Abductive Visual Question Answering for Label Efficient Learning

Karan Samel
Georgia Tech
ksamel@gatech.edu

Binghong Chen
Georgia Tech
binghong@gatech.edu

Le Song
Georgia Tech
lsong@cc.gatech.edu

Abstract

We propose Abductive Visual Question Answering (AB-VQA), a neuro-symbolic framework that addresses VQA tasks given limited labels by leveraging abduction. These VQA problems require multi-step reasoning based on the vision model's understanding of the visual scenes, which require a large number of labels to train. Our work focuses on the small data regime, where only a few of these annotated scenes are provided. We further leverage QA feedback, whose labels are easier to obtain, to tune the vision models. This is done by abducing all the possible correct vision labels necessary to arrive at the correct answer. In order to make minimal label changes, we formulate the abduction process as an optimization problem with consistency constraints. We show that in this limited label scenario we are more accurate and sample efficient than competitive neuro-symbolic VQA methods.

1 Introduction

Humans are able to handle complex tasks by using both perception, such as vision, and symbolic knowledge, which are known facts and hypothesis. The recent success of deep learning has led to efficient and accurate methods of these preception capabilities [21]. Similarly, symbolic systems have been long used for reasoning tasks [25] and as expert features for machine reasoning models [3].

There is a recent surge of interests in the AI community to band these two forms of reasoning into a single process. On the one hand, continuous neural sub-symbolic representations can be converted into symbolic one via Probabilistic Logic Programming [6] or Statisitical Relational Learning [20]. On the other hand, symbolic programs can be written in a sub-symbolic friendly fashion through Differentiable Programming [1]. There are many works that try to bridge the gap, as each representation has its advantages, [4, 5, 23]. These hybrid neural and symbolic methods have been used in many applications from geographical forecasting [9] to new forms of quantum computing [10].

Many of these methods focus on problems that require simple perception models and limited logical rules. However, we are interested in exploring how neuro-symbolic frameworks scale to more difficult problems, such as VQA. The challenges in neuro-symbolic VQA are:

1. Annotated images with object and relation attributes are expensive to annotate. With few data points current VQA methods perform poorly.
2. Leveraging symbolic logic to train VQA systems end to end require many training iterations and have high variance.

In our AB-VQA setting, we convert sub-symbolic representations to symbolic ones during our reasoning process. This lets us leverage logical abduction [19], which involves finding a grounding for a set of variables that leads to an observed output conditioned on some background rules. This

Preprint. Under review.

parallels our framework where we abduce neural object predictions to arrive at the question's answer given logical reasoning rules.

To address the first challenge of expensive image annotations, AB-VQA initializes vision models by pre-training on a few annotated images. Then it leverages end-to-end question answer feedback, which is easier to annotate, to propose new image annotations for training through abduction.

For the second challenge, we propose a new abduction policy that minimizes the changes from the original predictions to produce more reliable intermediate prediction steps. This is because generic abduction sampling finds a single set of intermediate labels that satisfies the final output, but this can lead to spurious label groundings. We empirically show that our abduction policy provides more accurate supervision data when training and reduces sample variance, which is especially important when data is limited.

We test our AB-VQA framework with limited annotated images and QA pairs. In this setting we outperform state-of-the-art neuro-symbolic VQA models by a large margin. We also compare against traditional abduction search, where we greatly reduce the model variance and training times.

2 Related Work

There are previous works that leverage logical abduction for neuro-symbolic tasks. Additionally in the works of VQA, there are methods ranging from purely deep learning based to more neuro-symbolic.

Abductive learning. Within the works of machine learning and abduction, ABL [4] looks into leveraging abduction to harmonize neural and symbolic models. In their framework, they look towards jointly learning neural modules and symbolic logic for the task of bit addition using MNIST digits. In our setting we fix the symbolic logic but have more complex consistency constraints with respect to the compositional reasoning required for complex VQA scenes. Additionally we handle multiple perception models, akin to larger real-world reasoning framework requirements.

Deep learning. For deep learning VQA methods, LCGN [14] model runs message passing between the detected objects jointly with the input question, predicting the final answer from the last message passing step. The work on MAC [15] proposes a new recurrent architecture that modulates the relevant parts of question and object features needed for final reasoning. Similarly, NSM [17] builds an initial scene graph of objects and relations, then leverages an recurrent network to modulate the attentions states between the objects to determine if it is the answer. Leveraging the successes of BERT [7], LXMERT [27] builds a cross-modal BERT representation to implicitly encode relational attention between objects and text. These joint text-vision representations have been gaining more traction with the availability of more data [22, 26].

Neural modules. These approaches uses a functional program form of the question. By executing the question step by step, it arrives to the answer in the last step. The executions are performed over modular networks that learn the behavior of each function. N2NMN [13] dynamically compose these modules into a question specific network and is executed on the entire image feature. MMN [2] executes the program on a single meta-module network to select different objects, parametrized by the function signature at each step.

