

Language as an Abstraction for Hierarchical Deep Reinforcement Learning

Yiding Jiang*, Shixiang Gu, Kevin Murphy, Chelsea Finn
Google Research
{ydjiang, shanegu, kpmurphy, chelseaf}@google.com

Abstract

Solving complex, temporally-extended tasks is a long-standing problem in reinforcement learning (RL). We hypothesize that one critical element of solving such problems is the notion of *compositionality*. With the ability to learn concepts and sub-skills that can be composed to solve longer tasks, i.e. hierarchical RL, we can acquire temporally-extended behaviors. **However, acquiring effective yet general abstractions for hierarchical RL is remarkably challenging.** In this paper, we propose to use language as the abstraction, as it provides unique compositional structure, enabling fast learning and combinatorial generalization, while retaining tremendous flexibility, making it suitable for a variety of problems. Our approach learns an instruction-following low-level policy and a high-level policy that can reuse abstractions across tasks, in essence, permitting agents to reason using structured language. To study compositional task learning, we introduce an open-source object interaction environment built using the MuJoCo physics engine and the CLEVR engine. We find that, using our approach, agents can learn to solve to diverse, temporally-extended tasks such as object sorting and multi-object rearrangement, including from raw pixel observations. Our analysis reveals that the compositional nature of language is critical for learning diverse sub-skills and systematically generalizing to new sub-skills in comparison to non-compositional abstractions that use the same supervision.²

1 Introduction

Deep reinforcement learning offers a promising framework for enabling agents to autonomously acquire complex skills, and has demonstrated impressive performance on continuous control problems [35, 56] and games such as Atari [41] and Go [59]. However, the ability to learn a variety of compositional, long-horizon skills while generalizing to novel concepts remain an open challenge. Long-horizon tasks demand sophisticated exploration strategies and structured reasoning, while generalization requires suitable representations. In this work, we consider the question: how can we leverage the compositional structure of *language* for enabling agents to perform long-horizon tasks and systematically generalize to new goals?

To do so, we build upon the framework of hierarchical reinforcement learning (HRL), which offers a potential solution for learning long-horizon tasks by training a hierarchy of policies. However, the abstraction between these policies is critical for good performance. Hard-coded abstractions often lack modeling flexibility and are task-specific [63, 33, 26, 47], while learned abstractions often find degenerate solutions without careful tuning [5, 24]. One possible solution is to have the higher-level policy generate a sub-goal state and have the low-level policy try to reach that goal state [42, 36]. However, using goal states still lacks some degree of flexibility (e.g. in comparison

*Work done as a part of the Google AI Residency program

²Code and videos of the environment, and experiments are at <https://sites.google.com/view/hal-demo>



Figure 1: The environment and some instructions that we consider in this work, along with an illustration of hindsight instruction relabeling (HIR), which we use to enable the agent to learn from many different language goals at once (Details in Section 4.2).

to goal regions or attributes), is challenging to scale to visual observations naively, and does not generalize systematically to new goals. In contrast to these prior approaches, language is a flexible representation for transferring a variety of ideas and intentions with minimal assumptions about the problem setting; its compositional nature makes it a powerful abstraction for representing combinatorial concepts and for transferring knowledge [22].

In this work, we propose to use language as the interface between high- and low-level policies in hierarchical RL. With a low-level policy that follows language instructions (Figure 1), the high-level policy can produce actions in the space of language, yielding 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 actions correspond to language instructions, making it easier to recognize and diagnose failures. Third, language abstractions can be viewed as a strict generalization of goal states, as an instruction can represent a *region of states* that satisfy some abstract criteria, rather than the entirety of an individual goal state. Finally, studies have also suggested that humans use language as an abstraction for reasoning and planning [21, 49]. In fact, the majority of knowledge learning and skill acquisition we do throughout our life is through languages.

While language is an appealing choice as the abstraction for hierarchical RL, training a low-level policy to follow language instructions is highly non-trivial [20, 6] as it involves learning from binary rewards that indicate completion of the instruction. To address this problem, we generalize prior work on goal relabeling to the space of language instructions (which instead operate on regions of state space, rather than a single state), allowing the agent to learn from many language instructions at once.

To empirically study the role of language abstractions for long-horizon tasks, we introduce a new environment inspired by the CLEVR engine [28] that consists of procedurally-generated scenes of objects that are paired with 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. We find that our approach is able to learn a variety of vision-based long-horizon manipulation tasks such as object reconfiguration and sorting, while outperforming state-of-the-art RL and hierarchical RL approaches. Further, our experimental analysis finds that HRL with non-compositional abstractions struggles to learn the tasks, even when the non-compositional abstraction is derived from language instructions themselves, demonstrating the critical role of compositionality in learning. Lastly, we find that our instruction-following agent is able to generalize to instructions that are systematically different from those seen during training.

