Weakly-supervised Semantic Parsing with Abstract Examples

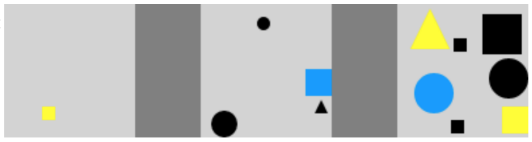
Omer Goldman[*], **Veronica Latcinnik**[*], **Udi Naveh**[*], **Amir Globerson, Jonathan Berant**
Tel-Aviv University
{omergoldman@mail,veronical@mail,ehudnave@mail,gamir@post,joberant@cs}.tau.ac.il

Abstract

Semantic parsers translate language utterances to programs, but are often trained from utterance-denotation pairs only. Consequently, parsers must overcome the problem of *spuriousness* at training time, where an incorrect program found at search time accidentally leads to a correct denotation. We propose that in small well-typed domains, we can semi-automatically generate an abstract representation for examples that facilitates information sharing across examples. This alleviates spuriousness, as the probability of randomly obtaining a correct answer from a program decreases across multiple examples. We test our approach on CNLVR, a challenging visual reasoning dataset, where spuriousness is central because denotations are either TRUE or FALSE, and thus random programs have high probability of leading to a correct denotation. We develop the first semantic parser for this task and reach 83.5% accuracy, a 15.7% absolute accuracy improvement compared to the best reported accuracy so far.

1 Introduction

The goal of semantic parsing is to map language utterances to executable programs. Early work on statistical learning of semantic parsers utilized supervised learning, where training examples included pairs of language utterances and programs. (Zelle and Mooney, 1996; Kate et al., 2005; Zettlemoyer and Collins, 2005, 2007). However, collecting such training examples at scale has quickly turned out to be difficult, because expert annotators that are familiar with formal languages are required. This has led to a significant body of work on weakly-supervised semantic parsing (Clarke et al., 2010; Liang et al., 2011; Krishnamurthy and Mitchell, 2012; Kwiatkowski et al., 2013; Berant et al., 2013; Cai and Yates, 2013; Artzi and Zettlemoyer, 2013), where training examples correspond to utterance-denotation pairs. A *denotation* is the result of executing a program against the target knowledge-base (KB) (see Figure 1). Naturally, collecting denotations is much easier, because it can be performed by non-experts.



k :[[{y_loc: ..., color: 'Black', type: 'square', x_loc: ... size: 20}, ...]]

x :*There is a small yellow item not touching any wall*

y :`True`

z :`Exist(Filter(ALL_ITEMS, λx.And(And(IsYellow(x), IsSmall(x)), Not(IsTouchingWall(x, Side.Any))))))`

Figure 1: Overview of our setup for visual reasoning. Given an image I that was rendered from a KB k and an utterance x, our goal is to parse x to the correct program z that results in the denotation y. Our training data includes (x, k, y) triplets.

Training semantic parsers from denotations raises two fundamental learning challenges: (a) *Search*: When training from denotations, the algorithm must learn to search through the huge space of programs, to find ones that execute to the correct denotations. This is a difficult search problem due to the combinatorial nature of the search space. (b) *Spuriousness*: Incorrect programs can easily lead to the right denotation, and thus the learner can go astray based on these programs. Of the two mentioned problems, spuriousness has attracted relatively less attention (Pasupat and Liang, 2016; Guu et al., 2017).

Recently, the Cornell Natural Language for Visual Reasoning corpus (CNLVR) was released (Suhr et al., 2017), and presented a difficult problem of spuriousness. In this task, an image con-

[*] Authors equally contributed to this work.

taining three boxes is shown, and in each box there are several objects of various shapes, colors and sizes. Each image is paired with a complex natural language utterance, and the goal is to determine whether the statement is true or false given the image (see Figure 1). The task comes in two flavors, where in one the input is the image itself (pixels), and in the other the input is the KB from which the image was synthesized. Because the images are fairly simple, the vision problem of translating the image to the structured KB is relatively simple. Given the input KB, it is easy to view CNLVR as a semantic parsing problem: our goal is to translate language utterances into programs that will be executed against the KB to determine their correctness (Johnson et al., 2017b; Hu et al., 2017). Because there are only two return values, it is easy to generate incorrect programs that execute to the right denotation, and thus the problem of spuriousness is significant compared to previous datasets.

