Tackling the Cornell NLVR task using seq2seq model with weak supervision

Udi Naveh **Veronica Latcinnik** **Omer Goldman**

Abstract

Natural language understanding is an ongoing challenge in the machine learning community. We experiment with the newly published CNLVR dataset of True/False statements regarding an image, in the framework of semantic parsing to logical forms using weak supervision that was proposed by Guu et al. 2016. The small size of the dataset along with complex linguistic structures it contains provide a reasonable challenge that required us to add several modifications to the model. We improve the achieved accuracy on the dev and test sets to 85% and 84%, as compared to 68% accuracy baseline presented by the dataset authors.

1 Introduction

The project presented in this paper was aimed at an attempt to provide a better solution to the problem recently presented by Suhr et al., the Cornell Natural Language Visual Reasoning (NLVR)[1]. The paper presents a new dataset consists of pairs of natural language statements and synthetic images, each annotated with a binary label indicating whether the statement is true or false about the image. The synthetic images are relatively simple and contain only a few two-dimensional geometric shapes (see example in figure 1), but the sentences associated with them are natural and unstructured. This makes the dataset unique among previous related datasets, like CLEVR[4] and SHAPES[5] that contain only synthetic language. While the naturally produced sentences present a variety of linguistic phenomena, from soft and hard cardinality to co-reference, the lexicon quite poor, only 220 words are in use including typos, and very few syntactic structures are highly abundant.

The simplicity of the images allows usage of their structured representation, isolating the linguistic problem from the vision-related part. The dataset contains 3,962 sentences, paired with up to 4 structured representations that were used to create 6 images each by permutation of the sub images. Since we used the structured representations we had a train set of 12,410 image-sentence pairs, dev set of 990 pairs and a test set of 989. The authors presented some initial results of various models to serve as baselines, as shown in table 1. We felt that the baselines did not leverage at all the linguistic properties of the dataset, and that a semantic-driven approach will achieve better results. Thus, we have decided to implement a semantic parser that will convert the input sentences to a logical program that can be executed on the structured representation of an image. The parser will be trained with weak supervision, and rewarded when an outputted program returns a correct label.

2 Related work

Semantic parsing is a vast field of research. In previous years works in the field related heavily on the presence of annotated meaning and logical forms representations (Zelle and Mooney, 1996; Tang and Mooney, 2001; Zettlemoyer and Collins, 2005; Ge and Mooney, 2005; Zettlemoyer and Collins, 2007; Wong and Mooney, 2007). In the last years, many works are trying to expand this paradigm onto training parsers using weak supervision, which avoids the need in expensive annotations. In addition, in the last years semantic parsing was scaled to be used in different domains, with multiple knowledge bases and applications, mainly in the field of question-answering (Clarke et al., 2010; Liang et al., 2011; Krishnamurthy and Mitchell, 2012; Artzi and Zettlemoyer, 2013).

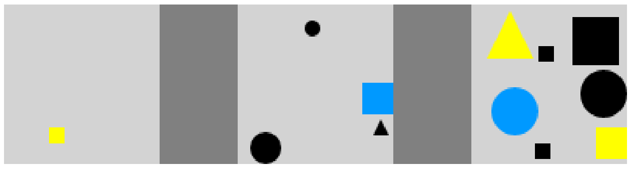
The main challenges of this approach include the combinatorial explosion in the space of logical forms, that makes the search very difficult, and the existence of spurious programs, which may output correct answer while being incorrect, thus harming the learning. Several works propose solutions to these problems, including Pasupat and Liang's WikiTables[7] that suggested improvements to CKY algorithm to enable treatment of non-injective mapping from words to logical elements while adding pruning strategies based on type and denotation constraints. Berant *et al.*[8] went on searching the logical form relevant to a question also using an initial non-injective mapping. They added "bridging operation" to widen the search scope, and eventually combined the logical elements using λ-DCS grammar.