Neuro-symbolic methods. Instead of learnable neural modules, the logic for each function can be programmed in a symbolic and deterministic fashion for operations such as `sum` or `max`. These neuro-symbolic methods then just have to train vision models to extract the objects for deterministic reasoning. NS-VQA [30] trains their object extraction and detection in an end-to-end fashion to extract the scene graph and uses discrete module representations during execution. NS-CL [24] uses natural supervision through only the object features, text questions and answers. Their program function representations are continuous, which can be used to train their model end-to-end. Both NS-VQA and NS-CL achieve almost perfect accuracy on CLEVR, [18], a synthetic VQA dataset.

We test our methods on an augmented version of CLEVR. In this setting we show that AB-VQA outperforms the most recent VQA methods given limited training labels. We also provide an analysis of what parts of our framework contribute to this performance.

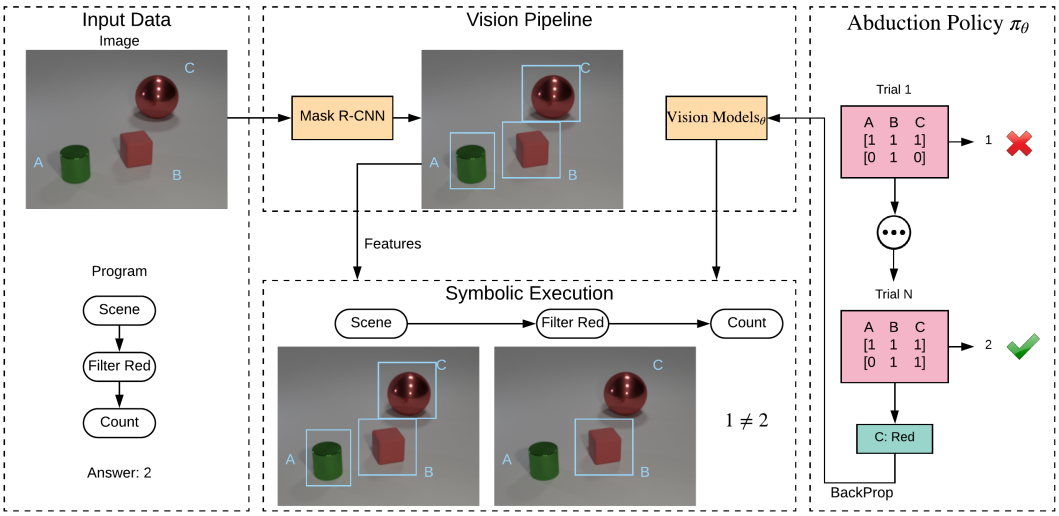

Figure 1: During training AB-VQA takes in an image, question program and answer. Given the image it extracts the detected objects. Then it executes the question program sequentially, using the intermediate object attribute predictions by the vision models. If the answer is incorrect, we abduce possible labels for the detected objects. Here a bitmask is used to select which objects are chosen for a particular operation. Once an abduced label is found, the objects that were labeled incorrectly are corrected and used for retraining.

3 Method

In AB-VQA, the reasoning process is broken down into 3 major steps, highlighted in Figure 1. The first is to extract the objects from the scene and generate features for those objects. In the second, we represent the image as a scene graph to store the predicted object attributes and relationships when executing the symbolic program. If the execution fails to produce a correct answer, the third step is to abduce the correct label and retrain the vision models to reflect these corrections. This last step is core to producing supervised labels in a fast and accurate manner to boost model performance given limited labels.

3.1 Neural Vision Models

Object detection. To detect the objects we trained a Mask R-CNN [11] to extract the corresponding bounding boxes of the objects. We then take detected objects and pass it through a pre-trained ResNet-50 model [12] to extract image features. At this stage we just have detections of some objects and their corresponding feature vector, which are fed to our vision models during training.

Concept prediction vision models. These vision models are two-layer neural networks used to predict object concepts, such as `red` or `cube`, and relations given the raw object features. By decoupling the detection and prediction, we can control the amount of scene graph labels provided to these vision models during experimentation.

The entire vision pipeline consists of the object detection, which is assumed to be accurate, and our vision models, which we tune during training through abduced labels. The implementation details can be found in appendix A.

3.2 Symbolic Program Execution

Given a natural language question, we use the ground truth functional program representation in all our tests, provided in CLEVR. We note that question to program translation models [24] have already been well optimized for CLEVR, thus is not the main focus for our work.

Each program is composed of functions with symbolic parameters. For example, in `filter_red(scene)` the function `filter` with the parameter `red` selects the objects in our image

whose color attribute is the concept `red`. Due to this, all functions are implemented in a symbolic manner following NS-VQA.

Each function within the program is executed sequentially and calls a specific vision model needed to execute that function. In `filter_red(scene)`, we only need to call the color model as other attributes are irrelevant. As the program gets executed, we build the minimal scene graph needed to answer the question, while most methods pre-compute the scene graph.