In this paper, we present the first semantic parser for the CNLVR dataset. We develop a formal language for visual reasoning, inspired by Johnson et al. (2017b), and a semantic parser based on recent work by Guu et al. (2017). Our main insight is that in a small and well-typed domain, we can manually cluster a small number of lexical items and generate abstract representations for utterances and programs. These representations alleviate the challenges of search and spuriousness inherent to weakly-supervised semantic parsing. By creating abstract examples, we can identify similar examples and reward programs that are correct in multiple examples, thereby substantially reducing the problem of spuriousness. This can also be done for partial programs at search time, which helps combat the search challenge. Moreover, we can generate thousands of utterance-program pairs automatically from a small number of manually created abstract examples, and train a supervised parser that will provide a good starting point for the search process in the weakly-supervised setup.

Our method allows us to train a semantic parser from weak supervision, and obtain an accuracy of 83.5%, a 15.7% absolute accuracy improvement compared to state-of-the-art. All our code is publicly available at `https://github.com/udiNaveh/nlvr_tau_nlp_final_proj`.

2 Setup

Problem Statement Our task is as follows: given a training set of N examples $\{(x_i, k_i, y_i)\}_{i=1}^{N}$, where x_i is a language utterance, k_i is a KB describing objects in an image and $y_i \in \{\text{TRUE}, \text{FALSE}\}$ denotes whether the utterance is true or false in the context of the KB, our goal is to learn a semantic parser that will map new utterances-KB pairs (x, k) to a program z that is executed to the correct denotation y (see Figure 1)

Programming language The original KBs in the CNLVR dataset describe an image as three sets of objects, where each object has a color, shape, size and location in absolute coordinates. We define a programming language over the KB that is more amenable to spatial reasoning, inspired by work on the CLEVR dataset (Johnson et al., 2017b). This programming language provides access to functions that allow us to check the size, shape, and color of an object, to check whether it is touching a wall, to obtain sets of items that are above and below a certain set of items, etc.[1]

More formally, a program is a sequence of tokens describing a possibly recursive sequence of function applications in prefix notation. Each token is either a function with fixed arity (all functions have either one or two arguments), a constant, a variable or a λ term used to define Boolean functions. Functions, constants and variables have one the following atomic types: `Int`, `Bool`, `Item`, `Size`, `Shape`, `Color`, `Side` (sides of a box in the image); or a composite type `Set(?)`, and `Func(?,?)`. Valid programs have the type `Bool`. Table 1 provides two examples for utterances and their correct program in the context of an image. Note that we provide commas and parenthesis in the table for readability, but they are not part of the language. We provide a full description of all program tokens, their arguments and return types in Appendix A.

Unlike CLEVR, CNLVR requires substantial set-theoretic reasoning (utterances refer to various aspects of sets of items in one of the three boxes in the image), which required extending the language described by Johnson et al. (2017b) to include set operators and lambda abstraction. We manually

[1] We leave the problem of learning the programming language functions from the original KB for future work.

x: *"There is a small yellow item not touching any wall."*	
z: `Exist(Filter(ALL_ITEMS, λx.And(And(IsYellow(x), IsSmall(x)), Not(IsTouchingWall(x, Side.Any)))))`	
x: *"One tower has a yellow base."*	
z: `≤(1, Count(Filter(ALL_ITEMS, λx.And(IsYellow(x), IsBottom(x)))))`	

Table 1: Examples for utterance-program pairs. Commas and parenthesis are provided for readability only.

sampled 100 training examples from the training data and estimate that roughly 95% of the utterances in the training data can be expressed with this programming language.

3 Model

We base our model on the semantic parser presented by Guu et al. (2017). In their work, they use an standard encoder-decoder architecture (Sutskever et al., 2014) to define a distribution $p_\theta(z \mid x, k)$. The utterance x is encoded with a bidirectional LSTM (Hochreiter and Schmidhuber, 1997) that creates a contextualized representation h_i for every utterance token x_i, and the decoder is a feed-forward network combined with attention mechanism over the encoder outputs (Bahdanau et al., 2015). The feed-forward decoder takes as input the last K tokens that were decoded.