After RNN became frequent and successful model for Neural Machine Translation[11], some attempts have been done to utilize this tool for semantic parsing as well, considering the search for a sentence's logical form as a translation from natural language to a logical one, while still training using weak supervision. Among these works we can mention Liang *et al.*[16] work over WebQuestionsSP dataset and Johnson *et al.*[14] work over the CLEVR dataset, both training an RNN-based encoder and decoder with REINFOCE algorithm[9] to output a logical form that is executed by a "computer", i.e. deterministic executer, over the image. In addition, variations on the training method are also present, as done by Guu *et al.*[2] and detailed below.

Some combinations of classical and neural approaches can be also found, for example Jia and Liang[12], that use a sequence to sequence RNN induced by a more classic CFG.

3 Logical forms

To be able to translate natural language utterances into logical forms that can be executed on the structured representations of images, we first had to create a simple 'programming language' that is suitable for the task, defined by a *syntax* for building valid programs and *semantics* defining their meaning. We used a pre-defined set of *functions* and *values*, that were inspired by the work of Johnson *et al.*[14] on the CLEVR dataset. Each value in our language has a *type* (such as *item, item set, color, integer*) and each function has a fixed arity of 1 or 2, and defined argument types. In this framework, logical programs can be represented as a sequence of tokens in prefix notation, and thus the task of outputting them is viewed as a series of decision - choosing the next token at each step. Using a stack that keeps track of the types used, we were able to limit a decoder to output only syntactically valid programs that have a truth value with regard to the structured representation. Hence the task of learning to translate sentences into such programs focuses on learning semantics. Unlike CLEVR, our dataset requires considerable set-theoretic reasoning, (as each image is divided into different boxes with items). To be able to account for that we also included some elements of lambda-calculus in our syntax in way the preserve the fixed-arity and strong typing constraints. In total, our language included 76 unique tokens, and is expressive enough to represent the meaning of over 95% of the sentences in the training dataset, as estimates from a random sample 100 sentences. [1]



Sentence: there is a small yellow item not touching any wall
Label : True
correct logical form:
exist filter ALL_ITEMS lambda x: AND AND is_yellow x is_small x NOT is_touching_wal x Side.ANY
spurious logical form:
exist filter ALL_ITEMS lambda x: AND is_yellow x is_touching_wall x Side.ANY

Figure 1: an example of an image with correct and spurious logical forms

Pre-processing

The sentences in the training set included 222 unique words in 3163 sentences. Using heuristic spelling correction, lemmatization, and some manually defined replacements of synonyms we cut the size of the vocabulary by half. Replacing rare words (less than five instances in the processed sentences) with <UNK> token, we further diminished it to 86 unique words. This was done in order to reduce the effective dimensionality of the task which was important due to the small size of the data set[2].

Sentence and program patterns

We observed that many of the sentences in the dataset share common patterns, that differ from one another

[1] Assuming that each sentence has a well-defined meaning. This is in fact not the case, as many of the sentences in the dataset are ambiguous.
[2] We note that we have not tested this assumption empirically by running the learning on the unprocessed data

only by a small number of keywords such as numbers, color and shape names, words denoting spatial relation and quantitative comparison. for example, both sentences

'There are exactly 3 yellow squares touching the wall' and *'There are at least 2 blue circles touching the wall'* are instances of the same pattern, and were formalized as.
'There are T_QUANTITY_COMPARE T_INT T_COLOR T_SHAPE touching the wall'.

This revealed that more than half the sentences in the data can be formalized using only about 200 such patterns.
We annotated some of the most common patterns with formalized logical forms using the same replacement rules. So for example, a possible annotation pattern for the examples above would be:

T_QUANTITY_COMPARE T_INT count filter ALL_ITEMS lambda x: AND AND is_T_COLOR x is_T_SHAPE x is_touching_wall x

By keeping a dictionary mapping sentence patterns to logical forms patterns, these patterns can, in turn, be matched back to specific sentences and provide their annotation, that can be run on the structured representation to produce the result. This approach, that involves no learning whatsoever, served as our baseline for all other approaches that take sentence structure into account.