During program execution, the selection of objects are carried out through sampling the trained vision models. During training if there are ground truth scene graphs, then we can obtain direct supervision to these models. However, such ground truth labeling of attributes and relational information is quite expensive to collect, requiring human annotator to spend significant amount of effort to generate.

If there are no ground truth scene graph, we can leverage other weak signals such as end-to-end QA accuracy. In this scenario with discrete functions, training needs to be carried out in a REINFORCE manner [28], or abducted label method described next.

3.3 Abduction Policy

When evaluating a question for an image, the label information only involves the answer to the question. This is relatively cheap type of feedback from an annotator, without the need for careful and expensive labeling of the scene graph or reasoning path. However, this type of feedback also makes learning harder, so we aim to design algorithms to learn from such low cost labels or weak signals.

The function program $Q = (o_1, \ldots, o_T)$ tells us to execute a sequence of operations or functions o_i. If there are multiple reasoning branches that are aggregated, the execution trace follows a DAG dependency structure. We use a Markov decision process (MDP) formulation for the execution process:

- A policy $a_t \sim \pi_\theta(a_t|S_t, o_t, I)$ with vision models parameters θ will take the operation execution from the previous stage S_t, the operator indicator o_t and image I, and output an action a_t. This action a_t corresponds to the object selections made for the current operation from the input image.
- The action will cause the state S_t to be updated to S_{t+1}, and the transition is described by $P(S_{t+1}|S_t, a_t, o_t)$. This is the operation execution for the current stage. Depending on the operator o_t, the transition may be deterministic or stochastic. For instance, if the module corresponds to a `count(objects)` operation, the result is deterministic. If the module corresponds to a probabilistic logic rule, the result is stochastic.

After this sequence of operations, we obtain the a reward $R(A, S_{t+1})$ by comparing the last state with the answer, and tell us whether the answer is correct or not. If the answer is correct $R = 1$, and otheriwse 0. Then the expected correctness of answer given the uncertainty in π_θ and P is

$$\mathbb{E}[R(A, S_{t+1})], \text{ where} \tag{1}$$

$$S_{t+1} \sim P(S_{t+1}|S_t, a_t, o_t),\ a_t \sim \pi_\theta(a_t|S_t, o_t, I),\ S_0 = \emptyset,\ \forall t = 1, \ldots, T \tag{2}$$

Given m data points expressed as image I, questions Q and answer A triplets: $(I_1, Q_1, A_1), \ldots, (I_m, Q_m, A_m)$, the learning problem can be expressed as:

$$\pi_\theta^* = \arg\max_{\pi_\theta} \sum_{i=1}^{m} \mathbb{E}[R(A_i, S_{i,t+1})] \tag{3}$$

Label abduction. If for a particular data point (I, Q, A) the answer according to the model is incorrect, one can try to find corrections by making minimal changes to the model prediction as follows:

$$c_1^*, \ldots, c_T^* = \text{argmin}_{c_1,\ldots,c_T}\ D(a_1, \ldots, a_T \| c_1, \ldots, c_T) \tag{4}$$

$$\text{s.t. } R(A, S_{t+1}) = 1 \text{ makes the answer right.} \tag{5}$$

$$S_{t+1} \sim P(S_{t+1}|S_t, c_t, o_t), \forall t = 1, \ldots, T \tag{6}$$

where $D(\|)$ is some distance measure between the corrections and the model predictions. For instance, $D(\|)$ can be the negative likelihood of the correction under our model π_θ, or the edit distance.

Intuitively, we want to attempt the most likely corrections iteratively to find a solution by minimizing $D(\|)$. Sampling all possible corrections at once, can cause right answers through spurious label changes, which we want to mitigate.

Sampling methods. Due to the compositional nature of the reasoning tasks, we attempt to optimize c_i^* in a greedy fashion at each step of the program from $i = 1$ to $i = T$ instead of jointly from $i \subseteq \{1, \ldots, T\}$.

This better enforces the consistency constraint when a single c_i^* update leads to valid program executions. Valid executions mean that the predicted or abduced labels c_i^* lead to a final answer, whether right or wrong. This is opposed to making conflicting changes in all $c_i^*,\ c_j^*$ where $i \neq j$, leading to a failed program execution due to the manual abduced changes. If this greedy approach fails to find an answer, we fall back on exhaustively sampling $a_t \sim \pi_\theta$ at all program levels for a fixed number of iterations. Additional details are laid out in Appendix B.

Supervised training. Suppose the abductive label has certain position which is different from the argmax action of current the policy. Say in the i-th position,

$$c_i^* \neq a_i \tag{7}$$

We will then create two labeled data point $(c_i, +)$ and $(a_i, -)$ to update the parameter in the policy $\pi_\theta(a_t|S_t, o_t, I)$.

Curriculum learning. The optimization may be difficult especially for long sequence of complex actions. Thus we will use curriculum learning to address this problem. Essentially, we will start from simple questions which involve very few actions, and then gradually move to those questions involving two or more selection actions. This way the optimization to find the abductive label will have good initial values, will require fewer changes to get the right answers.