More formally the probability of a program is the product of the probability of its tokens given the history and KB: $p_\theta(z \mid x, k) = \prod_t p_\theta(z_t \mid x, z_{1:t-1}, k)$ (we will drop the KB k when it is clear from context), and the probability of a decoded token is computed as follows. First, an utterance representation $\hat{x}$ is computed by the Bi-LSTM encoder:

$$\begin{aligned} h_i^F &= \text{LSTM}(h_{i-1}^F, \phi_x(x_i)), \\ h_i^B &= \text{LSTM}(h_{i+1}^B, \phi_x(x_i)), \\ h_i &= [h_i^F; h_i^B], \\ \hat{x} &= [h_{|x|}^F; h_1^B], \end{aligned}$$

where $\phi_x(\cdot)$ provides utterance embeddings and ';' corresponds to concatenation. Then decoding produces the program token by token:

$$\begin{aligned} q_t &= \text{relu}(W_q[\hat{x}; z_{t-K-1:t-1}]), \\ \alpha_t &\propto \exp(q_t^\top W_\alpha h_i), \\ c_t &= \sum_i \alpha_i \cdot h_i, \\ p_\theta(z_t \mid x, z_{1:t-1}) &\propto \exp(\phi_z(z_t)^\top W_s[q_t; c_t]), \end{aligned}$$

where $\phi_z(\cdot)$ provides program embeddings, $z_{i:j} = (z_i, \ldots, z_j)$, and the matrices W_q, W_α, W_s are learned parameters (along with the LSTM parameters).

Search As explained, searching through the large space of programs is a fundamental challenge in weakly-supervised semantic parsing. To combat this challenge we apply several techniques. First, we use beam search at decoding time and when training from weak supervision (see Section 4), as was done in prior work (Liang et al., 2017; Guu et al., 2017). At each decoding step we maintain a beam of B program prefixes of length n, expand them exhaustively to programs of length $n+1$ and keep the top B program prefixes with highest model probability.

Second, we utilize our semantic typing system to only construct programs that are syntactically valid, and substantially prune the program search space (similar to the type constraints introduced by Krishnamurthy et al. (2017)). We maintain a stack that keeps track of the expected semantic type at each decoding step. The stack is initialized with the type `Bool`. Then, at each decoding step, only tokens that return the semantic type at the top of the stack are allowed, the stack is popped, and if the decoded token is a function, the semantic types of its arguments are pushed to the stack. This dramatically reduces the search space and guarantees that only syntactically valid programs will be produced. Figure 2 illustrates the state of the stack when decoding a program for an input utterance.

Given the constrains on valid programs, our model is defined as:

$$\begin{aligned} p'_\theta(z \mid x) &= \prod_t p'_\theta(z_t \mid x, z_{1:t-1}), \\ p'_\theta(z_t \mid x, z_{1:t-1}) &= \frac{p_\theta(z_t \mid x, z_{1:t-1}) \cdot \mathbb{1}(z_t \mid z_{1:t-1})}{\sum_{z'} p_\theta(z' \mid x, z_{1:t-1}) \cdot \mathbb{1}(z' \mid z_{1:t-1})}, \end{aligned}$$

where $\mathbb{1}(z_t \mid z_{1:t-1})$ indicates whether a certain program token is valid given the program prefix.

Program re-ranking Our model is a locally normalized model that provides a distribution for every token emitted by the decoder. We noticed that often programs generated on the final beam do not cover well all the words that occur in the

Figure 2: An example for the state of the type stack s while decoding a program z for an utterance x.

Utterance	Program	Cluster	#
"yellow"	`IsYellow`	`C-Color`	3
"big"	`IsBig`	`C-Size`	3
"square"	`IsSquare`	`C-Shape`	4
"3"	`3`	`C-Num`	9
"exactly"	`Equal`	`C-QuantMod`	5
"top"	`IsTop`	`C-Location`	2
"above"	`GetAbove`	`C-SpaceRel`	9
		Total:	**35**

Table 2: Example mappings from utterance tokens to program tokens for the seven clusters used in the abstract representation. The rightmost column counts the number of mapping in each cluster, resulting in a total of 26 mappings.

utterance. Therefore, we optionally apply a re-ranker that re-ranks the beam programs according to the following heuristic. For every program, we count how many of the utterance tokens the program covers, based on a small lexicon that maps partial programs to utterance phrases. Then, we return the program that has the best coverage. In Section 5 we show this results in a small boost to the final performance of the parser. In the future we plan to train a globally normalized re-ranker that will re-rank the programs in the beam and replace this heuristic.