4 Model

As we framed the task as one of learning semantic parsing, a common approach would be to model a distribution over pairs of sentences and logical forms $p(\boldsymbol{z}|\boldsymbol{s})$.
We used a seq2seq encoder-decoder model, in which an input sentence is first encoded using an RNN, and then at each step t, the decoder uses the sentence encoding, denoted by e_m , as well as some representation of the decoding history, $f(z_{1:t-1})$, to calculate the probability distribution over the next tokens $p(z_t|x, z_{1:t-1})$. The final probability of generating a program z given sentence s is given by:
$p(\boldsymbol{z}|\boldsymbol{s}) = \prod_{t=1}^{T} p(z_t|x, z_{1:t-1})$. We employed the same architecture used by Guu et al, that used a bi-directional LSTM as an encoder, and a feedforward network with hidden layers, that computes an attention vector α_i over the outputs of the LSTM and uses it to output a probability vector over all.
The model is fully described by the following equations:[3]

Encoder

1) $h_i^F = LSTM\left(h_{i-1}^F, \phi_u(u_{m,i})\right)$

2) $h_i^B = LSTM\left(h_{i+1}^B, \phi_u(u_{m,i})\right)$

3) $h_i = [h_i^B; h_i^F]$

4) $e_m = [h_{|u_m|}^F; h_1^B]$

Decoder

5) $q_t = ReLU\big(W_q[e_m; f(z_{1:t-1})]\big)$

6) $\alpha_i \propto \exp(q_t^T W_\alpha h_i)$

7) $c_t = \sum_i \alpha_i h_i$

8) $p(z_t|x, z_{1:t-1}) \propto \exp(\Phi_z(z_t)^T W_s[q_t; c_t])$

where $\phi(x)$ is the embeddings vector of the English word x and $\Phi_z(z_t)$ is the embedding of logical token z_t.
For the history embedding $f(z_{1:t-1})$, we used a standard simple approach that concatenates that takes the last k tokens, $z_{t-k:t-1}$ and concatenates their embedding.

4.1 Beam search

Given the model and a sentence s, we would like to use it to output a logical form z with high probability. Finding $argmax_z(p(\boldsymbol{z}|\boldsymbol{s}))$ is not a tractable problem, but several approaches can be taken to get a good estimation. We used beam search for that purpose, and experimented with several modifications to boost its performance (in inference as well as in learning). We discuss some of the approaches here:

[3] We do not elaborate here on the details of this architecture, as it is described in the original paper

We used several kinds of constraints for limiting the exponential search space of the decoding.

Syntactic constraints are enforced by the state of the stack, as described before. This ensures that at time *t*, all partial programs in the beam are prefixes of syntactically valid programs. When the stack of a program is empty, it is means this is a complete valid program. The search stops after a predefined number of steps or when all programs in the beam are complete. Thus, the programs that stay in the beam and are shorter than the maximum allowed length are always valid and will compile, preventing the beam from filling with logically incorrect programs.

Sentence-dependent constraints. An additional restrictions on the space of possible tokens is driven directly by the sentence. It is evident that some tokens can only appear in a correct decoding of a sentence only if their corresponding natural language words appeared in the sentence. For example, the logical token *is_black* is clearly irrelevant unless "*black*" is part of the original sentence.[4]

Other constraints. We used several additional hand-written rules for beam pruning. The reasons for that were avoiding logical forms that are not likely to end up being correct or even to complete after the maximum number of steps. In the context of unsupervised learning (discussed later), this helps avoiding spurious programs, or programs that are not compact (including redundancy or tautologies, such as *lambda_x: AND AND is_yellow x exist All items is_yellow x*) - and thus reward potentially harmful 'behaviors'. We also limited the size of the stack to 3, for similar reasons.

Given the constraints, the effective probability of a program is $p_\theta'(\boldsymbol{z}|\boldsymbol{s}) = \prod_{t=1}^{T} p_\theta'(z_t|x, z_{1:t-1})$.
where $p_\theta'(z_t|x, z_{1:t-1}) = \frac{\psi(z_t, z_{1:t-1}) p_\theta(z_t|x, z_{1:t-1})}{\sum_{\bar{t}} \psi(z_{\bar{t}}, z_{1:t-1}) p_\theta(z_{\bar{t}}|x, z_{1:t-1})}$; $\psi(z_{\bar{t}}, z_{1:t-1}) = 1$ iff $z_{\bar{t}}$ is a valid token given its predecessors, 0 else ; and $p_\theta(z_t|x, z_{1:t-1})$ is the value given by the probability vector outputted by the model.

