Published in the proceedings of the Workshop on “Structure & Priors in Reinforcement Learning” at ICLR 2019

LANGUAGE AS AN ABSTRACTION FOR HIERARCHICAL DEEP REINFORCEMENT LEARNING

Yiding Jiang[*], **Shixiang Gu, Kevin Murphy, Chelsea Finn**
Google Research
Mountain View, CA 94043, USA
{ydjiang,shanegu,kpmurphy,chelseaf}@google.com

1 INTRODUCTION

Deep reinforcement learning has seen remarkable advancements in recent times. For example, reinforcement learning agents can solve difficult continuous control tasks (Schulman et al., 2015; Lillicrap et al., 2015b; Gu et al., 2017; Heess et al., 2017) and achieve impressive performance on many challenging games such as Atari games (Mnih et al., 2015) and Go (Silver et al., 2017). The generality of reinforcement learning makes it an ideal framework for optimizing a wide range of sequential decision-making problems. However, many challenges remain for applications of reinforcement learning. Questions such as long-horizon planning or receiving instructions from a human in the loop are still open problems. In this work, we are interested in building intelligent agents that are able to learn complex, temporally extended skills (e.g. arranging multiple objects) and, at the same time, reason using language. [1]

One approach that attempts to address the temporally extended nature of tasks is *hierarchical reinforcement learning* (HRL) (Dayan & Hinton, 1993; Parr & Russell, 1998). In standard settings of HRL, the agents leverage a hierarchy of policies with varying levels of temporal and behavioral abstractions, where the high-level policy achieves long-horizon tasks by controlling the low-level policies, which focus on more short-term tasks such as moving in a specific direction. While, in principle, HRL is a promising framework for solving temporally extended tasks, it is highly non-trivial to apply it in practice. One challenge of applying HRL is to choose the appropriate abstraction. Hard-coded abstractions often lack modeling flexibility and are task-specific (Sutton et al., 1999; Konidaris & Barto, 2007; Heess et al., 2016; Peng et al., 2017), while learned abstractions require carefully-tuned regularization (Bacon et al., 2017; Harb et al., 2017). One possible solution is to have the higher-level policy generate a goal state and have the low-level policy try to reach that goal state. Intuitively, a goal state can be seen as an instruction to the low-level policy. So far, most works in goal-conditioned reinforcement learning have used simple goal representations such as points in the state space, which does not scale well into high dimensional state space (Schaul et al., 2015; Andrychowicz et al., 2017). For instance, in the visual domain, generating high fidelity images is still an area of active research.

In contrast, language is a flexible representation for transferring a variety of ideas and intentions with minimal assumptions about the problem setting, while its compositional nature makes it a powerful abstraction for transferring knowledge and instructions (Grice, 1975; Mordatch & Abbeel, 2018). In this work, we propose to use language as the interface between high- and low-level policies in hierarchical reinforcement learning. With a low-level policy that follows instructions, the high-level policy can produce actions in the space of language, giving us a number of appealing benefits. First, the low-level policy can be re-used for different high-level objectives without retraining. Second, the high-level policies are human interpretable as the action space consists of instructions in language; the low-level policy that understands language instruction also allows human to communicate with the agent, making it easier to recognize and diagnose failures. Finally, studies have also suggested that language plays an important role as an abstraction for human reasoning and planning (Gleitman & Papafragou, 2005; Piantadosi et al., 2012). In fact, the majority of knowledge learning and skill acquisition we do are through languages throughout our life.

While language is an appealing choice as the abstraction for HRL, training a low-level policy that is capable of following language instruction is a non-trivial problem as it involves learning from

[*]Work done as part of the Google AI Residency Program
[1]Videos and supplementary materials can be found at https://sites.google.com/view/halsite

binary reward signals that indicate whether or not the instruction was completed. To address this problem, we extend and generalize previous work on goal relabeling to the space of language goals, allowing the agent to learn from many language instructions at once. We evaluate on a new interactive environment that consists of procedurally-generated scenes of objects that are paired with rich programatically-generated language descriptions. The low-level policy's objective is to manipulate the objects in the scene such that a description or statement is satisfied by the arrangement of objects in the scene. On high-level object reconfiguration and sorting tasks, we find that language abstractions and hindsight instruction relabeling are critical for learning a wide variety of these temporally-extended skills.

In summary, the main contribution of our work is three-fold:

1. a framework for incorporating language abstractions into hierarchical RL, with which we find that the structure and flexibility of language enables agents to solve a variety of long-horizon control problems.
2. an algorithm for effectively training a multi-task reinforcement learning agent to complete language-based instructions through *hindsight instruction relabeling*.
3. an interactive continuous control environment integrated with language tasks inspired by the CLEVR dataset (Johnson et al., 2017), built on MuJoCo physics simulator (Todorov et al., 2012).