4 Learning from Abstract Examples

The main premise of this work is that in small, well-typed domains such as visual reasoning, the main challenge is handling language compositionality, as utterances can be complex. Conversely, the problem of mapping lexical items to functions and constants in the programming language is limited and can be substantially reduced using a handful of manually defined rules. Thus, if we abstract away from the actual utterance into a more general representation we can more easily generalize across examples in datasets with a small number of examples. Consider the utterances:

1. *"There are exactly 3 yellow squares touching the wall."*

2. *"There are at least 2 blue circles touching the wall."*

While the surface forms are different in these two utterances, at an abstract level they are quite similar and we it would be useful to leverage this similarity at training time.

We therefore define an abstract representation for utterances and logical forms that is suitable for spatial reasoning. We define seven abstract clusters (see Table 2) that correspond to the main spatial concepts in our domain. Then, we associate each cluster with a small lexicon that contains language-program token pairs associated with this cluster. Table 2 shows the seven clusters we use, with an example for an utterance-program token pair from the cluster, and the number of mappings in each cluster. In total, 35 mappings are used to define abstract representations.

We now show how abstract examples can be employed to develop a rule-based semantic parser, a supervised semantic parser, and to combat the spuriousness and search challenges in a weakly-supervised semantic parser.

Rule-based semantic parser To examine the diversity in the dataset, we created abstract representations for all 3,962 utterances in the training examples by mapping utterance tokens to their cluster label. We counted how many distinct abstract utterances exist, and found that 200 abstract utterances cover roughly half of the training examples in the original training set.

To create a rule-based parser, we manually annotated 106 abstract utterances with their corresponding abstract program (including alignment between abstract tokens in the utterance and program). For example, the aforementioned utterances are both mapped to the abstract program `C-QuantMod(C-Num, Count(Filter(ALL_ITEMS, λx. And (And (IsC-Color(x), IsC-Shape(x), IsTouchingWall(x))))))`.

This defines a rule-based semantic parser: Given a new utterance x, we create its abstract representation $\bar{x}$, and if it exactly matches one of our manually annotated utterances, we map it to

its corresponding abstract program $\bar{z}$, replace the abstract program tokens with the aligned corresponding program tokens, and obtain a final program z.

Supervised semantic parser through data augmentation While the rule-based semantic parser has high precision and gauges the amount of structural variance in the data, it cannot generalize beyond observed examples.

Because our clusters are small and semantically coherent, we can generate correct examples from abstract examples by sampling pairs of utterance-program tokens for each cluster. This is equivalent to a synchronous context-free grammar (Chiang, 2005) that has a rule for generating each manually-annotated abstract utterance-program pair, and rules for generating synchronously utterance and program tokens from the seven clusters.

We generated 4,962 (x, z) examples using this method and trained a standard supervised semantic parser by maximizing $\log p'_\theta(z \mid x)$ in the model described above. Our goal is to examine whether by training on this dataset we can generalize beyond the rule-based semantic parser.

4.1 Weakly-supervised semantic parser

To train a weakly-supervised semantic parser we follow Guu et al. (2017) and treat the program z as a latent variable that is approximately marginalized. Our objective is to maximize:

$$\begin{aligned} p(y \mid x) &= \sum_{z \in \mathcal{Z}} p'_\theta(z \mid x) \cdot p(y \mid z, x) \\ &= \sum_{z \in \mathcal{Z}} p'_\theta(z \mid x) \cdot R(z, y) \\ &\approx \sum_{z \in \hat{\mathcal{Z}}} p'_\theta(z \mid x) \cdot R(z, y), \end{aligned}$$

where $\mathcal{Z}$ is the space of all programs and $\hat{\mathcal{Z}}$ is a subset of all programs that is found by the process of beam search.