Auto-complete tokens. In some cases, at a step t in beam search, the set of the valid continuations for a partial program, may contains only one token which is chosen automatically for this step. We experimented with a different behavior in these cases: whenever there is only one possible valid continuation, the model will choose it and "roll forward" to the next token, at the same beam step. In such way, the lengths (number of tokens) of the programs in the beam may vary, but the number of actual choices made in every program is the same across all programs.

Beam re-ranking. We explored several options for choosing the label based on the execution result of the final programs in the beam. The commonly used methods are the result of the execution of the program with the highest log probability, or the majority of the execution results of the programs. We experimented with another ranking method, in the following way: We manually created a lexicon that maps some of the words in the data to logical tokens. For example, the word "*yellow*" is mapped to the sets: {{*'is_yellow'*}, {*Color.YELLOW, equal, query_color*}}. For each word in the sentence that has a lexicon entry – it is said to be represented in the program if at least one of the sets it maps to is present in the sentence (regardless of its order). Given a sentence s and a program z, the relevance score of z is the ratio of the sum of represented words in s and the number of words in s that have lexicon entries. The programs in the beam are sorted by their relevance score, and then by their model log probability. This method favors correct programs over spurious ones.

4.2 Supervised learning

Using the previously discussed set of patterns of 'formalized' sentences and logical forms, we generated a set of 3000 training examples (s,z). We trained the network using cross-entropy loss on each token, until conversion on a held out validation set (generated in the same way). This served two purposes: first, to see whether this approach can generalize from the annotated patterns to other types of sentences, and thus improve the with comparison to the no-learning-baseline. Second, the learned weights from the supervised model were used to initialize the parameters of the unsupervised model.

[4] Note that this works only in one direction, as the word "*black*" in the sentence does not dictate the use of the token *is_black* in the decoding.

4.3 Learning from weak supervision

The main motivation of this project was to approach the task as one of learning from denotations.
Having formulated the parsing problem as sequence prediction, we had to choose a learning algorithm and define a reward function. Given a sentence *s*, and a logical form *z*, a structured representation *r*, and a binary label *y*, indicating whether *s* is true about *r*, we denote by $[\![z]\!]$ the result of running z on r.
A natural reward function would be R(z) = 1 if $[\![z]\!]$ =y else 0. However, with spuriousness being a known problem in such settings, we tried to avoid it by training on the unique sentences rather than on the separate samples. Every program that was outputted for a sentence was first run on all of its' four related samples, and received a reward only if it was correct for the four of them. Though this mitigates the spuriousness problem – it does not eliminate it: given the vast number of possible logical forms, a random program that is consistent with four samples is still much more likely to be spurious than to accurately represent the true meaning of a sentence. To further address this issue, we chose the RANDOMER learning algorithm proposed by Guu et al. [2]. RANDOMER combines the sampling exploration strategy of REINFORCE algorithm [9] and the beam search used in maximum marginal likelihood (MML)[10]. It tackles the spuriousness problem it two ways: to sample programs it conducts an "$\varepsilon -$*greedy randomized beam search*", that enables wider exploration during beam search by including randomly chosen programs along with those with the highest model probability. Secondly, it uses *beta-meritocratic* weighting of the gradients to minimize the upweighting of spurious programs. The combination of these two approaches is aimed at preventing spurious programs from gaining more and more model probability, keeping other potentially correct programs with low model probability and undiscovered.

4.4 Additional modifications

We experimented with additional variants in order to try boost the performance of the model. These methods, presented below, were first tested separately to assess their own impact, and then combined in the final model.
Other variants. Using the restriction on the possible continuation during beam search described in the previous section, sampling using $\varepsilon -$*greedy* approach is naturally biased towards program prefixes with many valid continuations. To prevent this bias, we implemented the epsilon-sampling by sampling first from the unified distribution of all existing prefixes, and then uniformly on its continuations.