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

1. a framework for using language abstractions in HRL, with which we find that the structure and flexibility of language enables agents to solve challenging long-horizon control problems
2. an open-source continuous control environment for studying compositional, long-horizon tasks, integrated with language instructions inspired by the CLEVR engine [28]
3. empirical analysis that studies the role of compositionality in learning long-horizon tasks and achieving systematic generalization

2 Related Work

Designing, discovering and learning meaningful and effective abstractions of MDPs has been studied extensively in hierarchical reinforcement learning (HRL) [15, 46, 63, 16, 5]. Classically, the work

on HRL has focused on learning only the high-level policy given a set of hand-engineered low-level policies [60, 38, 9], or more generically *options* policies with flexible termination conditions [63, 52].

Recent HRL works have begun to tackle more difficult control domains with both large state spaces and long planning horizons [26, 34, 65, 17, 42, 43]. These works can typically be categorized into two approaches. The first aims to learn effective low-level policies end-to-end directly from final task rewards with minimal human engineering, such as through the option-critic architecture [5, 24] or multi-task or meta learning [18, 58]. While appealing in theory, this end-to-end approach relies solely on final task rewards and is shown to scale poorly to complex domains [5, 42], unless distributions of tasks are carefully designed [18]. The second approach instead augments the low-level learning with auxiliary rewards that can bring better inductive bias. These rewards include mutual information-based diversity rewards [13, 17], hand-crafted rewards based on domain knowledge [33, 26, 34, 65], and goal-oriented rewards [15, 55, 4, 69, 42, 43]. Goal-oriented rewards have been shown to balance sufficient inductive bias for effective learning with minimal domain-specific engineering, and achieve performance gains on a range of domains [69, 42, 43]. Our work is a generalization on these lines of work, representing goal *regions* using language instructions, rather than individual goal *states*. Here, *region* refers to the sets of states (possibly disjoint and far away from each other) that satisfy more abstract criteria (e.g. “red ball in front of blue cube” can be satisfied by infinitely many states that are drastically different from each other in the pixel space) rather than a simple ϵ -ball around a single goal state that is only there for creating a reachable non-zero measure goal. Further, our experiments demonstrate significant empirical gains over these prior approaches.

Since our low-level policy training is related to goal-conditioned HRL, we can benefit from algorithmic advances in multi-goal reinforcement learning [29, 55, 4, 50]. In particular, we extend the recently popularized goal relabeling strategy [29, 4] to instructions, allowing us to relabel based on achieving a language statement that describes a *region* of state space, rather than relabeling based on reaching an individual state.

Lastly, there are a number of prior works that study how language can guide or improve reinforcement learning [37, 19, 30, 6, 20, 12, 10]. While prior work has made use of language-based sub-goal policies in hierarchical RL [57, 14], the instruction representations used lack the needed diversity to benefit from the compositionality of language over one-hot goal representations. In a concurrent work, Wu et al. [70] show that language can help with learning difficult tasks where more naive goal representations lead to poor performance, even with hindsight goal relabeling. While we are also interested in using language to improve learning of challenging tasks, we focus on the use of language in the context of hierarchical RL, demonstrating that language can be further used to compose complex objectives for the agent. Andreas et al. [3] leverage language descriptions to rapidly adapt to unseen environments through structured policy search in the space of language; each environment is described by one sentence. In contrast, we show that a high-level policy can effectively leverage the combinatorially many sub-policies induced by language by generating a sequence of instructions for the low-level agent. Further, we use language not only for adaptation but also for learning the lower level control primitives, without the need for imitation learning from an expert. Another line of work focuses on RL for textual adventure games where the state is represented as language descriptions and the actions are either textual actions available at each state [25] or all possible actions [45] (even though not every action is applicable to all states). In general, these works look at text-based games with discrete 1-bit actions, while we consider continuous actions in physics-based environments. One may view the latter as a high-level policy with oracular low-level policies that are specific to each state; the discrete nature of these games entails limited complexity of interactions with the environment.