Importantly, the probability of obtaining the denotation y given x and z is defined by a binary reward function $R(z, y)$ that is 1 iff z is "correct". In most semantic parsers z is defined to be correct if when executing it it results in the correct denotation y. This goes a long way because often the probability of generating the correct denotation y by a randomly generated z is low. However, in CNLVR, we observe binary denotations, and so spuriousness is a central problem as previously explained. To alleviate the spuriousness problem we utilize an interesting property of the data: the same utterance appears 4 times with 4 different images. If a program is spurious it is likely that it will yield the wrong denotation in one of those 4 images. Thus, we can re-define each training example to be $(x, \{(k_j, y_j)\}_{j=1}^4)$, where each utterance x is paired with 4 different KBs and the denotations of the utterance with respect to these KBs. Then, we maximize $p(\{y_j\}_{j=1}^4 \mid x, \{k_j\}_{j=1}^4)$ by maximizing the objective above, except that $R(z, \{y_j\}_{j=1}^4) = 1$ iff the denotation of z is correct for **all** four KBs. This dramatically reduces the problem of spuriousness, as the chance of randomly obtaining a correct denotation goes down from $\frac{1}{2}$ to $\frac{1}{16}$. This approach is reminiscent of the method proposed by Pasupat and Liang (2016), where random permutations of Wikipedia tables are shown to crowdsourcing workers to eliminate spurious programs over those tables.

4.2 Tackling spuriousness by caching abstract examples

Our approach for tackling spuriousness described above hinges on the fact that the same utterance appears in the dataset multiple times. However, as explained above, many examples are similar to one another at an abstract level even if they are not identical. Thus, a natural idea is to combat spuriousness by sharing information across utterances that have the same abstract representation.

We construct a cache that maps abstract utterances to abstract programs. After obtaining a beam of final programs for every training example utterance x, we add to the cache every abstract utterance-program pair $(\bar{x}, \bar{z})$, and record whether it obtained positive reward. Thus, for every $(\bar{x}, \bar{z})$ pair we can estimate a probability p_{sp} for whether it obtains positive reward.

To construct an abstract example from from an utterance-program pair in the beam we perform the following procedure. First, we create an abstract utterance by replacing utterance tokens with their cluster label, as in the rule-based semantic parser. Then, we go over every program token, and replace it with an abstract cluster if the utterance contains a token that is mapped to this program token according to one of the 35 mappings from Table 2. This also provides an alignment from ab-

stract program tokens to abstract utterance tokens that is necessary when utilizing the cache.

We propose two variants for taking advantage of the constructed cache:

1. *Full program retrieval:* For every utterance x, we construct an abstract utterance $\bar{x}$, retrieve from the cache the top-D abstract programs $\bar{z}$ based on p_{sp}, compute the programs z using the alignments from program tokens to utterance tokens, and add those D programs to the final beam.

2. *Program prefix retrieval:* At every decoding step, let Z_t be the beam of decoded programs at step t and Z_{t+1} be the beam of decoded programs computed at step $t+1$. For every utterance x, we find the top-D cashed programs $\bar{Z}$ as in the former variant. Then at each step t, for every abstract program $\bar{z} \in \bar{Z}$ we add $\bar{z}_{1:t}$ to Z_t, and exhaustively search over their possible continuations as well when constructing Z_{t+1}. This allows the parser to potentially construct new programs that are not in the cache already in subsequent steps. This combats spuriousness but also the search challenge, because we add to the beam promising program prefixes that might have fallen off of it in early decoding steps.

Liang et al. (2017) also suggested to use a cache for adding correct programs to the beam, but they did not use any example abstraction, and only stored full programs as a way to stabilize learning.