Caching programs in beam. We extract the pattern of each sentence, and for every outputted program that got reward, we cache the patterns of the sentence and the program. For each program in the beam, we keep track of its accuracy rates during the training, and only program with high accuracy rates stay in the cache. Given a sentence, it is suggested up to 10 best cached programs (the cached program patterns are sorted by the binomial probability of their accuracy so far). This process over time naturally favors correct programs over spurious programs. We experimented with two variants of caching: in the simple variant, after the beam search is done, each suggested program that is not in the final beam is added to it. In the more complex variant, at each step t of the beam search, the prefix corresponding to t steps of the suggested program is injected to the beam if it is not there. Thus, this process might lead to new programs that were not used before.

Curriculum learning. Following the work of Bengio et al.[3], we explored the possibility of training the model using a curriculum. The difficulty measure we used was the length of the input sentence. We sorted the samples to four groups by difficulty and trained by increasing order for several epochs each time.

5 Main results

5.1 Hyper-parameters:

We trained the models using 30-size LSTMs, feed-forward decoder with hidden layer size 50. Word embeddings embeddings of size 12 were acquired by running CBOW on the preprocessed training sentences. Logical tokens embeddings were also of size 12 and were trained as part of the model. Beam of size 40. These parameters were chosen after trying several variations, but a full grid search was not performed due to shortage of time. Grid search for epsilon and beta values was conducted and the chosen values were 0 and 0.5, respectively. Regarding the history embedding, we experimented with larger size and with a bag of-tokens approach: binary representation of which tokens appeared so far in the decoding - as a compensation of the bounded memory caused by using only last tokens embeddings rather than LSTM decoder. Both approaches lead to slightly better learning of the supervised set but showed no advantage when the learned weights from the supervised session were used on the real dataset, so we chose to use a 4-tokens long history embedding.
All models are implemented in TensorFlow (Abadi et al., 2015). Model parameters are randomly initialized

(Glorot and Bengio, 2010), with no pre-training. We use the Adam optimizer (Kingma and Ba, 2014), a learning rate of 0.001, a minibatch size of 8 examples (sentences), and trained for 13 epochs for the supervised model and 15 epochs for the weakly supervised model.

5.2 Evaluation

The models were evaluated on the provided dev and test sets. In addition to the prediction accuracy one samples, which is the evaluation method used in the NLVR task, we used a measure of consistency, which is the percent of programs predicting the correct label for all four related images ("consistency %"). This measure represents the learning process more truthfully, and is less prone to random calibrations.

5.3 Comparison with previous work

The only previous work we are aware of in regard of the CNLVR task are the baseline models presented by the task creators. The best model used was maximum entropy model that achieved a 68% accuracy, and consistency of 25%. Our models performed significantly better, as can be seen in Table 2.

We started from implementing the baseline model of Guu et al. [2]. However, the model did not succeed in outputting any reward-getting programs at all. In order to start a learning process we had to give the model an initial boost, so we started examining different modification upon it.

5.4 Effect of supervised pre-training on patterns

The use of the logically-valid constraint, that limits possible continuations to only logically valid tokens so that the programs in the beam would compile, while leading to more reasonable program prefixes, still did not help much: the beam either stayed empty (because the programs were not finished when reaching maximum allowed length) or produced incorrect programs that did not get a reward. In these cases and without any learning process, we always output "True" during inference - bringing us to 56% accuracy. The introduction of supervised pre-training had a major effect on the performance during inference, reaching 79% accuracy and 57% consistency on the dev set. In order to verify that the improvement was indeed due to learning and not simply because the patterns "solved" the dataset, we ran the following test: each input sentence whose pattern was trained upon in the supervised learning setup was correctly parsed to its' matching program, executed and the result was returned. If a sentence was not in the patterns list, it returned always "True". This experiment achieved 69.87% accuracy and 35.2% consistency on the dev set, confirming that the supervised training led to an actual learning.

However, when the supervised-learned parameters were used to initialize the unsupervised training, the performance dropped significantly to 57% accuracy. It seemed that the model "forgot" what it has learned, probably due to spuriousness and the noisy signal it received in the unsupervised training.