2 Hierarchical Reinforcement Learning with Language Abstractions

In this section, we present our framework *Hierarchical Abstraction with Language* (HAL) for training a 2-layer hierarchical policy with language as the abstraction between the high-level policy and the low-level policy. We open the exposition with formalizing the problem of solving temporally extended task with language with HRL. Then we discuss how to train the low-level policy, $\pi_l(\boldsymbol{a}|\boldsymbol{s}_t, \boldsymbol{g})$ conditioned on language instructions, and how a high-level policy, $\pi_h(\boldsymbol{g}|\boldsymbol{s}_t)$, can be trained using such a low-level policy.

2.1 Problem statement

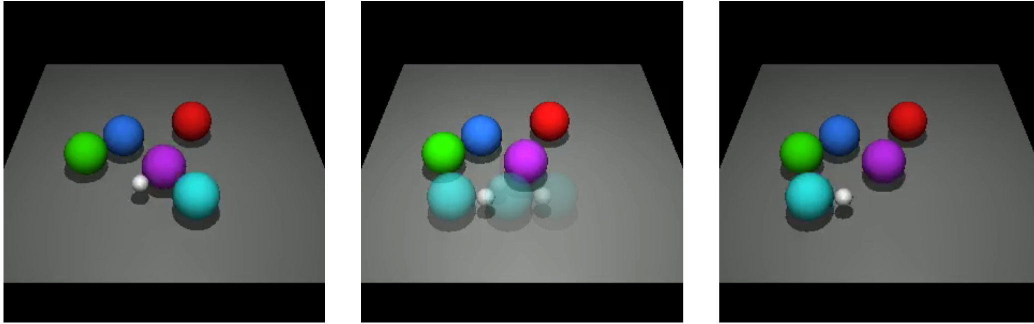
We adopt an object-oriented representation of states by assuming $\boldsymbol{s}_t = \{\boldsymbol{o}_i\}_{i=1}^{k_t}$, where $\boldsymbol{o}_i \in \mathbb{R}^{d_o}$ is the representation of object i, and k_t is the number of objects (which can vary over time). $\mathbb{S}$ is the space of all possible $\boldsymbol{s}$. We also assume $\boldsymbol{a}_t = \{\boldsymbol{\alpha}_i\}_{i=1}^{k_t}$, where each $\boldsymbol{\alpha}_i \in \mathbb{R}^{d_{\boldsymbol{\alpha}}}$ acts on individual the object $\boldsymbol{o}_i$ and denote the space of all possible action as $\mathbb{A}$. One generic scenario for complex decision making problems is one where the agent needs to interact with multiple entities or objects in the world. A simplified instance that captures this property is the Multi-Armed Bandits. However, in many problems, the order of the actions matter and the number of objects can vary. Our formulation captures these characteristics. Finally we assume there is an oracle Ω that can map states to language statements. Such oracle can be a program that is executed on $\boldsymbol{s}_t$ similar to CLEVR, a captioning algorithm for image or other state representation if the ground truth (e.g. deployment outside of simulation) is not available at training time or even a human supervisor. Concretely, we assume $\Omega(\boldsymbol{s}_t) = \{\boldsymbol{g}_i\}_{i=1}^{n}$, so we can specify goals as subset of these statements. Implementation details of oracle we use can be found in appendix A.1.3.

More precisely, we consider $\mathbb{G}$ to be the space of task-relevant Boolean functions whose input is $\boldsymbol{s}_t$, i.e. $\boldsymbol{g} : \mathbb{S} \rightarrow \{0, 1\}$. More specifically, we take $\mathbb{G}$ to be a set of language statements that can be evaluated to be either True or False on $\boldsymbol{s}_t$. The value of $\boldsymbol{g}(\boldsymbol{s}_t)$ is 1 if $\boldsymbol{s}_t$ satisfies the goal, or 0 if the $\boldsymbol{s}_t$ does not satisfy the statement. Any goal specified in the state space can be easily expressed by a Boolean function of this form by checking if two states match or are close to each other.

The low-level policy's objective is to solve an augmented MDP where $\mathbb{G}$ is the range of the oracle Ω. For simplicity, we assume that Ω's output is uniform over $\mathbb{G}$. The high-level policy solves a standard MDP whose state space is the same $\mathbb{S}$ as the low-level policy, action space is $\mathbb{G}$, and reward is sparse. The high-level policy and low-level policy are trained separately, so the same low-level policy can be reused for different high-level policies. Jointly fine-tuning the low-level policy with a

specific high-level policy is certainly possible, but we found separate training to be satisfactory even without any fine-tuning. We leave exploring this direction to future work.