5 Experimental Evaluation

Experimental details Our encoder is a Bi-LSTM where the hidden state dimension for each LSTM is 30. The decoder is a feed-forward network with one hidden layer of dimension 50, that obtains the last 4 decoded tokens as input (in addition to encoder information). Utterance tokens (words) embeddings are of dimension 12, beam size $B = 40$ and we retrieve $D = 10$ programs from the cache. We initialize word embeddings by running the CBOW algorithm (Mikolov et al., 2013) on the training data and then optimize them end-to-end. In the weakly-supervised parser we encourage exploration at training time by using meritocratic gradient updates with $\beta = 0.5$ (Guu et al., 2017), where the weight of a program with correct denotation is an interpolation of the model probability distribution and a uniform distribution. Using standard policy gradient ($\beta = 1$) resulted in slightly lower performance. Models are implemented in TensorFlow. In the supervised parser, the parameters other than word embeddings are randomly initialized (Glorot and Bengio, 2010), while in the weakly-supervised parser we initialize parameters using the learned parameters of the supervised model to warm-start the model. We use the Adam optimizer (Kingma and Ba, 2014), a learning rate of 0.001, a mini-batch size of 8 examples, and trained for 7 epochs for the supervised model and 15 epochs for the weakly-supervised model.

Pre-processing Because the number of utterances is relatively small for training a neural model, we take the following steps to reduce sparsity. We lowercase all utterance tokens, and also use their lemmatized form. We also use spelling correction to replace words that contain typos. Last, we replace 8 domain-specific words with their more frequent synonym according to the training set (e.g., *"object"* is replaced with *"item"*, and *"wall"* is replaced with *"edge"*). After pre-processing we replace every word that occurs less than 5 times with an `UNK` symbol.

Evaluation We evaluate on the public development set and test set of CNLVR as well as on the hidden test set by submitting our code to the authors of CNLVR. The standard evaluation metric is accuracy, that is, how many examples are correctly classified. In addition, we report a *consistency* measure, which is the proportion of utterances for which the resulting program has the correct denotation for all 4 images/KBs. We believe this is an important measure as it captures better whether a model can consistently produce a correct answer for an utterance.

Baselines We compare our models to the MAJORITY baseline that always picks the majority class (TRUE in our case). We also compare to the best results reported by Suhr et al. (2017) when taking the KB as input, which is a maximum entropy classifier (MAXENT).

We compare these previous results to the following models:

- RULE: The rule-based parser described in Section 4. We back-off to the MAJORITY baseline when the parser cannot parse the utterance.

Model	Dev		Test-P		Test-H	
	Acc.	Con.	Acc.	Con.	Acc.	Con.
MAJORITY	55.3	-	56.2	-	55.4	-
MAXENT	68.0	-	67.7	-	67.8	-
RULE	66.0	29.2	66.3	32.7	-	-
SUP.	72.8	43.1	69.5	40.2	-	-
SUP.+RERANK	76.5	53.9	74.4	50.4	-	-
WEAKSUP.	79.5	57.3	77.8	52.6	-	-
W.+RERANK	83.5	63.7	80.4	59.0	**83.5**	**64.7**

Table 3: Results of baselines and models on the development set, public test set (Test-P), and hidden test set (Test-H). For each of our models we report both accuracy and consistency.

- SUP.: A supervised semantic parser trained on 3,721 examples automatically generated from abstract examples (and validated on the remaining 1,241 examples).
- WEAKSUP.: A weakly-supervised semantic parser described in Section 4.
- +RERANK: For both the supervised parser and the weakly-supervised parser, we add the heuristic global re-ranking describe in Section 3.

Main results Table 5 describes our main results. Our weakly-supervised semantic parser combined with global re-ranking of the beam (W.+RERANK) obtains 83.5 accuracy and 64.7 consistency on the test set, improving accuracy by 15.7 points compared to state-of-the-art. This is the only model we evaluated on the hidden test set according to the rules of CNLVR evaluation.[2]

The rule-based parser, RULE, is able to handle a non-negligible part of the data and its accuracy is less than two points lower than the MAXENT baseline on then development set and public test set. Training a supervised parser improves accuracy to 69.5 on the public test set, showing that generalizing from generated examples is better than memorizing the manually-defined patterns. Our weakly-supervised parser significantly improves over SUP., reaching an accuracy of 77.8 on the test before re-ranking. For both SUP. and WEAKSUP. re-ranking substantially improves both accuracy and consistency.

To conclude, we develop a semantic parsing approach for the CNLVR dataset. We show that a supervised parser trained by generating examples from abstract examples already improves over approaches that do not use a programming language, and that we are able to overcome the challenges of search and spuriousness, and develop a weakly supervised semantic parser that substantially improves state-of-the-art.