5.5 Effect of beam constraints

Introducing the additional constraints during beam search seemed to help the model overcome the confusion due to noisy signal and spuriousness. The models' performance when using the supervised pre-trained parameters to initialize the unsupervised training with beam constraints reached 80% accuracy and 62% consistency on the dev set. Therefore, all of the proceeding experiments we ran used this initial setting.

5.6 Effect of auto-complete tokens

The automatic completion of tokens when only a single possibility is available, helped to enhance the baseline models' performance up to 80.8% accuracy and 64.8% consistency on the dev set. We assume that this is due to the methods' reduction of wasted model runs and creation of larger variability of sentences lengths in the beam, which helps to choose the best program.

5.7 Effect of curriculum learning

Training the model on examples by increasing difficulty, which in our case was measured by the length of the sentence, actually seemed to damage the model's performance. This might be due to a wrong difficulty measure – the length of the sentence does not necessarily represents its' linguistic complexity. A better measure would perhaps be the probability of the sentence by its' n-grams, or training loss. We prepared the basis for these experiments, but did not execute them due to time restriction.

5.8 Effect of caching in beam

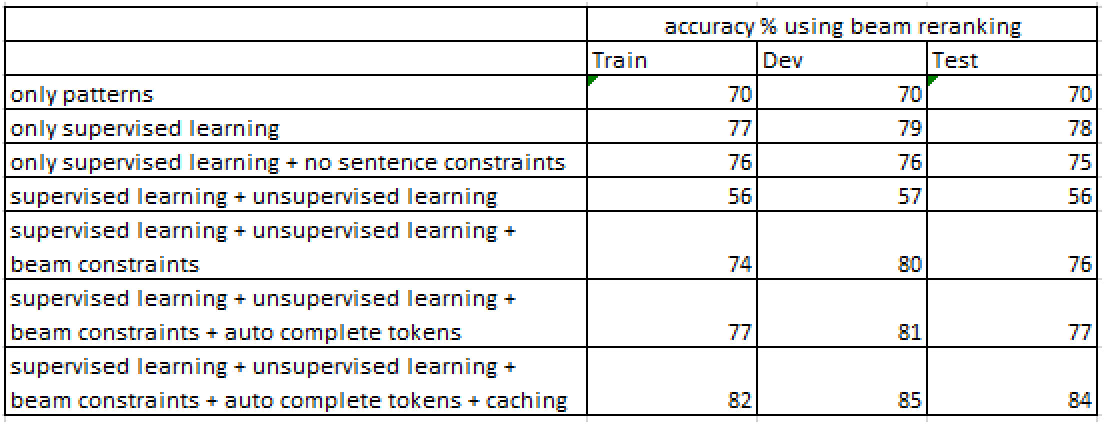
Caching correct programs by sentence pattern during training increased the performance of the model by about 4%. Interestingly, the effect was found only in variant involving 'injecting' programs to the bean during the search, which suggests that No significant effect was found for the simpler depending on the caching variant.

5.9 Effect of beam re-ranking

Re-ranking of beam results was not used during training, but it improved the accuracy scores by 1-4% and the consistency scores by 1%-8%. It is interesting to note that it had larger effect during early phases of training, where the difference in accuracy reached 6-7%.

After combining all of the performance-increasing methods together, we achieved 85% accuracy and 70%

consistency on the dev set and a 84% accuracy and 67% consistency on the test set. The summary of our results is presented in table 2.



	accuracy % using beam reranking		
	Train	Dev	Test
only patterns	70	70	70
only supervised learning	77	79	78
only supervised learning + no sentence constraints	76	76	75
supervised learning + unsupervised learning	56	57	56
supervised learning + unsupervised learning + beam constraints	74	80	76
supervised learning + unsupervised learning + beam constraints + auto complete tokens	77	81	77
supervised learning + unsupervised learning + beam constraints + auto complete tokens + caching	82	85	84

Table 2. Summary of the results of our models evaluation, using beam re-ranking

6 Discussion

The results our model achieved show that the CNLVR task is indeed a challenging task. While beating the baselines presented by the authors, our best approach achieved only 84% accuracy, which seems low for a dataset with such a limited vocabulary. Being a task involving a small and domain-specific data set, it is reasonable to assume that approaches that that are more sensitive to grammar and word-order (for example CCG parsing) could achieve higher scores here. However, our goal, by adopting the model of Guu et al.[2], was to see if we can tackle this task with a general approach.

