[RE] LANGUAGE AS AN ABSTRACTION FOR HIERARCHICAL DEEP REINFORCEMENT LEARNING

A PREPRINT

Abaho Katabarwa
Department of Computer Science
Brown University
Providence, RI
abaho_katabarwa@brown.edu

Berkan Hiziroglu
Department of Computer Science
Brown University
Providence, RI
berkan_hiziroglu@brown.edu

Amy Pu
Department of Computer Science
Brown University
Providence, RI
amy_pu@brown.edu

Omer Dai
Department of Computer Science
Brown University
Providence, RI
omer_dai@brown.edu

December 1, 2019

Abstract

We tackle the issue of long-horizon planning and temporally-extended tasks in our replication, using language as abstraction for hierarchical reinforcement learning. The proposed approach selects language as the choice of abstraction because of its compositional structure, ensuring an ability to break down tasks into smaller sub-tasks. The authors train a low-level policy and high-level policy using an interactive environment built using the MuJoCo physics engine and the CLEVR engine. The authors show that using language as the framework between low-level policy and high-level policy allows the agent to learn complex tasks requiring long term planning, including object sorting and multi-object rearrangement. We focused on implementing and training the low-level policy from scratch, as that is where HIR is first introduced. For the low-level policy, we show that encoding the instruction with a GRU and using HIR performs better than a one-hot encoded representation of the instruction. However, our results for one-hot encoded representation as the number of total instructions grew contradicted what the conclusions from the original paper.

1 Introduction

Deep reinforcement learning faces open problems of long-horizon planning and receiving instructions from a human. Hierarchical reinforcement learning (HRL) attempts to solve tasks requiring long term planning. Agents utilize a hierarchy of policies to learn complex skills and accomplish tasks. The high-level policy completes the long term tasks by directing the low-level policy and generating goals to satisfy some simpler objective. The low-level policy address more short term and simpler tasks, used in a combination to fulfill the high-level policy's objective.

In this paper, we reproduce the paper of Jiang et al [11]. The authors present a new framework for incorporating language abstractions into hierarchical reinforcement learning, using a low-level policy that follows language instructions and a high-level policy that can produce actions in the space of

language. The low-level policy's objective is to manipulate specific objects in the environment such that a description is satisfied, and the high-level policy's objective is to instruct the low-level policy by generating goals to satisfy some reconfiguration or sorting of the objects in the environment. One of the shortcomings of HRL is that rewards are often infrequent, so to address this problem, the authors develop a procedure called Hindsight Instruction Relabeling (HIR), an adaptation of Hindsight Experience Replay (HER) [8].

2 Related Work

Using abstractions are especially important in reinforcement learning to solve MDPs to avoid hand-engineered methods for specific problems and to build agents which are effective against broader range of problems [3, 4]. Q-learning algorithm is one of the most popular RL algorithms to solve MDPs, but it is not always effective because it suffers from scaling where the MDP may be too complex due to large state space and/or sparse rewards in the environment.

Hierarchical reinforcement learning tries to solve this issue by creating an abstraction between solving the sub-problems by using low-level policies and the main problem by learning a high-level policy [3, 4]. The classical approach of HRL focuses on learning the high-level policy which choose a set of hard coded low-level sub-policies [7]. This approach suffers from generality because we may not know enough about the problem to hand-engineer sub policies; even if we did, it could take an unreasonable amount of time to implement which would make this approach infeasible. A more general learning method is using the options framework [9] where the high-level policy and low-level sub-policies are learned separately which provides better abstractions between the policies because the only communication between the high-policy and the low-level policy is satisfying the sub-goal. Bacon et al. [7] have approached HRL using the option-critic architecture using Deep Q-learning, learning low-level policies directly from final task rewards. However, this approach suffers when rewards of the environment are sparse because it can seriously hinder the agent's low-level learning. To contrast, Jiang et al. [11] demonstrate how language can be used to represent complex tasks, and Wu et al. [12] show that language can improve performance when compared with naïve representations.