2.2 Training a language conditioned low-level policy



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

(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 $\boldsymbol{g}'(\boldsymbol{s}_{t+1})$ = *True*

Figure 1: Illustration of hindsight instruction relabeling (HIR), which we use to enable the agent to learn from many different language goals at once.

A straightforward way to represent the reward for the low level policy would be $R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) = \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)}[\boldsymbol{g}(\boldsymbol{s}_{t+1})]$ or, with a deterministic transition model, $R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}) = \boldsymbol{g}(\boldsymbol{s}_{t+1})$. However, optimizing with this reward directly is difficult because the reward signal is only non-zero when the goal is achieved. Unlike prior work, which uses a state vector as the goal, it is hard to define distance in the space of language statements and, consequently, difficult to make the reward signal smooth by giving partial credits such as the ℓ_p norm of the difference between 2 states. To overcome these difficulties, we use a relabeling technique we call *hindsight instruction relabeling* (HIR): Instead of relabeling the trajectory with states reached, we relabel the trajectory with the elements of $\Omega(\boldsymbol{s}_t)$ as the goals[2]. For more details, see Algorithm 1 for the pseudocode of HIR and Figure 1 for an illustrated example.

Note that multiple statements can simultaneously be changed from False to True and every statement can be paraphrased in numerous ways through changing words with their synonyms, rearranging the order of the words or deleting parts of the corpus. Due to these properties of language, using the dynamic statements as instructions opens up a wide design space for the relabeling schemes (A.3).

2.3 Training a high-level language policy

With a trained low-level policy $\pi_l(\boldsymbol{a}|\boldsymbol{s}_t, \boldsymbol{g})$, the high-level policy can instruct the low-level policy by generating the goals. This allows the high-level policy to structurally explore with actions that are semantically meaningful and span multiple time steps. Pseudocode for the algorithm can be found in the appendix (Algorithm 2).

In principle, the high-level policy, $\pi_h(\boldsymbol{g}|\boldsymbol{s})$, can be trained with any reinforcement learning algorithm, given a suitable way to generate sentences for the goals. However, generating coherent discrete data such as sentences is still very much an open problem even in the unsupervised learning settings. Training a language generative model with existing reinforcement learning algorithms is unlikely to achieve favorable results. The dimensionality of language also makes the *discretized* action space for the high-level policy prohibitively high. Fortunately, while the the size of the instructions space $\mathbb{G}$ scales exponentially with the size of the vocabulary, the elements of $\mathbb{G}$ are naturally redundant – many elements correspond to effectively the same underlying instruction with different synonyms or grammar. While the low-level policy understands all the different instructions, the

[2]Note that we use the terms "*instruction*" and "*goal*" interchangeably as a goal can be thought of as an instruction about where to go or what to do.

Algorithm 1 Hindsight instruction relabeling (HIR)