Analysis of consistency rates of different patterns of sentences shows that some syntactic structures were more easily learned than other. For example, the trained model struggles with even simple sentences involving coordination, such as: "there are 3 blue squares and 2 yellow triangles" – but performs flawlessly on sentences like "there is exactly one blue block as the bottom of a tower with at least 3 items". This might be due to the structure of our logical forms, which expect the relation function to come before the objects it describes: “AND equal_int 3 count filter (…) equal_int 2 count filter (…)”, thus supporting the observation that a CCG parsing would work better for this problem.

We have noticed that the beam constrains have a major contribution to the mitigation of spurious programs, especially comparing to the baseline. Even so, some spuriousness still exists. It is possible that some patterns are more prone to spuriousness, and even our use of reward that considers all 4 samples per sentence did not prevent the creation of such programs. It appears that the main challenge in of natural language understanding in this task is relations between objects in the dataset are complex to learn using so little data, especially with the limited memory of outputted tokens that is used by the model. We feel that this approach might not be the best suitable for this task, and a better solution will be domain-specific.

Some additional work that we could have done, have we had more time, would include using an LSTM decoder instead of feed-forward network, to better memorize the outputted tokens.
Another possible direction is using the beam re-ranking during training, reweighting the programs with consideration to their relevance instead of the meritocratic-beta approach.

References

[1] Alane Suhr, Mike Lewis, James Yeh, and Yoav Artzi. 2017. *A Corpus of Natural Language for Visual Reasoning.* http://alanesuhr.com/suhr2017.pdf

[2] Kelvin Guu, Panupong Pasupat, Evan Zheran Liu and Percy Liang. 2017. *From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood*. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (2017).*

[3] Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In ICML.

[4] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. 2016. *CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning*. *CoRR* abs/1612.06890.

[5] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016b. *Neural module networks*. In *Conference on Computer Vision and Pattern Recognition*.

[6] John M. Zelle. 1995. *Using inductive logic programming to automate the construction of natural language parsers*. Ph.D. thesis, University of Texas, Austin.

[7] 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)*. https://doi.org/10.3115/v1/P15-1142

[8] Jonathan Berant, Andrew Chou, Roy Frostig and Percy Liang. 2013. *Semantic parsing on Freebase from question-answer pairs*. In *Empirical Methods in Natural Language Processing (EMNLP)*

[9] R. J. Williams. 1992. *Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning 8*.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. 1977. *Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society: Series B 39*.

[11] 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)*.

[12] R. Jia and P. Liang. 2016. *Data Recombination for Neural Semantic Parsing*. https://arxiv.org/pdf/1606.03622.pdf

[13] 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

[14] J. Johnson, B. Hariharan, L. van der Maaten, J. Hoffman, F. Li, C. L. Zitnick, and R. B. Girshick. *Inferring and executing programs for visual reasoning*. CoRR, abs/1705.03633, 2017.

[15] J. M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic proramming. In Proc. of the National Conference on Artificial Intelligence (AAAI).

[16] 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)*.

[17] L. R. Tang and R. J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In Proc. of the European Conference on Machine Learning (ECML).

[18] L. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In Proc. of the Annual Conference in Uncertainty in Artificial Intelligence (UAI).

[19] R. Ge and R. Mooney. 2005. A statistical semantic parser that integrates syntax and semantics. In Proc. of the Annual Conference on Computational Natural Language Learning (CoNLL).

[20] L. Zettlemoyer and M. 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 on Computational Natural Language Learning (EMNLP-CoNLL).

[21] Y.-W. Wong and R. Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In Proc. of the Annual Meeting of the Association for Computational Linguistics (ACL).

[22] 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.

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

[24] 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.