risk annealing for training log-linear models. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 787–794. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 2238–2249.

Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. 2017. A corpus of natural language for visual reasoning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 217–223.

Hao Tan and Mohit Bansal. 2018. Object ordering with bidirectional matchings for visual reasoning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Zhaopeng Tu, Zhengdong Lu, Yang Liu, Xiaohua Liu, and Hang Li. 2016. Modeling coverage for neural machine translation. *ACL*.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *ACL*.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *ACL'17*.

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-Volume 2*, pages 1050–1055. AAAI Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI'05*.

Yuchen Zhang, Panupong Pasupat, and Percy Liang. 2017. Macro grammars and holistic triggering for efficient semantic parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1214–1223.

A Logical form language and lexicon for NLVR

Basic Types: `bool (t)`, `box (b)`, `object (o)`, `shape (s)`, `color (c)`, `number (n)`

In the grammar and lexicon that follow we use the following placeholders,

`quantifier` ∈ {`any`, `all`, `none`}
`comparator` ∈ {`equals`, `not_equals`, `lesser`, `lesser_equals`, `greater`, `greater_equals`}
`color` ∈ {`yellow`, `blue`, `black`}
`shape` ∈ {`square`, `triangle`, `circle`}
`size` ∈ {`big`, `medium`, `small`}
`location` ∈ {`above`, `below`, `top`, `left`, `right`, `bottom`, `corner`, `wall`}
`number` ∈ {1...9}

A.1 Grammar

Constants

```
b -> all_boxes
c -> color_black,
     color_blue, color_yellow
n -> 1, 2, ..., 9
o -> all_objects
s -> shape_circle,
     shape_square, shape_triangle
```

Object filtering functions

```
<o,o> -> [location], [color],
 [shape], [size], same_color,
 same_shape, touch_object,
 touch_bottom, touch_top,
 touch_left, touch_right,
 touch_corner, touch_wall,
 member_color_[quantifier]_equals
<b,<n,b>> ->
 member_count_[comparator]
 member_color_count_[comparator],
 member_shape_count_[comparator]
<b,b> -> member_color_different,
 member_color_same,
 member_shape_different,
 member_shape_same
```

Box filtering functions

```
<b,<s,b>> ->
  member_shape_[quantifier]_equals
<b,<c,b>> ->
  member_color_[quantifier]_equals
<b,<n,b>> ->
  member_count_[comparator]
  member_color_count_[comparator],
  member_shape_count_[comparator]
<b,b> -> member_color_different,
 member_color_same,
 member_shape_different,
 member_shape_same
```

Assertion functions

```
<b,t> -> box_exists
<o,t> -> object_exists
<b,<n,t>> -> box_count_[comparator]
<o,<c,t>> ->
  object_color_[quantifier]_equals
<o,<s,t>> ->
  object_shape_[quantifier]_equals
<o,<n,t>> ->
  object_color_count_[comparator],
  object_shape_count_[comparator],
  object_count_[comparator]
```

Other functions

```
<b,o> -> object_in_box
<<o,o>,<o,o>> -> negate_filter
```

A.2 Lexicon for NLVR

there is a box → `box_exists`
there is a [other] → `object_exists`
box . . . [color] → `color_[color]`
box . . . [shape] → `shape_[shape]`
not → `negate_filter`
contains → `object_in_box`
touch . . . [location] → `touch_[location]`
[location] → `[location]`
[shape] → `[shape]`
[color] → `[color]`
[size] → `[size]`
[number] → `[number]`

B Logical form language and lexicon for WIKITABLEQUESTIONS

We use the language from Liang et al. (2018). For coverage, in addition to triggering productions for numbers, and column names and cell strings in the table, we use the following lexicon for coverage.

B.1 Lexicon for WIKITABLEQUESTIONS

at least → `filter`$_{\geq}$
[greater|larger|more] than → `filter`$_{\geq}$
at most → `filter`$_{\leq}$
no [greater|larger|more] than → `filter`$_{\leq}$
[next|below|after] → `next`
[previous|above|before] → `previous`
[first|top] → `top`
[last|bottom] → `bottom`
same → `same_as`
total → `sum`
difference → `diff`
average → `average`
[least|smallest|lowest|smallest] → `argmin`
[most|longest|highest|largest] → `argmax`
[what|when] . . . [last|least] → `min`
[what|when] . . . [first|most] → `max`
how many → `count`