1: **Inputs:** off-policy RL algorithm $\mathcal{A}$ with parameter ϕ; relabeling strategy $\mathcal{S}$; language statement oracle Ω; environment $\mathcal{E}$; replay buffer $\mathbb{B}$; number of relabeled future K
2: **Initialize** $\mathcal{A}, \mathbb{B}, \phi$
3: **for** episode $i = 1$ **to** M **do**
4: $\quad \boldsymbol{s}_0 \leftarrow$ reset $\mathcal{E}$
5: $\quad \boldsymbol{g} \sim \mathcal{U}(\{\boldsymbol{g} \in \Omega(\boldsymbol{s}_0) | \boldsymbol{g}(\boldsymbol{s}_0) = 0\})$
6: $\quad \tau \leftarrow [\,]$
7: $\quad$ **for** step $t = 0$ **to** T **do**
8: $\quad\quad \mathbb{U}_t \leftarrow \{\boldsymbol{g} \in \Omega(\boldsymbol{s}_t) | \boldsymbol{g}(\boldsymbol{s}_t) = 0\}$
9: $\quad\quad \boldsymbol{a}_t \sim \pi_{\mathcal{A}}(\boldsymbol{a}|\boldsymbol{s}_t, \boldsymbol{g})$
10: $\quad\quad \boldsymbol{s}_{t+1} \leftarrow$ Take action $\boldsymbol{a}_t$ from $\boldsymbol{s}_t$
11: $\quad\quad r_t \leftarrow \boldsymbol{g}(\boldsymbol{s}_{t+1})$
12: $\quad\quad \mathbb{V}_t \leftarrow \mathbb{U}_t \setminus \{\boldsymbol{g} \in \Omega(\boldsymbol{s}_{t+1}) | \boldsymbol{g}(\boldsymbol{s}_{t+1}) = 0\}$
13: $\quad\quad$ add $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}, r_t, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \mathbb{V}_t)$ to τ
14: $\quad\quad$ **if** $r_t = 1$ **then**
15: $\quad\quad\quad \boldsymbol{g} \sim \mathcal{U}(\{\boldsymbol{g} \in \Omega(\boldsymbol{s}_{t+1}) | \boldsymbol{g}(\boldsymbol{s}_{t+1}) = 0\})$
16: $\quad\quad$ **end if**
17: $\quad$ **end for**
18: $\quad$ **for** step $t = 0$ **to** T **do**
19: $\quad\quad$ Store $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}, r_t, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})$ in $\mathbb{B}$
20: $\quad\quad$ **for** $\boldsymbol{g}' \in \mathbb{V}_t$ **do**
21: $\quad\quad\quad$ Store $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}', 1, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})$ in $\mathbb{B}$
22: $\quad\quad$ **end for**
23: $\quad\quad \mathbb{W} \leftarrow \mathcal{S}(\tau, t, K)^{\dagger}$
24: $\quad\quad$ **for** $(\boldsymbol{g}', r') \in \mathbb{W}$ **do**
25: $\quad\quad\quad$ Store $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{g}', r', \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})$ in $\mathbb{B}$
26: $\quad\quad$ **end for**
27: $\quad$ **end for**
28: $\quad$ Update $\mathcal{A}$, ϕ for N steps using minibtach from $\mathbb{B}$
29: **end for**
30: † Details in appendix A.3.

high-level policy only needs to generate instruction from a much smaller subset of $\mathbb{G}$ to direct the low-level policy in many cases. We denote such subsets of $\mathbb{G}$ as $\mathbb{I}$.

If $\mathbb{I}$ is sufficiently small, the problem can be recast as an RL problem with a discrete action space, and can be solved with algorithms such as DQN (Mnih et al., 2015). This is the approach we adopt in this work. However, there are a number of alternative options for optimizing the high-level policy, such as using using continuous action or language embedding (Lillicrap et al., 2015a; Gu et al., 2016; Schulman et al., 2017) in combination with appropriate recurrent or autoregressive decoders (Sutskever et al., 2014; Dauphin et al., 2016).

As the instruction often represents a sequence of low-level actions, we take T' actions with the low-level policy for every high-level instruction. T' can be a fixed number of steps, or computed dynamically by a terminal policy learned by the low-level policy like the option framework. While the latter is a more principled approach, we found rolling out a fixed 5 steps to be sufficient in our experiments. Further, the choice of termination condition is a design choice that is orthogonal to the training of the high-level policy. We leave exploring the termination policy to future work.

3 EXPERIMENTS

3.1 LOW-LEVEL POLICY

We evaluate the performance of our low-level policy by the average number of instructions it can successfully achieve each episode (100 steps) measured over 100 episodes. We demonstrate the importance of HIR by comparing agents that are trained with and without relabeling.

We also demonstrate the benefit of using language as an abstraction compared to a more naive goal parameterization based on one-hot encodings. To do this, we start with 600 instructions, which we paraphrase and modify by swapping synonyms, resulting in about 10,000 total instructions. For the one-hot encoding, we assign each instruction a varying number of bins in the one-hot vector. Concretely, we give each instruction of the 600 instruction 1, 4, 10, and 20 bins in the one-hot vector, which means the effective size of the one-hot vector is 600, 2400, 6000 and 12000. When sampling goals, each goal is uniformly dropped into one of its corresponding bins. The one-hot encoding low-level policy is trained with HER. As shown in Figure 2 (middle), the performance of the algorithm deteriorates as the number of instructions increase. Figure 2 (left) also shows, counter-intuitively, that HIR with more instructions (10000+ vs 600) learn faster. We hypothesize the reason to be that with more diversity in sentences, the language model can learn to generalize better rather

than to memorize, thereby speeding up the learning (Zhang et al., 2016). Figure 2 (right) indeed shows good generalization performance on unseen instructions based on splitting the instruction sets train (70%) to test (30%).

To evaluate the importance of compositional structure of language, we compare HIR with compositional language to HIR with non-compositional language representation. The non-compositional language representation is obtained by training a Seq2Seq autoencoder (Sutskever et al., 2014) with GRU and 64 hidden units on the 600 instructions. This autoencoder achieves near 0 cross-entropy error across the entire train set and achieves perfect token-level reconstruction. This implies that all information of the sentence is contained in the bottleneck layer. Note that this representation contains the same information as the original language and the only difference is that the compositional structure is gone. We found that this autoencoded representation performs worse than all compositional approaches.