Moreover, it is known that a vanilla DQN can over-estimate action values when the action values are not accurate with respect to the source of the approximation error. If these optimistic values are not uniform and are not concentrated on the states which we wish to explore, it can lead to sub-optimal policies. Double DQN solves this issue which is used by the original authors to train high and low level policies [2].

3 Implementation

The CLEVR-Robot environment engine was made available by the authors [10]. The engine was written in Python. We replicated the architecture presented in the paper using PyTorch. Our goal was to reproduce the graphs produced solely by the low-level policy, so our project only includes the implementation and the experiments of the low-level policy. The overarching architecture of the low-level policy was the Double DQN model described in Deep Reinforcement Learning with Double Q-learning" [2]. $\tau = 0.05$, $\gamma = 0.993$ were used as parameters to the Double DQN model as stated in the paper. The primary and target model within the Double DQN model contained our conception of the feed-forward network described in the paper. While the feed-forward model was singular in that it took as inputs a state representation of the environment and a target question, two slightly different sub models were used in order to create the tensor representation of the target input question. One represented the input question as an embedding vector and the another represented the input question as a one-hot vector.

The embedding vector model took the input question as input and transformed each word in the question into a embedding vector of size 30. These embedding vectors were fed to a GRU layer. The GRU's final state was used as the representation of the question. This vector was concatenated with the direct state observation as depicted in Figure 7 in the original paper. The resulting matrix is the input to the Double DQN model.

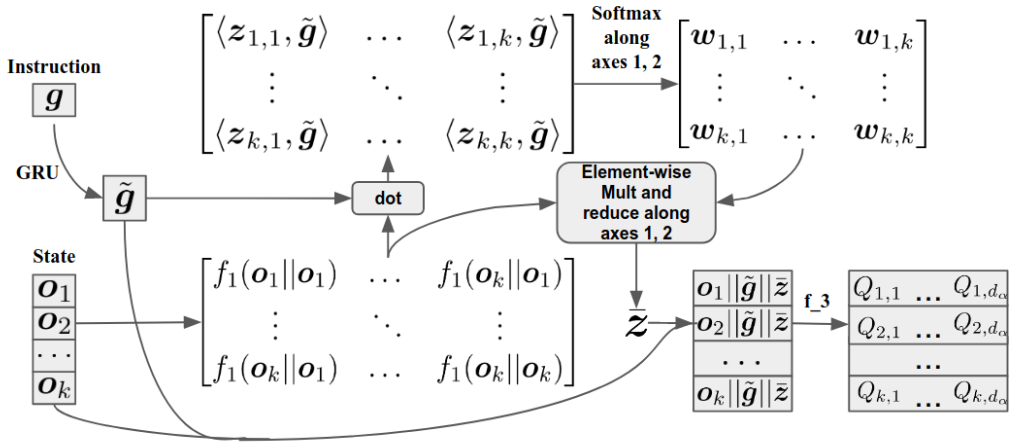


Figure 1: Computation graph of the state-based low level policy (from the original paper).

The one hot vector model's initialization indexed every question in the question bank and used a bin number to determine the size of the one-hot vectors. The one-hot vector size was $|questions\ in\ bank| * num\_bins$. Given a question, the one hot vector was determined by sampling uniformly from the all the possible one-hot representations of the question within its bin. Subsequently, this one-hot vector was passed through a linear layer whose output was used as the output representation of the question.

Given the state and the vector representation of the target question, O and E respectively, the DQN model first builds a Z matrix which is constructed of the every pairing of every object in $o_i|o_j \in O$ passed through a linear layer $f1$. Therefore the elements of the Z are $f1(o_i|o_j), \forall ij \in range(|O|)$. Now an attention matrix P is created for Z. P is constructed by $softmax(\forall z \in Z, E \cdot z)$. $\hat{Z} = E * Z$ Finally for each object in the environment, $\hat{Z}$, Z, E, are concatenated to form the input to the final linear layer $F2$ that will produce the action space probabilities.