3 Preliminaries

Standard reinforcement learning. The typical RL problem considers a *Markov decision process* (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where \mathcal{S} is the state space, \mathcal{A} is the action space, the unknown transition probability $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, \infty)$ represents the probability density of reaching $s_{t+1} \in \mathcal{S}$ from $s_t \in \mathcal{S}$ by taking the action $a \in \mathcal{A}$, $\gamma \in [0, 1]$ is the discount factor, and the bounded real-valued function $R : \mathcal{S} \times \mathcal{A} \rightarrow [r_{\min}, r_{\max}]$ represents the reward of each transition. We further denote $\rho_\pi(s_t)$ and $\rho_\pi(s_t, a_t)$ as the state marginal and the state-action marginal of the trajectory induced by policy $\pi(a_t|s_t)$. The objective of reinforcement learning is to find a policy $\pi(a_t|s_t)$ such that the expected discounted future reward $\sum_t \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} [\gamma^t R(s_t, a_t)]$ is maximized.

Goal conditioned reinforcement learning. In goal-conditioned RL, we work with an *Augmented Markov Decision Process*, which is defined by the tuple $(\mathcal{S}, \mathcal{G}, \mathcal{A}, T, R, \gamma)$. Most elements represent the same quantities as a standard MDP. The additional tuple element \mathcal{G} is the space of all possible goals, and the reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \rightarrow [r_{\min}, r_{\max}]$ represents the reward of each transition *under a given goal*. Similarly, the policy $\pi(\mathbf{a}_t | \mathbf{s}_t, \mathbf{g})$ is now conditioned on \mathbf{g} . Finally, $p_g(\mathbf{g})$ represents a distribution over \mathcal{G} . The objective of goal directed reinforcement learning is to find a policy $\pi(\mathbf{a}_t | \mathbf{s}_t, \mathbf{g})$ such that the expected discounted future reward $\sum_t \mathbb{E}_{\mathbf{g} \sim p_g, (\mathbf{s}_t, \mathbf{a}_t) \sim \rho_\pi} [\gamma^t R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{g})]$ is maximized. While this objective can be expressed with a standard MDP by augmenting the state vector with a goal vector, the policy does not change the goal; the explicit distinction between goal and state facilitates discussion later.

Q-learning. Q-learning is a large class of off-policy reinforcement learning algorithms that focuses on learning the Q-function, $Q^*(\mathbf{s}_t, \mathbf{a}_t)$, which represents the expected total discounted reward that can be obtained after taking action \mathbf{a}_t in state \mathbf{s}_t assuming the agent acts optimally thereafter. It can be recursively defined as:

$$Q^*(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1}} [R(\mathbf{s}_t, \mathbf{a}_t) + \gamma \max_{\mathbf{a} \in \mathcal{A}} (Q^*(\mathbf{s}_{t+1}, \mathbf{a}))] \quad (1)$$

The optimal policy learned can be recovered through $\pi^*(\mathbf{a}_t | \mathbf{s}_t) = \delta(\mathbf{a}_t = \arg \max_{\mathbf{a} \in \mathcal{A}} Q^*(\mathbf{s}_t, \mathbf{a}))$. In high-dimensional spaces, the Q-function is usually represented with function approximators and fit using transition tuples, $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_t)$, which are stored in a *replay buffer* [41].

Hindsight experience replay (HER). HER [4] is a data augmentation technique for off-policy goal conditioned reinforcement learning. For simplicity, assume that the goal is specified in the state space directly. A trajectory can be transformed into a sequence of goal augmented transition tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_g, \mathbf{s}_{t+1}, r_t)$. We can relabel each tuple’s \mathbf{s}_g with \mathbf{s}_{t+1} or other future states visited in the trajectory and adjust r_t to be the appropriate value. This makes the otherwise sparse reward signal much denser. This technique can also be seen as generating an implicit curriculum of increasing difficulty for the agent as it learns to interact with the environment more effectively.

4 Hierarchical Reinforcement Learning with Language Abstractions

In this section, we present our framework for training a 2-layer hierarchical policy with compositional *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, including our assumptions regarding the availability of supervision. We will then discuss how we can efficiently train the low-level policy, $\pi_l(\mathbf{a} | \mathbf{s}_t, \mathbf{g})$ conditioned on language instructions \mathbf{g} in Section 4.2, and how a high-level policy, $\pi_h(\mathbf{g} | \mathbf{s}_t)$, can be trained using such a low-level policy in Section 4.3. We refer to this framework as *Hierarchical Abstraction with Language* (HAL, Figure 2, Appendix C.1).

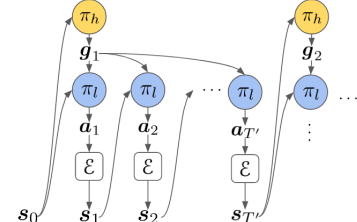


Figure 2: HAL: The high-level policy π_h produces language instructions \mathbf{g} for the low level policy π_l .