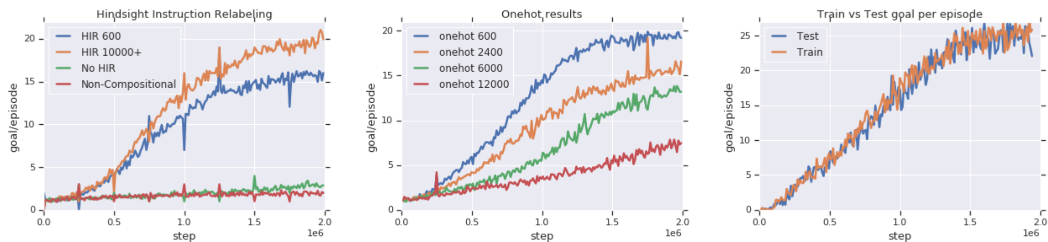
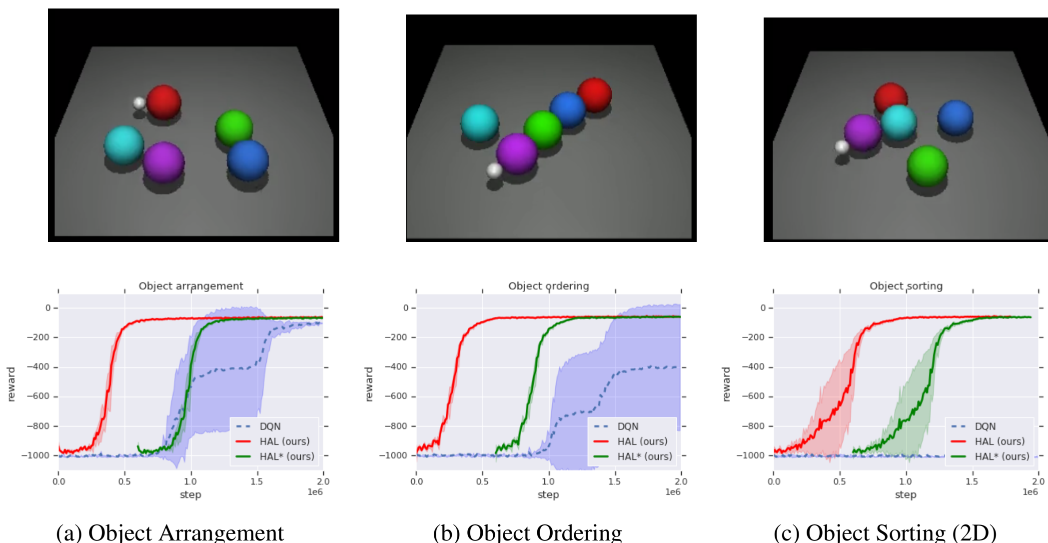


Figure 2: Results for low-level policies in terms of goals accomplished per episode over training steps for HIR on language representation (left) v.s. one-hot representation (middle). Since the one-hot cannot leverage compositionality of the language, it suffers significantly as instruction sets grow, while HIR on sentences in fact learns even faster when instruction sets increase. Figure (right) shows train v.s. test performances of low-level policies based on a split of instruction sets, showing very good generalizations on unseen instructions for HIR. Non-compositional representation (left red) performs worse than all compositional representation.

3.2 High-level policy



(a) Object Arrangement (b) Object Ordering (c) Object Sorting (2D)

Figure 3: Results for high-level policy on the proposed 3 different tasks. Green curves for HAL include the steps for training the low-level policy. In all settings, HAL demonstrates faster and more stable learning than the baseline DQN with a better asymptotic performance.

We evaluate HAL with 80 instructions (i.e. $|\mathbb{A}_h| = 80$) on 3 tasks with highly sparse binary reward by comparing to a vanilla DQN that uses the low-level policy's action space, which is to push individual object in one of the 8 cardinal directions (i.e. $|\mathbb{A}_{nh}| = 8 \times k$ where k is the number of

objects in the scene and k is 5 for all 3 tasks). A possible goal configuration of solving these tasks are in Figure 3 (top); however, only relative spatial relationship is considered so there exist many different solutions. A more detailed description of these tasks can be found in appendix A.1.2.