Figure 1 shows the architecture for the state-based low level policy. The output dimensions of the f_1 function is not stated. Moreover, reducing a 2d matrix along axes 1,2 would output a scalar value. However, the original authors state that $\bar{Z}$ is a fixed size vector. Although this seems like a minor detail, this could be the main reason between our results and the original results.

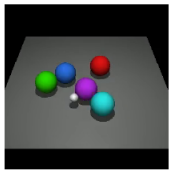
4 Reproducibility

4.1 Methodology

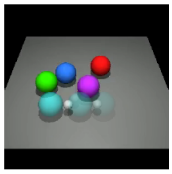
As part of the reproducibility challenge, we would like to share our architecture, experiments and results. In addition to these, we would also like to suggest minor modifications as to better reproduce the results from the paper and possibly provide some improvement techniques. Specifically, we want to replicate the exact settings for the results shown in Figures 2 and 3. Although the paper was missing few key implementation details, we think that the paper was well written overall. The figures given provided a better understanding of the architecture.

4.2 Dataset

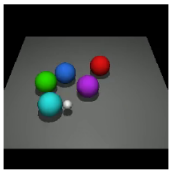
The environment engine provided by the authors also included the instruction set that was used for training and testing of the agent (Figure 1). We followed the standard approach of random split of the 600 instructions into train and test sets. The number of effective instructions used in training were



(a) Goal is $\boldsymbol{g}_0$: *"There is a **red** ball; are there any matte **cyan** sphere **right** of it?"*. Currently $\Psi(\boldsymbol{s}_t, \boldsymbol{g}_0) = 0$



(b) Agent performs actions and interacts with the environment and tries to satisfy goal.



(c) Resulting state $\boldsymbol{s}_{t+1}$ does not satisfy $\boldsymbol{g}_0$, so relabel goal to $\boldsymbol{g}'$: *" There is a **green** sphere; are there any rubber **cyan** balls **behind** it?"* so $\Psi(\boldsymbol{s}_{t+1}, \boldsymbol{g}') = 1$

Figure 2: Environment and some instructions (from the original paper).

changing dynamically depending on the bin number for one-hot encoding scheme. It was still for the non-compositional one.

4.3 Details

The performance evaluation of the agent on the test set is reported in Table 1 in the original paper. Due to computational limits, we are not reporting the final performance graph of the low-level policy on different training and test instruction distributions but we are providing the training performance graph that is depicted in Figure 4 in the original paper. We show that the proposed architecture and methodology for hierarchical reinforcement learning show practical potential.

4.4 Cost

The original paper state that the training time is 2 days for the state-based low-level policy. We only experimented with the low-level policy so we are only going to report the required computational cost for those experiments. The instruction set size is very small, only 600 questions. However, the unique architecture of this model prevents parallel computation and hence the usage of GPUs do not decrease the training time significantly. The construction of state-based low policy requires a function f_1 to be applied to every element of the Z matrix which is a pair-wise concatenation of the objects. These operations cannot be broadcasted or run in parallel and therefore the computational complexity of the training time grows linearly with respect to the model parameters.

On the other hand, the environment itself is very costly to run. In Figure 2 of the original paper, the authors reported that they trained for 2 million training steps. Using the given original hyperparameters for epochs, cycles, episodes, the authors aimed to train for a total of 12.5 million training steps, but never reached that far. One call of the step function in the provided environment takes around 0.02 seconds, making 2 million training steps takes around 12 hours. This time does not include the inference, the loss calculation and the update of the model parameters. So, it would normally take more than 2 days just to interact with the environment without any training if the authors did not stop the training early. We believe it is reasonable to say that the experiments reported in this paper are very costly in terms of time.

5 Results

In this section we will describe the results of our experiments and compare the proposed architectures' performances. Our experiments centered on the replication of the first two graphs shown in figure 4 in the original paper. The first experiment attempted to replicate the curve labeled HIR 600 and the second attempted to replicate the second graph in figure 4 in the original paper.

5.1 HIR 600 training

Results for low-level policies show us that HIR-model reaches the highest number of instructions fulfilled in an episode. In the original paper, as the number of training instructions increase, the performance of the HIR-model also increases. The original graph is shown below.