4.1 Problem statement

We are interested in learning temporally-extended tasks by leveraging the compositionality of language. Thus, in addition to the standard reinforcement learning assumptions laid out in Section 3, we also need some form of grounded language supervision in the environment \mathcal{E} during training. To this end, we also assume the access to a conditional density $\omega(\mathbf{g} | \mathbf{s})$ that maps observation \mathbf{s} to a distribution of language statements $\mathbf{g} \in \mathcal{G}$ that describes \mathbf{s} . This distribution can take the form of a supervised image captioning model, a human supervisor, or a functional program that is executed on \mathbf{s}_t similar to CLEVR. Further, we define $\Omega(\mathbf{s}_t)$ to be the support of $\omega(\mathbf{g} | \mathbf{s}_t)$. Moreover, we assume the access to a function Ψ that maps a state and an instruction to a single Boolean value which indicates whether the instruction is satisfied or not by the \mathbf{s} , i.e. $\Psi : \mathcal{S} \times \mathcal{G} \rightarrow \{0, 1\}$. Once again, Ψ can be a VQA model, a human supervisor or a program. Note that any goal specified in the state space can be easily expressed by a Boolean function of this form by checking if two states are close to each other up to some threshold parameter. Ψ can effectively act as the reward for the low-level policy.

An example for the high-level tasks is arranging objects in the scene according to a specific spatial relationship. This can be putting the object in specific arrangement or ordering the object according

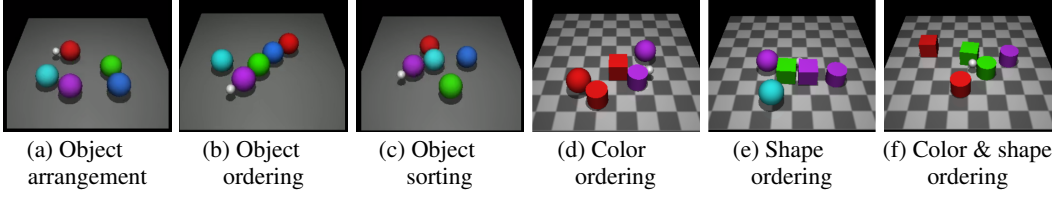


Figure 3: Sample goal states for the high-level tasks in the standard (a-c) and diverse (d-f) environments. The high-level policy only receives reward if all constraints are satisfied. The global location of the objects may vary.

the the colors (Figure 3) by pushing the objects around (Figure 1). Details of these high-level tasks are described in Section 5. These tasks are complex but can be naturally decomposed into smaller sub-tasks, giving rise to a naturally defined hierarchy, making it an ideal testbed for HRL algorithms. Problems of similar nature including organizing a cluttered table top or stacking LEGO blocks to construct structures such as a castle or a bridge. We train the low-level policy $\pi_l(a|s, g)$ to solve an augmented MDP described Section 3. For simplicity, we assume that Ω 's output is uniform over \mathcal{G} . The low-level policy receives supervision from Ω and Ψ by completing instructions. The high-level policy $\pi_h(g|s)$ is trained to solve a standard MDP whose state space is the same \mathcal{S} as the low-level policy, and the action space is \mathcal{G} . In this case, the high-level policy's supervision comes from the reward function of the environment which may be highly sparse.

We separately train the high-level policy and low-level policy, so the low-level policy is agnostic to the high-level policy. Since the policies share the same \mathcal{G} , the low-level policy can be reused for different high-level policies (Appendix C.3). Jointly fine-tuning the low-level policy with a specific high-level policy is certainly a possible direction for future work (Appendix C.1).

4.2 Training a language-conditioned low-level policy

To train a goal conditioned low-level policy, we need to define a suitable reward function for training such a policy and a mechanism for sampling language instructions. A straightforward way to represent the reward for the low-level policy would be $R(s_t, a_t, s_{t+1}, g) = \Psi(s_{t+1}, g)$ or, to ensure that a_t is inducing the reward:

$$R(s_t, a_t, s_{t+1}, g) = \begin{cases} 0 & \text{if } \Psi(s_{t+1}, g) = 0 \\ \Psi(s_{t+1}, g) \oplus \Psi(s_t, g) & \text{if } \Psi(s_{t+1}, g) = 1 \end{cases}$$

However, optimizing with this reward directly is difficult because the reward signal is only non-zero when the goal is achieved. Unlike prior work (e.g. HER [4]), which uses a state vector or a task-relevant part of the state vector as the goal, it is difficult to define meaningful distance metrics in the space of language statements [8, 53, 61], and, consequently, difficult to make the reward signal smooth by assigning partial credit (unlike, e.g., the ℓ_p norm of the difference between 2 states). To overcome these difficulties, we propose a trajectory relabeling technique for language instructions: Instead of relabeling the trajectory with states reached, we relabel states in the trajectory τ with the elements of $\Omega(s_t)$ as the goal instruction using a relabeling strategy \mathcal{S} . We refer to this procedure as *hindsight instruction relabeling* (HIR). The details of \mathcal{S} is located in Algorithm 4 in Appendix C.4. Pseudocode for the method can be found in Algorithm 2 in Appendix C.2 and an illustration of the process can be found in Figure 1.