4 Experiments

We are interested in seeing how well we handle the more complex, compositional questions if we train on only simple questions. For this we build on top of CLEVR to create CLEVR Decomposed (CLEVR-DC), an augmented dataset. In this dataset we take the questions that perform reasoning multiple reasoning branches, such as `union` or `greater_than`, and form new questions containing the individual reasoning programs. Within CLEVR-DC we only train on these short decomposed questions of length 3 to 5, and test on the original validation set. Appendix C lays out the data set generation details.

4.1 Training Setup

To test our methods we restrict number of image scene graphs we used to train our vision models. To better emulate human concept learning, we only use 7 such scene graphs (0.01%) of all scene graphs. Within these scene graphs we only have 15 examples on average for each concept, and we expect the model to get better by implicitly learning better representations as part of more complex reasoning tasks through question answering.

Given these pre-trained models, we provide a fixed set of image, question, answer pairs to learn from. The images during the question answering are not supervised, so AB-VQA extracts the object attributes and relations through the vision pipeline. We only provide feedback whether the question was answered correctly, and abduce new vision labels using an edit distance policy when the answer is wrong. All the implementation and training schedules are available in Appendix D.

4.2 Results

We test AB-VQA against other state-of-the-art neuro-symbolic baselines on CLEVR-DC in Figure 2 and run 5 trials for all experiments. Since we start with pre-trained image models, we burn in NS-CL with many sampled questions generated from the 7 scene graphs, for a more direct comparison.

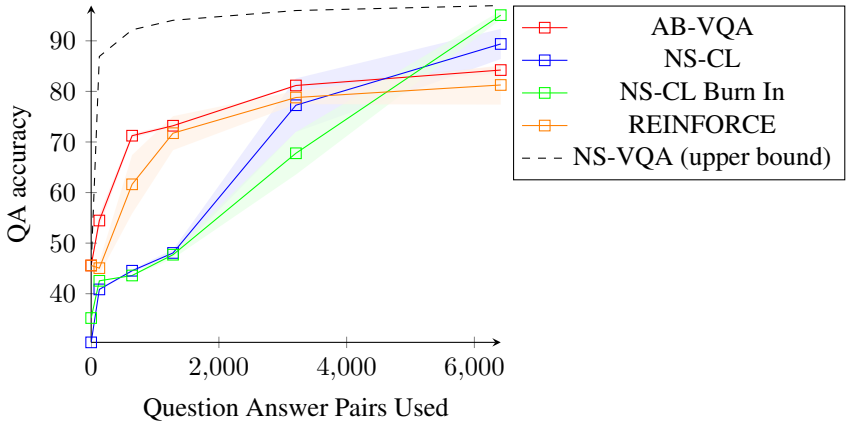

Figure 2: Method Comparison on 7 supervised images.

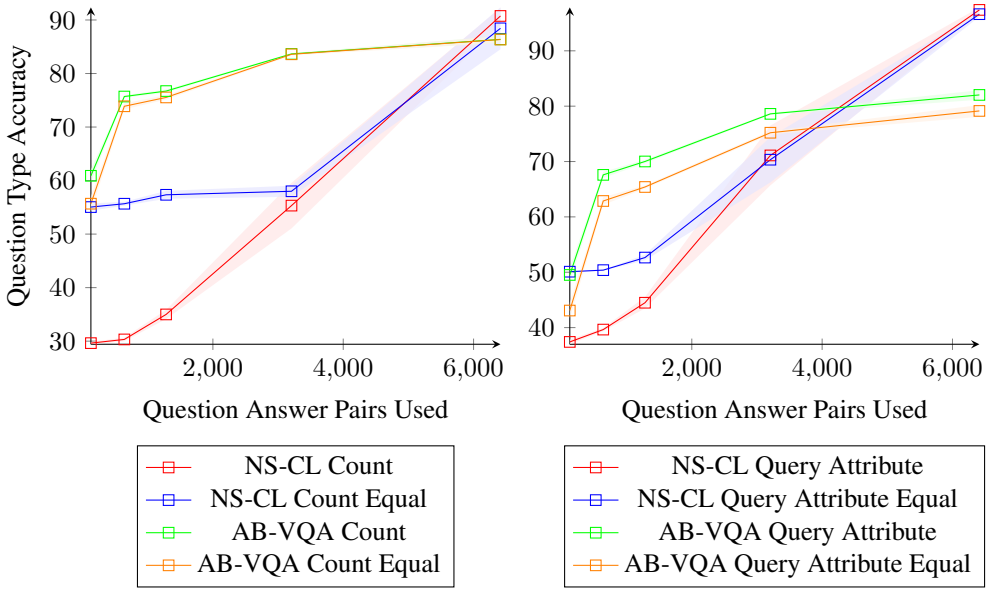

Figure 3: Validation QA accuracy by question type. We are comparing the performance discrepancies between AB-VQA and NS-CL burn in. Count questions require get multiple object detections correct and query requires retrieving an attribute relating to a single object. The equality questions compare if two program executions, either count or query, are equal.