We, however, were unable to replicate this result as we do not have the information about paraphrasing and synonym substitution. The HIR model performs better than the one-hot encoded representation model in our experiments and this confirms the original paper.

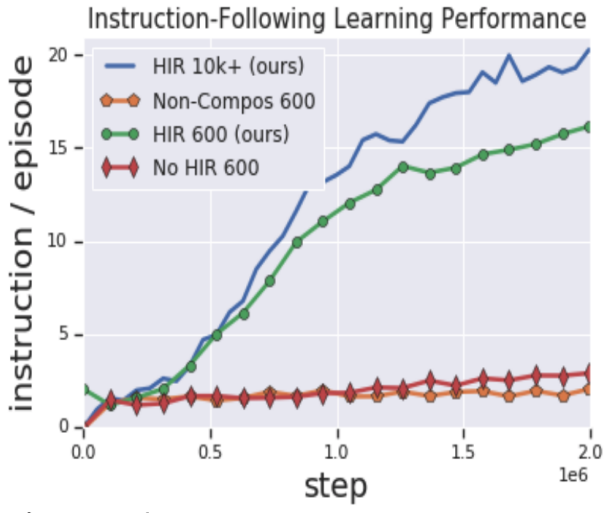


Figure 3: HIR with different number of instructions and results with non-compositional representation and with no relabeling (from the original paper).

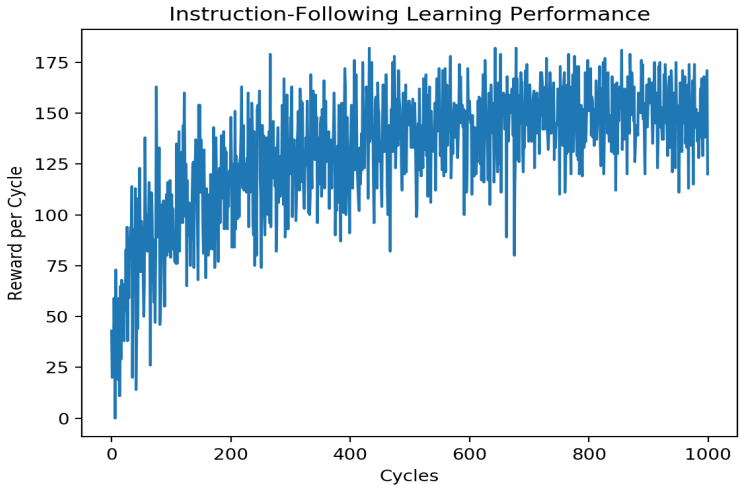


Figure 4: HIR 600 Training

5.2 One-hot encoding with instruction sets

We attempted to recreate the results show in the second/middle graph of Figure 4 in the paper. This figure is shown below.

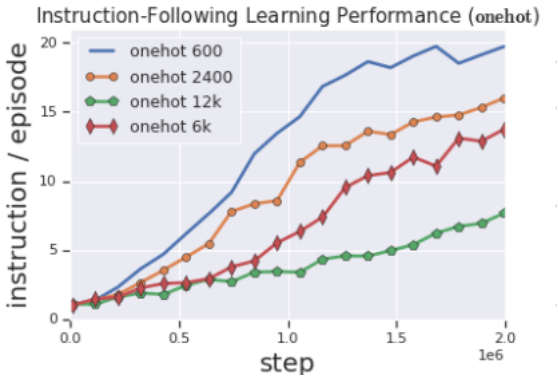


Figure 5: Middle graph from figure 4 from original paper. Performance suffers as instruction set grows

The purpose of this graph was to demonstrate that as the instruction set grows performance suffers, since the model cannot leverage the compositionality of the language. Because we were not able to train these models for the $2 * 1e6$ steps shown in due to cost and time constraints, we trained our models for 1000 cycles. Furthermore, our graph shows reward per cycle since rewards per episode were often low and the variance was high. Our results contradict the conclusions made in the paper. In our experiments, as the instruction set grew, performance increased, but in the original paper, performance decreased as the instruction set grew. "one hot 1 bin" corresponds to "onehot 600, "one hot 4 bin" corresponds to "onehot 2400", and "one hot 10 bin" corresponds to "onehot 6k".