Plotted in Figure 3 (bottom) are cumulated reward per episode averaged over 3 runs with standard deviation plotted in Figure 3. In all 3 tasks, HAL (red curves) converges faster than the DQN baseline and shows much smaller variance in the performance. In the arrangement task, HAL performs asymptotically slightly better than DQN while in object ordering and object sorting HAL's performance is superior in both convergence speed and final performance.

Note that while a *single* trained low-level policy is used across *all* 3 tasks, a case can be made for the training time of the low-level policy to be taken into account when measuring the performance of the high-level policy. As such, we use a low-level policy that is trained for 6×10^5 steps with approximate performance of 5 goals/episode. The training curves adjusted with training step of the low-level policy is colored green in Figure 3. Even with an imperfect low-level policy, HAL still compares favorably against the DQN baseline.

REFERENCES

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*, pp. 5048–5058, 2017.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pp. 1726–1734, 2017.

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks, 2016.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pp. 271–278, 1993.

Lila Gleitman and Anna Papafragou. Language and thought. *Cambridge handbook of thinking and reasoning*, pp. 633–661, 2005.

H Paul Grice. Logic and conversation. *1975*, pp. 41–58, 1975.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pp. 2829–2838, 2016.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 3389–3396. IEEE, 2017.

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.

Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*, 2016.

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

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*, pp. 2901–2910, 2017.

George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pp. 895–900, 2007.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2015a.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel van de Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2017)*, 36(4), 2017.

Steven T Piantadosi, Joshua B Tenenbaum, and Noah D Goodman. Bootstrapping in a language of thought: A formal model of numerical concept learning. *Cognition*, 123(2):199–217, 2012.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

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.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

A APPENDIX

A.1 ENVIRONMENT AND TASKS

In this section, we present the details about the environment we use, the language oracle, and the high-level tasks of interest.

A.1.1 MUJOCO INTERACTIVE ENVIRONMENT

In robotics, manipulating and rearranging objects is a fundamental way through which robots interact with the environment, which is often cluttered and unstructured. To succeed in these environments, the agents must be able to handle different number of objects with varying properties. Our environment contains up to 5 objects, and each object is uniquely indexed by an integer i. We will refer to all the elements in the environment collectively as the *world state*.

Each object is represented by $\boldsymbol{o}_i$ that contains the *3d coordinate*, $\boldsymbol{p}_i$, of its center of mass, and a one-hot representation of its 4 properties: *color, shape, size*, and *material*. The environment keeps an internal relation graph $\mathcal{G}_{adj}$ for all the objects currently in the scene. The relation graph is stored as an adjacency list whose i^{th} entry is a nested array storing $\boldsymbol{o}_i$'s neighbors in 4 cardinal directions *left, right, front* and *behind*. The criterion for $\boldsymbol{o}_j$ to be the neighbor of $\boldsymbol{o}_i$ in certain direction is if $||\boldsymbol{p}_j - \boldsymbol{p}_i|| \leq r_{\max}$ and the angle between $\boldsymbol{p}_j - \boldsymbol{p}_i$ and the cardinal vector of that vector is smaller than $\beta_{\max}$. After every interaction between the agent and the environment, o_i and the relation graph are updated to reflect the current world state.

Before each interaction, the environment stores a set of language statements that are not satisfied by the current world state. These statements are re-evaluated after the interaction. The statements whose values change to True during the interaction can be used as the goals or instructions for relabeling the trajectories (cf. pre and post conditions used in classical AI planning). Assuming the low-level policy only follows a single instruction at any given instant, the reward for every transition is 1 if the goal is achieved and 0 otherwise. The action space we use in this work consists of a point mass agent pushing one object in 1 of the 8 cardinal directions for a fixed number of frames, so the discrete action space has size $8k_t$, where $k_t \leq 5$ is the number of objects.

A.1.2 HIGH LEVEL TASKS

The high-level policy's reward function can be tailored towards the task of interests, where we propose three difficult benchmark tasks with extremely sparse rewards. The first task we consider is **object arrangement**. We sample a random set of statements that can be simultaneously satisfied and, at every time step, the agent receives a reward of -10.0 if at least 1 statement is not satisfied satisfied and 0.0 only if all statements are satisfied. At the beginning of every episode, we reset the environment until none of the statements is satisfied. The second task is **object ordering**. An example of such a task is *"arrange the objects so that their colors range from red to blue in the horizontal direction, and keep the objects close vertically"*. In this case, the configuration can be specified with 4 pair-wise constraint between the objects. We reset the environment until at most 1 pair-wise constraint is satisfied involving the x-coordinate and the y-coordinate. At every time step, the agent receives a reward of -10.0 if at least 1 statement is not satisfied satisfied and 0.0 only if all statements are satisfied. The third task is **object sorting**. In this task, the agent needs to sort 4 object around a central object; further, the 4 objects cannot be too far away from the central object. Once again, the agent receives a reward of -10.0 if at least 1 statement is not satisfied satisfied and 0.0 only if all statements are satisfied and environment is reset until at most 1 constraint is satisfied. Images of end goal for each high-level tasks are show in the top row of Figure 3.