Limited QA samples. We observe that AB-VQA can better refine the vision models with limited QA samples by estimating the error with discrete abduced labels. This indicates that with few incidental QA signals, this abductive sampling is able to estimate the error gradient better than the continuous representation.

The validation questions are further decomposed by count and query types (Figure 3). Answering whether two attribute or quantities are equal require multiple reasoning chains to retrieve those values. Due to this, equal questions should be more difficult, which are shown by the AB-VQA results. However, this is not the case for the NS-CL burn in results, where the equal surpasses the counts in the low QA region. This highlights the necessity to debug these reasoning systems in a symbolic manner, as used in AB-VQA and NS-VQA. Understanding the logic steps in a sequential manner can flag spurious model results.

Many QA samples. With more data, the continuous representation of NS-CL dominates as with a more reliable error gradient. We hypothesize that this is due to NS-CL directly optimizing the QA performance, while we optimize two separate objectives: training our abduction policy vision models

and sampling the abduced label. This joint parameter tuning and abduction sampling means that our method takes would take more samples to reach the same performance.

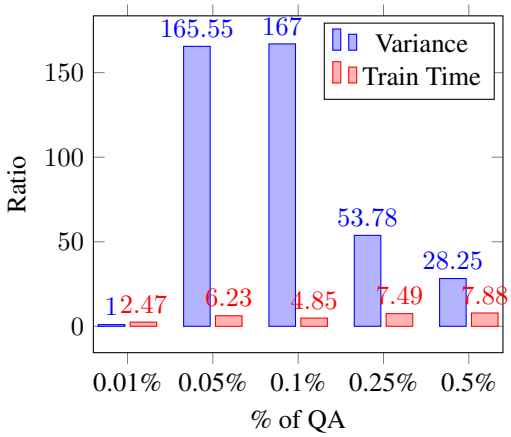

Figure 4: Compare metrics between REINFORCE and AB-VQA. The variance ratio is $\frac{Var(\text{REINFORCE})}{Var(\text{AB-VQA})}$. We compute the average train time for each lesson epoch $\frac{\text{Train Time REINFORCE}}{\text{Train Time AB-VQA}}$. The questions are sampled as percentages of the entire CLEVR-DC dataset.

REINFORCE. We also test a baseline that uses our same setup, but uses REINFORCE instead of our abduction policy. The REINFORCE sampling can be observed as a generic abduction policy that exhaustively samples the model prediction. We surpass REINFORCE as we can more precisely sample labels through our abduction policy. This also greatly reduces our overall variance as seen in Figure 4, giving more confidence during model tuning. AB-VQA also speeds up training, as REINFORCE requires many sampling and retraining steps within each epoch to achieve comparable performance.

NS-VQA. We are also interested in the NS-VQA performance which trains its vision models end-to-end on 4k scene graphs. In comparison, AB-VQA just uses the object detection to just retrieve the object features from and defer the scene graph construction to the vision models, which were trained on 7 scene graphs. We modified NS-VQA to train on the scene graphs for each question set, thus can be seen as an upper bound performance for each set of questions used.

4.3 Ablation Study

	Questions Used				
Component(s)	128 (0.01%)	641 (0.05%)	1.2k (0.1%)	3.2k (0.25%)	6.4k (0.5%)
B	45.58	45.58	45.58	45.58	45.58
B + C	47.34 $\pm$ 0.00	57.48 $\pm$ 0.00	61.21 $\pm$ 0.46	70.27 $\pm$ 0.59	67.16 $\pm$ 0.33
B + C + A	50.09 $\pm$ 0.00	64.77 $\pm$ 0.09	68.59 $\pm$ 0.43	79.75 $\pm$ 0.30	81.04 $\pm$ 0.17
B + C + A + S	53.87 $\pm$ 1.19	69.11 $\pm$ 2.27	74.08 $\pm$ 3.58	80.80 $\pm$ 0.46	83.75 $\pm$ 0.65

Table 1: Ablation performance by cumulative components.

AB-VQA components. We perfomed an ablation study to see how much each component of our AB-VQA contributed to the validation performance in Table 1. The components are described as follows and are utilized in a cumulative fashion in AB-VQA:

- Base (B): The base vision models with no QA signals.
- Correct (C): If we get the answer correct, we add those vision predictions to the data set for retraining. This can be seen as data augmentation with data that aligns with our current prediction model.
- Abduction (A): If we get the answer wrong, we perform abduction and add those image predictions if we can abduce the correct label.