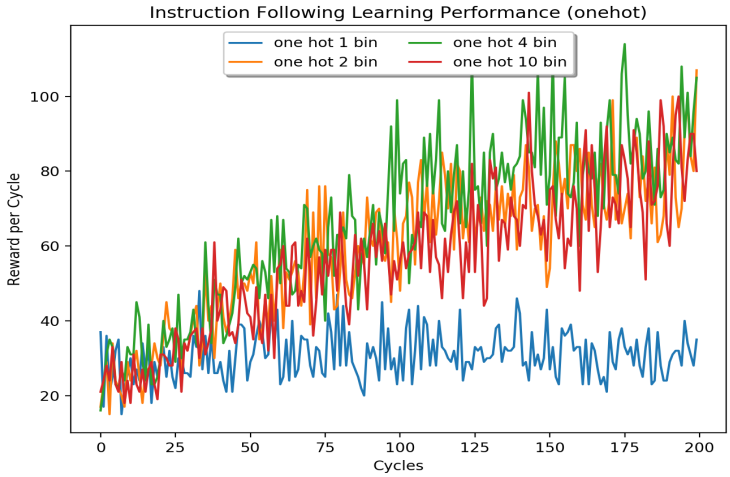


Figure 6: Replication of one-hot encoded representations.

6 Conclusion

Overall, the paper clearly explains the framework for using language as abstraction for hierarchical deep learning and the HIR procedure. We were able to train the low-level policy using the instructions provided by the environment and implement Hindsight Instruction Relabeling. We were also able to show that changing the bin size does in fact affect the agent's ability to learn. From our experiments, 1 bin per instruction performed the worst, whereas 4 bins per instruction performed the best. Regardless, we have shown that one-hot encoded representation model faces challenges in scaling, even when the underlying number of instructions does not change. Moreover, we have shown that HIR with 600 instructions achieves higher reward than the one-hot encoded representation.

For future improvement of our results, we hope to train HIR with more than 10,000 instructions by paraphrasing and using synonyms and train the low-level policy on the test set to ensure that it is able to generalize on unseen instructions for HIR. We also hope to scale up and beyond the CLEVR environment so as to increase the body of instructions used in training the low-level policy.

7 Acknowledgements

We would like to thank to Evan Cater for clarifying aspects in implementing HIR and mentoring us throughout this process. We would also like to thank Prof. Michael Littman for providing the required resources for this project.

References

[1] Qianqian Fang and David A. Boas, "Monte Carlo Simulation of Photon Migration in 3D Turbid Media Accelerated by Graphics Processing Units", Opt. Express 17, 20178-20190 (2009)

[2] Hado van Hasselt and Arthur Guez and David Silver, "Deep Reinforcement Learning with Double Q-learning", CoRR, abs/1509.06461, 2015,http://arxiv.org/abs/1509.06461

[3] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In Advances in neural information processing systems, pages 271–278, 1993.

[4] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In Advances in neural information processing systems, pages 1043–1049, 1998.

[5] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In International Symposium on abstraction, reformulation, and approximation, pages 212–223. Springer, 2002.

[6] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artificial intelligence, 112(1-2) 181–211, 1999.

[7] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In AAAI, pages 1726–1734, 2017.

[8] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.

[9] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. arXiv preprint arXiv:1709.04571, 2017.

[10] CLEVR-Robot Environment, 2019, GitHub, GitHub repository, `https://github.com/google-research/clevr_robot`, 38bf0bc3e06bc169e92478f33eae57fa71836309

[11] Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. CoRR, abs/1906.07343, 2019.

[12] Yuhuai Wu, Harris Chan, Jamie Kiros, Sanja Fidler, and Jimmy Ba. ACTRCE: Augmenting experience via teacher's advice, 2019. URL https://openreview.net/forum?id=HyM8V2A9Km.

8 Supplemental Material

Code available at `https://github.com/abahocodes/cs2951f_final_project`.