[2] https://github.com/clic-lab/nlvr#running-on-the-held-out-test-set

Model	Dev.	
	Acc.	Con.
-ABSTRACTION	56.7	9.0
-BEAMCACHE	72.8	46.4
-EVERYSTEPBEAMCACHE	81.1	61.4

Table 4: Results of ablations of our main models on the development set. Explanation for the nature of the models is in the body of the paper.

5.1 Analysis

We analyze our results by running several ablations on our best model W.+RERANK on the development set.

To examine the importance of abstract examples, we run our weakly-supervised parser from scratch without pre-training a supervised parser on abstract examples and without using a cache for abstract examples (-ABSTRACTION). We find this parser is unable to overcome the challenges of training from weak supervision. Training and development accuracies remain very low throughout training, and thus we stopped training after 9 epochs, seeing that performance is close to the MAJORITY baseline.

To further examine the importance of the cache itself, we train the weakly-supervised parser by starting from the initialized supervised parser, but without the cache of abstract examples (-BEAMCACHE). We find that this weakly-supervised parser performs worse than the supervised parser it was initialized from by a few points. that is, we are still unable to improve learning by training from denotations. Last, we use a beam cache with full program retrieval only (Section 4), that is, programs are added to the beam only after decoding terminates, rather than at every decoding step (-EVERYSTEPBEAMCACHE). We see that this already results in good performance that is substantially higher than SUP., but is still 2.4 points worse than our best performing model on the development set.

6 Related Work

Training semantic parsers from denotations has been one of the most popular training schemes for

scaling semantic parsers since the beginning of the decade. Early work focused on traditional log-linear models (Clarke et al., 2010; Liang et al., 2011; Kwiatkowski et al., 2013; Berant et al., 2013), but recently denotations have been used to train neural semantic parsers as well (Liang et al., 2017; Krishnamurthy et al., 2017; Rabinovich et al., 2017; Cheng et al., 2017).

Visual reasoning has attracted considerable attention, with the release of datasets such as VQA (Antol et al., 2015) and CLEVR (Johnson et al., 2017a). The advantage of CNLVR is that the language utterances are both natural and compositional. Treating the problem of visual reasoning as an end-to-end semantic parsing problem has been previously done on CLEVR (Hu et al., 2017; Johnson et al., 2017b).

Our method for generating training examples resembles ideas for data re-combination suggested recently (Jia and Liang, 2016), where examples are generated automatically by replacing entities with their categories.

While spuriousness is central to semantic parsing when denotations are not very informative, there has been relatively little work on explicitly tackling it. Pasupat and Liang (2015) used manual rules to prune unlikely programs on the WIKITABLEQUESTIONS dataset, and then later utilized crowdsourcing (Pasupat and Liang, 2016) to eliminate spurious programs. Guu et al. (2017) proposed RANDOMER, a method for increasing exploration and handling spuriousness by adding randomness to beam search and a proposing a "meritocratic" weighting scheme for gradients. In our work we found that random exploration during beam search did not improve results while meritocratic updates slightly improved performance.

7 Conclusion

In this work we presented the first semantic parser for the CNLVR dataset, working on structured representations as input. Our main insight is that in small, well-typed domains we can generate abstract examples that can help combat the difficulties of training a parser from delayed supervision. First, we use abstract examples to semi-automatically generate utterance-program pairs that help warm-start our parameters, thereby reducing the difficult search challenge of finding correct programs with random parameters. Second, we focus on an abstract representation of examples, which allows us to tackle spuriousness and alleviate search, by sharing information about promising programs between different examples. Our semantic parser substantially improves results compared to previous state-of-the-art on the CNLVR dataset.

In future work, we plan to investigate training a globally-normalized re-ranker over the beam of programs generated by the our locally-normalized sequence-to-sequence model, and also train a parser over a simpler programming language that is closer to the representation in the target KB.

References

S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. 2015. Vqa: Visual question answering. In *International Conference on Computer Vision (ICCV)*. pages 2425–2433.

Y. Artzi and L. Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics (TACL)* 1:49–62.

D. Bahdanau, K. Cho, and Y. Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*.

J. Berant, A. Chou, R. Frostig, and P. Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Q. Cai and A. Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *Association for Computational Linguistics (ACL)*.