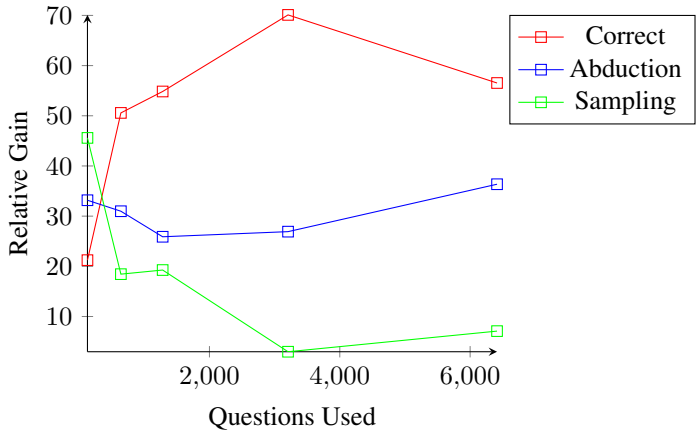

Figure 5: Relative gain for each component, where the total gain (TG) is calculated from all cumulative components used minus the Base ($TG = \text{Score}(B + C + A + S) - \text{Score}(B)$). Then the gain for a particular component, such as Abduction (A) is calculated as $\frac{\text{Score}(B+C+A)-\text{Score}(B+C)}{TG}$ from the cumulative scores in Table 1.

- Sampling (S): This is the sampling step after failing to abduce the label using the greedy method.

In general, a large gain comes from extending the training data set using the image features from the correct predictions. The overall QA accuracy gain from abduction is relatively constant around 30%. With more samples, the vision model accuracy rises. Therefore the correct labels contribute more to the data, and less gains are made from sampling. This roughly follows the trends in Figure 5.

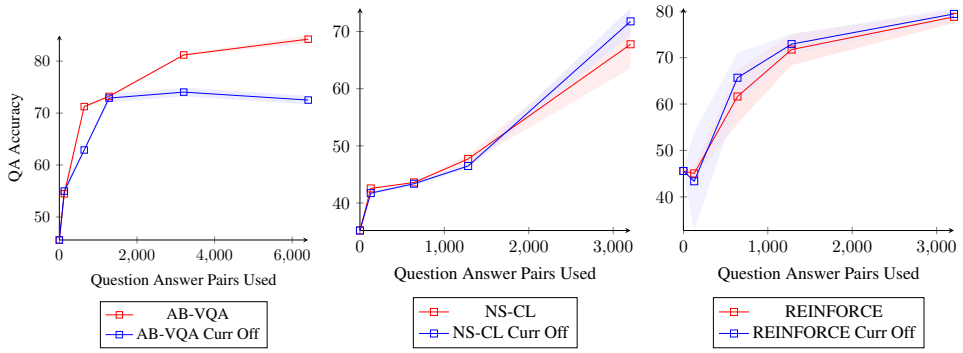

Figure 6: Performance of models with curriculum learning disabled.

Curriculum learning. During the training procedure we also inspected the importance of curriculum learning to our training procedure. Figure 6 lays out the fact that curriculum learning benefits for the abductive label case, but is comparable for NS-CL and REINFORCE in the small sample region.

5 Discussion and Future Work

We present a method that uses abduction to tune neural models for compositional reasoning tasks. The AB-VQA framework aims to bootstrap these neuro-symbolic reasoning tasks when data is limited and expensive to obtain. We observe that minimizing abduced label changes to train vision models outperform end-to-end differentiable models in the low data region. Furthermore we are more accurate and efficient than REINFORCE methods.

It is important to re-emphasize that AB-VQA is not specifically tailored to CLEVR, and can be used for any neuro-symbolic compositional reasoning task. With this in mind, we intend to test how

abduction scales to more complex reasoning tasks in both visual reasoning such as GQA [16] and numerical and text based reasoning with DROP [8].

As more data is added to the system, we will explore how to bridge the gap between these abductive logical and differentiable frameworks for complex reasoning tasks. There are differentiable logic frameworks which integrate neural modules such as DeepProbLog [23] that show good generalization in simple reasoning tasks, but will require more work to represent complex structures such as scene graphs.

Broader Impact

This research direction aims to build reasoning systems with limited labels. The overarching goal is that this requirement can aid domain experts or machine learning layman to leverage a reasoning system with what data they have. Using a symbolic logical framework provides a more grounded way of the expert to convey the knowledge they have to aid the system. This provides a synergy between the knowledge, existing data, and the robustness of neural models to automate and refine routine tasks.

The drawbacks are that the user must understand the system capability given the amount of data provided and the variance in performance. On over-reliance on neural components can skew in the reasoning process if the data is not representative of the original distribution.

References

[1] A Auslender. Differentiable stability in non convex and non differentiable programming. In *Point-to-Set Maps and Mathematical Programming*, pages 29–41. Springer, 1979.

[2] Wenhu Chen, Zhe Gan, Linjie Li, Yu Cheng, William Wang, and Jingjing Liu. Meta module network for compositional visual reasoning. *arXiv preprint arXiv:1910.03230*, 2019.

[3] James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *Proceedings of the fourteenth conference on computational natural language learning*, pages 18–27. Association for Computational Linguistics, 2010.

[4] Wang-Zhou Dai, Qiuling Xu, Yang Yu, and Zhi-Hua Zhou. Bridging machine learning and logical reasoning by abductive learning. In *Advances in Neural Information Processing Systems*, pages 2811–2822, 2019.