A.1.3 LANGUAGE ORACLE

In this work, each language statement generated by the environment is associated with a *functional program* that can be executed on the environment's relation graph to yield an answer that reflects the value of that statement on the current scene. The functional programs are built from simple elementary operation such as querying the property of objects in the scene, but they can represent a wide range of statements of different nature and can be efficiently executed on the relation graph. This scheme for generating language statements is reminiscent of the CLEVR dataset Johnson et al.

(2017) whose code we drew on and modified for our use case. Note that a language statement that can be evaluated is equivalent to a *question*, and the instructions we use also take the form of questions. For simplicity and computational efficiency, we use a smaller subset of question family defined in CLEVR that only involves pair-wise relationships between the objects. We plan to scale up to full and beyond CLEVR scale in future works. We omit the full implementation detail of the functional program which can be found in the CLEVR paper.

While evaluating the statements is relatively cheap, generating these statements can be expensive because this process involves performing graph searches. Running such operation at every time step incurs a non-trivial overhead. One solution to this problem is to keep a copy of all possible statements. This approach works well if the variation of the objects is small (e.g. 5 spheres with 5 different colors but the same shape, size, and material), but it does not scale to more general case where all properties of the objects can change. In fact, it scales exponentially with the number of possible properties.

Observe that while the number of total questions is large, the majority of the questions will always evaluate to False on the given world state. For example, a statement that contains a *large blue metallic cube* will never be True on a scene that does not contain such object. In other words, these statements are not *viable*. Therefore, a sensible alternative would be to keep a fixed set of objects and their corresponding viable questions between every episode of training, and the reset only randomly perturbs the coordinates of the objects; at the beginning of each episode, with probability $p_{resample}$ we sample a new set of objects and their corresponding viable questions. While the search for viable questions is expensive, at every interaction the number of statements that need to be checked is greatly reduced. In practice, this technique significantly speeds up the simulation.

A.2 Model implementation details

Algorithm 2 Training high-level policy

1: **Inputs:** Any RL algorithm $\mathcal{A}$; reward function $R : \mathbb{S} \to [r_{\min}, r_{\max}]^*$; instruction set $\mathbb{I}$; instruction encoder ϕ; low-level policy $\pi_l(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{g})$
2: **Initialize** $\mathcal{A}$
3: **for** episode $i = 1$ **to** M **do**
4: $\quad \boldsymbol{s}_0 \leftarrow$ reset $\mathcal{E}$
5: $\quad$ **for** step $t = 0$ **to** T **do**
6: $\quad\quad \boldsymbol{g} \leftarrow$ Sample from $\mathbb{I}$ using $\pi_h(\boldsymbol{g}|\boldsymbol{s}_t)$
7: $\quad\quad \boldsymbol{s}' \leftarrow \boldsymbol{s}_t$
8: $\quad\quad$ **for** substep $t' = 1$ **to** T' **do**
9: $\quad\quad\quad \boldsymbol{a}' \leftarrow$ Sample from $\pi_l(\boldsymbol{a}|\boldsymbol{s}', \boldsymbol{g})$
10: $\quad\quad\quad \boldsymbol{s}' \leftarrow$ Take action $\boldsymbol{a}'$ from $\boldsymbol{s}'$
11: $\quad\quad$ **end for**
12: $\quad\quad \boldsymbol{s}_{t+1} \leftarrow \boldsymbol{s}'$
13: $\quad\quad$ Store experience
14: $\quad$ **end for**
15: $\quad$ Update $\mathcal{A}$ accordingly with experience collected
16: **end for**
17: *Here we assume the reward is only based on the new state for simplicity