J. Cheng, S. Reddy, V. Saraswat, and M. Lapata. 2017. Learning structured natural language representations for semantic parsing. In *Association for Computational Linguistics (ACL)*.

D. Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Association for Computational Linguistics (ACL)*. pages 263–270.

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Computational Natural Language Learning (CoNLL)*. pages 18–27.

X. Glorot and Y. Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*.

K. Guu, P. Pasupat, E. Z. Liu, and P. Liang. 2017. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Association for Computational Linguistics (ACL)*.

S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko. 2017. Learning to reason: End-to-end module networks for visual question answering. In *International Conference on Computer Vision (ICCV)*.

R. Jia and P. Liang. 2016. Data recombination for neural semantic parsing. In *Association for Computational Linguistics (ACL)*.

J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick. 2017a. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR)*.

J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, L. Fei-Fei, C. L. Zitnick, and R. Girshick. 2017b. Inferring and executing programs for visual reasoning. In *International Conference on Computer Vision (ICCV)*.

R. J. Kate, Y. W. Wong, and R. J. Mooney. 2005. Learning to transform natural to formal languages. In *Association for the Advancement of Artificial Intelligence (AAAI)*. pages 1062–1068.

D. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* .

J. Krishnamurthy, P. Dasigi, and M. Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*.

J. Krishnamurthy and T. Mitchell. 2012. Weakly supervised training of semantic parsers. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*. pages 754–765.

T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Empirical Methods in Natural Language Processing (EMNLP)*.

C. Liang, J. Berant, Q. Le, and K. D. F. N. Lao. 2017. Neural symbolic machines: Learning semantic parsers on Freebase with weak supervision. In *Association for Computational Linguistics (ACL)*.

P. Liang, M. I. Jordan, and D. Klein. 2011. Learning dependency-based compositional semantics. In *Association for Computational Linguistics (ACL)*. pages 590–599.

T. Mikolov, K. Chen, G. Corrado, and Jeffrey. 2013. Efficient estimation of word representations in vector space. *arXiv* .

P. Pasupat and P. Liang. 2015. Compositional semantic parsing on semi-structured tables. In *Association for Computational Linguistics (ACL)*.

P. Pasupat and P. Liang. 2016. Inferring logical forms from denotations. In *Association for Computational Linguistics (ACL)*.

M. Rabinovich, M. Stern, and D. Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Association for Computational Linguistics (ACL)*.

A. Suhr, M. Lewis, J. Yeh, and Y. Artzi. 2017. A corpus of natural language for visual reasoning. In *Association for Computational Linguistics (ACL)*.

I. Sutskever, O. Vinyals, and Q. V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. pages 3104–3112.

M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Association for the Advancement of Artificial Intelligence (AAAI)*. pages 1050–1055.

L. S. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*. pages 658–666.

L. S. Zettlemoyer and M. Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP/CoNLL)*. pages 678–687.

Appendix A: Programming Language

Logical Function	Return Type	Argument(s) Type(s)
Exist	Bool	Set(?)
Filter	Set(?)	Set(?),BoolFunc(?)
Count	Int	Set(?)
And, Or	Bool	Bool,Bool
Not	Bool	Bool
GreaterThan, LessThan, GreaterEqual, LessEqual	Bool	Int,Int
Equal	Bool	(?,?)
QueryColor	Set(Color)	Set(Item)
QueryShape	Set(Shape)	Set(Item)
QuerySize	Set(Size)	Set(Item)
GetAbove, GetBelow, GetTouching	Set(Item)	Set(Item)
IsYellow, IsBlack, IsBlue, IsCircle, IsTriangle, IsSquare, IsBig, IsMedium, IsSmall, IsTop, IsBottom	Bool	Item
IsTouchingWall	Bool	Item,Side
All, Any	Bool	Set(?),BoolFunc(?)
Union	Set(?)	Set(Set(?))
Select	Set(?),Int	Set(Set(?)
AllSame	Bool	Set(?)

Constants	Type
ALL_ITEMS	Set(Item)
ALL_BOXES	Set(Set(Item))
Color.Blue, Color.Yellow, Color.Black	Color
Shape.Circle, Shape.Triangle, Shape.Square	Shape
Side.Top, Side.Bottom, Side.Left, Side.Right, Side.Any	Side