[5] Luc De Raedt, Sebastijan Dumančić, Robin Manhaeve, and Giuseppe Marra. From statistical relational to neuro-symbolic artificial intelligence. *arXiv preprint arXiv:2003.08316*, 2020.

[6] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100(1):5–47, 2015.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[8] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *arXiv preprint arXiv:1903.00161*, 2019.

[9] Florentino Fdez-Riverola, Juan M Corchado, and Jesús M Torres. Neuro-symbolic system for forecasting red tides. In *Irish Conference on Artificial Intelligence and Cognitive Science*, pages 45–52. Springer, 2002.

[10] Dario Gil and William MJ Green. 1.4 the future of computing: Bits+ neurons+ qubits. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 30–39. IEEE, 2020.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.

[14] Ronghang Hu, Anna Rohrbach, Trevor Darrell, and Kate Saenko. Language-conditioned graph networks for relational reasoning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 10294–10303, 2019.

[15] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*, 2018.

[16] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. *arXiv preprint arXiv:1902.09506*, 2019.

[17] Drew A. Hudson and Christopher D. Manning. Learning by abstraction: The neural state machine. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5901–5914, 2019.

[18] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2901–2910, 2017.

[19] Antonis C Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *Journal of logic and computation*, 2(6):719–770, 1992.

[20] Daphne Koller, Nir Friedman, Sašo Džeroski, Charles Sutton, Andrew McCallum, Avi Pfeffer, Pieter Abbeel, Ming-Fai Wong, David Heckerman, Chris Meek, et al. *Introduction to statistical relational learning*. MIT press, 2007.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[22] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, pages 13–23, 2019.

[23] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Deepproblog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

[24] Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The Neuro-Symbolic Concept Learner: Interpreting Scenes, Words, and Sentences From Natural Supervision. In *International Conference on Learning Representations*, 2019.

[25] Allen Newell and Herbert Simon. The logic theory machine–a complex information processing system. *IRE Transactions on information theory*, 2(3):61–79, 1956.

[26] Weijie Su, Xizhou Zhu, Yue Cao, Bin Li, Lewei Lu, Furu Wei, and Jifeng Dai. Vl-bert: Pre-training of generic visual-linguistic representations. *arXiv preprint arXiv:1908.08530*, 2019.

[27] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[29] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

[30] Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B Tenenbaum. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems (NIPS)*, 2018.

A Model Details

Object Extraction. We use Detectron2 [29] to train the object extraction models using the stock `mask_rcnn_R_50_FPN_1x` configuration. The image data set we use to train the extraction is one generated from NS-VQA. These image have both bounding boxes and masks, which enables them to be more precisely extracted. Our method and NS-CL just use this for detection, while NS-VQA leverage these models for object attribute prediction.

For the feature extractor we apply the standard ResNet-50 input transforms and extract the features from the final hidden layer. The ResNet-50 used is the standard PyTorch Vision implementation.

Vision Models. There are two different types of vision models required: attribute and relation models. The attribute models are neural networks with 1 hidden layer. Separate models are trained for each attribute in {`color, shape, size, material`}. The architecture is as follows:

$$\begin{aligned} \text{out} &= \text{ReLU}(O_i \cdot W_1 + b_1) \\ \text{logits} &= \text{out} \cdot W_2 + b_2 \end{aligned}$$

Here $O_i \in \mathbb{R}^{2048}$ is the object feature extracted from the object extraction step for object i. We then have $W_1 \in \mathbb{R}^{2048\times1024},\ b_1 \in \mathbb{R}^{1024},\ W_2 \in \mathbb{R}^{1024\times C},\ b_2 \in \mathbb{R}^C$. Where C are the number of output classes for the attribute. For example in CLEVR, `material` is either {`rubber, metal`} so C = 2.

The relation models are also have 1 hidden layer but the output is binary. Given the subject object's bounding box and the candidate object's bounding box, it determines a binary relation. There are two relations that are needed to be predicted: `front` vs `behind` and `left` vs `right`. These relations are not mutually exclusive, so an object can be both `behind` and `left` of another object.

For the model inputs we use the concatenated bounding boxes of subject i and object j of which we want to determine the relationship. We have the location feature $loc = [loc_i \,||\, loc_j]$ where $[*||*]$ indicates concatenation. Similar to above we have $W_1 \in \mathbb{R}^{8\times16},\ b_1 \in \mathbb{R}^8,\ W_2 \in \mathbb{R}^{16\times1},\ b_2 \in \mathbb{R}^1$.

B Abduction Policies

Object Sets. During abduction label changes we are flipping a bitmask or object set c_i^*, corresponding to which set of objects are selected in a step of the program. Only one object set is manipulated at once in our greedy approach, where 1 means that object is active in that step and 0 otherwise.

Flip Strategy. When choosing the labels to flip there are two high level strategies to make based on the incorrect answer. If we know that we know the direction of change, either increase or decrease the number of selected objects, then we only make changes in that direction.