Low-level policy. To handle a variable number of relations between the different objects, and their changing properties, we built a goal-conditioned self attention policy network. Given a set of k object $\{\boldsymbol{o}_i\}_{i=1}^k$, we first create pair-wise concatenation of the objects, $\mathbb{O} = \{\boldsymbol{o}_i \| \boldsymbol{o}_j\}_{j=1,i=1}^{k,k}$. Then we transform every pair-wise vectors with a single neural network f_1 into $\mathbb{D} = \{f_1(\boldsymbol{o}_i \| \boldsymbol{o}_j)\}_{j=1,i=1}^{k,k}$. A recurrent neural network with GRU (Cho et al., 2014), f_2, embeds the instruction $\boldsymbol{g}$ into a real valued vector $\widetilde{\boldsymbol{g}} = f_2(\boldsymbol{g})$. We use the embedding to attend over every pair of object to compute weights $\{w_i = \langle \widetilde{\boldsymbol{g}}, \boldsymbol{d}_i \rangle | \boldsymbol{d}_i \in \mathbb{D}\}$. We then compute a weighted combination of all p_i where the weights are equal to the softmax weights $\exp(w_i)/\sum_{j=1}^{k\times k}\exp(w_j)$. This combination transforms the elements of $\mathbb{D}$ are combined into a single vector $\bar{\boldsymbol{d}}$ of fixed size. Each $\boldsymbol{o}_i$ is concatenated with $\widetilde{\boldsymbol{d}}$ and $\bar{\boldsymbol{d}}$ into $\boldsymbol{o}'_i = (\boldsymbol{o}_i \| \widetilde{\boldsymbol{g}} \| \bar{\boldsymbol{d}})$. Then each $\boldsymbol{o}'_i$ is transformed with the another neural network f_3 whose

output is of dimension d_α. The final output $\mathbb{Q} = \{f_3(\boldsymbol{o}_i \| \widetilde{\boldsymbol{g}} \| \bar{\boldsymbol{d}})\}_{i=1}^{k}$ is in $\mathbb{R}^{k \times d_\alpha}$ which represents all state-action values of the state.

This policy network is trained with HIR in a similar way to Schaul et al. (2015). This training procedure can be seen as a goal conditioned version of DQN. Training minibatches are uniformly sampled from the replay buffer. Each episode lasts for 100 steps. When the current instruction is accomplished, a new one that is currently not satisfied will be sampled. To accelerate the initial training and increase the diversity of instructions, we put a 10 step time limit on each instruction, so the policy does not get stuck if it is unable to finish the current instruction.

High-level policy. (2) For simplicity, we use a vanilla DQN with double Q-learning (van Hasselt et al., 2015) to train the high-level policy. We uses an instruction set that consists of 80 instructions ($|\mathbb{I}| = 80$) that can sufficiently cover all relationships between the objects. We roll out the low-level policy for 3 steps for every high-level instruction ($T' = 3$). Training mini batches are uniformly sampled from the replay buffer. One single set of hyperparameter is used for all experiments.

A.3 Relabeling Strategy

HER (Andrychowicz et al., 2017) demonstrated that the relabeling strategy for trajectories can have significant impacts on the performance of the policy. The most successful relabeling strategy is the "k-future" strategy where the goal state and the reward are relabeled with k states in the trajectories that are reached *after* the current time step and the reward is discounted based on the discount factor γ and how far away the current state is from the future state in ℓ_2 distance. We modify this strategy for relabeling a language conditioned policy. One challenge with language instruction is that the notion of distance is not well defined as the instruction is under-determined and only captures a part of the information about the actual state. As such, conventional metrics for describing distance between sequences of tokens (e.g. edit distance) do not actually capture the information we are interested in. Instead, we adopt a more "greedy" approach to relabeling by putting more focus on 1-step transition where the instruction is actually fulfilled. Namely, we store all transition tuples in $\mathbb{V}_t$ to the replay buffer $\mathbb{B}$ (Algorithm 1). For future relabeling, we simply use the reward discounted by time steps into the future to relabel the trajectory. While the discounted reward does not usually capture the "optimal" or true discounted reward, we found it to provide sufficient learning signal. Detailed steps are shown below (Algorithm 3).

Algorithm 3 Future Instruction Relabeling Strategy ($\mathbb{S}$)

1: **Inputs:** Trajectory τ; current time step t; number of relabeled future K
2: $\Delta \leftarrow [\,]$
3: $\texttt{count} \leftarrow 0$
4: **while** $\texttt{count} < K$ **do**
5:     $\texttt{future} \sim \text{Unif}(\{t+1, \ldots, |\tau|\})$
6:     $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{g}, r, \boldsymbol{s}', \boldsymbol{a}', \mathbb{V}) \leftarrow \tau[\texttt{future}]$
7:     **if** $|\mathbb{V}| > 0$ **then**
8:         $\boldsymbol{g}' \sim \text{Unif}(\mathbb{V})$
9:         $r' \leftarrow r \cdot \gamma^{\texttt{future}-t}$
10:         Store $(\boldsymbol{g}', r')$ in Δ
11:         $\texttt{count} \leftarrow \texttt{count} + 1$
12:     **end if**
13: **end while**
14: **return** Δ