This can be done if we have an `exist` or `count` question, where we can determine the magnitude of our error. Thus we have the following strategy when abducing new labels:

1. If the question is `exist` or `count`, then determine if we need to increase or decrease the number of labels. Then only make those changes instead of at random.
2. If we cannot make a determination of how to change the object sets, then stick to the abduction policy, such as edit.

For the generic abduction step in 2. we enumerate all possible combinations in increasing change size. So, we would first flip every mask, then every two mask and so on. At each flip we are checking if the corresponding changes, if inserted into the program execution, makes the answer correct.

Thresholding. When making the label flips, we also want to avoid making unreasonable changes to the labels. For example, if during a `filter_red(object_A)`, `object_A` is predicted red with probability 0, then changing it to red may lead to a spurious correct answer. Therefore when making flips, we add an additional constraint such that if an object mask is changed from:

- $1 \to 0$ if $P(\texttt{object\_A})) > \epsilon$
- $0 \to 1$ if $P(\texttt{object\_A})) < 1 - \epsilon$

C CLEVR-DC

When splitting the longer questions, the sub-question programs for new questions in the dataset. For each sub-program we run a ground truth CLEVR parser that takes in the corresponding question scene graph and program to produce the correct answer for that new sub-question. This is because CLEVR does not provide the intermediate labels for the programs.

We also want the sub-questions to contain valid answers available in current CLEVR data set, such as numeric, boolean, or concept based. Some sub-questions just end with a bitmask object set, for example after a filter or relate operation. To make this a complete question, we append a count function to make it a viable independent question and then add it to the dataset.

Overall with the augmented questions, the number of training questions grew from 700k to 1282k. More importantly there are more short questions of length 3 to 5 that can be used for QA supervision. This helps us test the model generalization to longer question sequences of length 10 to 15+ of the CLEVR validation set.

program length	CLEVR	CLEVR-DC
3	1802	28016
4	10012	77522
5	34283	121402

D Training Schedule

AB-VQA. For the vision extraction we follow the default training settings for Detectron2, with a batch size of 4. The vision models were initialized as multi class for the attributes and minimized the cross entropy with Adam with $lr = 1e^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999,\ \epsilon = 1e^{8}$ (Adam recommended defaults). These were tuned after each lesson epoch and ran for 50 epochs on the correct and abduced data added.

Since we were using categorical data, it was easy to make a new label when flipping the object bit mask from $0 \rightarrow 1$, where we set the corresponding label as active. For the $1 \rightarrow 0$ case, we do not add the data in the multi class case as the required negative label changes is unclear.

For the abduction policy, we use the edit strategy. We set $\epsilon = 0.01$ in our experiments. As training goes on, we expect that the policy improves in selecting $a_t \sim \pi_\theta$. Therefore we are more restrictive on the threshold and increment it by $2e^{-4}$ every lesson epoch.

We use the 3 to 5 length questions from CLEVR-DC, but sampled for each X% QA specified in each experiment. Note that these X% QA is the question percentage over the entire CLEVR-DC data, not just the length 3 to 5 questions. Each lesson contains questions of a unique length of 3, 4, or 5. We run 5 epochs within each lesson. During failed abduction, we exhaustively sample for 2000 iterations. When testing the experiments with no curriculum we use the entire sampled question set and run for 5 epochs as well. This is similarly applied to NS-VQA, but we use a different number of scene graphs.

NS-CL. We kept NS-CL setting the same as in the default paper for the regular and the no curriculum runs. For the burn in data set, we took the 7 scene graphs and generated questions from those. This was done by synthesizing attribute and relation questions on these scenes. For the attribute question synthesis, we enumerated all possible attribute and concepts as a filter operation and followed that by a count operation.

For the relation questions we take the attribute questions that yield a count of 1 and enumerate relation questions on those. This is because we can only ask for objects that satisfy the relation of one subject. The attribute, relation and original questions pertaining to those scene graphs are aggregated and used for burn in.

REINFORCE. This setup uses binary attribute models instead of multi class ones. This is due to the fact that we do induce a negative reward penalty -1, when we do not get a correct answer. Therefore this reward has to be associated with a particular label, which we did not do in the multi class situation with AB-VQA. The negative reward helps the standard REINFORCE update avoid that particular concept in the future.

Additionally, we sample each question for each lesson epoch 30 times and accumulate the gradient. After each question burn in we retrain the related models. If following the standard train after lesson epoch as in AB-VQA, REINFORCE converges very slow. As for 30 sampling versus the 2000 for the AB-VQA, we note that it is hard to compare the two scenarios. This is because with the training after each question in the reinforce setting, the samples improve after each question. Therefore AB-VQA needs more samples to find the same number of corrections before training.

The curriculum length and epochs follow AB-VQA at 3 lessons for 5 epochs each. Even in this setting, we still observe a variance and computation gap highlighted in Figure 4.

All models were trained on a system with a Intel Xeon Silver 4116 CPU, Nvidia 2080Ti 11GB, and 256 GB of RAM.