The proposed relabeling scheme, HIR, is reminiscent of HER [4]. In HER, the goal is often the state or a simple function of the state, such as a masked representation. However, with high-dimensional observation spaces such as images, there is excessive information in the state that is irrelevant to the goal, while task-relevant information is not readily accessible. While one can use HER with generative models of images [54, 51, 44], the resulting representation of the state may not effectively capture the relevant aspects of the desired task. Language can be viewed as an alternative, highly-compressed representation of the state that explicitly exposes the task structure, e.g. the relation between objects. Thus, we can readily apply HIR to settings with image observations.

4.3 Acting in language with the high-level policy

We aim to learn a high-level policy for long-horizon tasks that can explore and act in the space of language by providing instructions g to the low-level policy $\pi_l(a|s_t, g)$. The use of language abstractions through g allows the high-level policy to structurally explore with actions that are semantically meaningful and span multiple low-level actions.

In principle, the high-level policy, $\pi_h(\mathbf{g}|\mathbf{s})$, can be trained with any reinforcement learning algorithm, given a suitable way to generate sentences for the goals. However, generating coherent sequences of discrete tokens is difficult, particularly when combined with existing reinforcement learning algorithms. We explore how we might incorporate a language model into the high level policy in Appendix A, which shows promising preliminary results but also significant challenges. Fortunately, while the size of the instruction space \mathcal{G} scales exponentially with the size of the vocabulary, the elements of \mathcal{G} are naturally structured and 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, in many cases, the high-level policy only needs to generate instruction from a much smaller subset of \mathcal{G} to direct the low-level policy. We denote such subsets of \mathcal{G} as \mathcal{I} .

If \mathcal{I} is relatively small, the problem can be recast as a discrete-action RL problem, where one action choice corresponds to an instruction, and can be solved with algorithms such as DQN [41]. We adopt this simple approach in this work. 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. We found that simply using a fixed T' was sufficient in our experiments.

5 The Environment and Implementation

Environment. To empirically study how compositional languages can aid in long-horizon reasoning and generalization, we need an environment that will test the agent’s ability to do so. While prior works have studied the use of language in navigation [1], instruction following in a grid-world [10], and compositionality in question-answering, we aim to develop a *physical* simulation environment where the agent must interact with and change the environment in order to accomplish long-horizon, compositional tasks. These criteria are particularly appealing for robotic applications, and, to the best of our knowledge, none of the existing benchmarks simultaneously fulfills all of them. To this end, we developed a new environment using the MuJoCo physics engine [66] and the CLEVR language engine, that tests an agent’s ability to manipulate and rearrange objects of various shapes and colors. To succeed in this environment, the agent must be able to handle varying number of objects with diverse visual and physical properties. Two versions of the environment of varying complexity are illustrated in Figures 3 and 1 and further details are in Appendix B.1.

High-level tasks. We evaluate our framework on 6 challenging temporally-extended tasks across two environments, all illustrated in Figure 3: (a) *object arrangement*: manipulate objects such that 10 pair-wise constraints are satisfied, (b) *object ordering*: order objects by color, (c) *object sorting*: arrange 4 objects around a central object, and in a more diverse environment (d) *color ordering*: order objects by color irrespective of shape, (e) *shape ordering*: order objects by shape irrespective of color, and (f) *color & shape ordering*: order objects by both shape and color. In all cases, the agent receives a binary reward only if all constraints are satisfied. Consequently, this makes obtaining meaningful signal in these tasks extremely challenging as only a very small number of action sequences will yield non-zero signal. For more details, see Appendix B.2.

Action and observation parameterization. The state-based observation is $\mathbf{s} \in \mathbb{R}^{10}$ that represents the location of each object and $|\mathcal{A}| = 40$ which corresponds to picking an object and pushing it in one of the eight cardinal directions. The image-based observation is $\mathbf{s} \in \mathbb{R}^{64 \times 64 \times 3}$ which is the rendering of the scene and $|\mathcal{A}| = 800$ which corresponds to picking a location in a 10×10 grid and pushing in one of the eight cardinal directions. For more details, see Appendix B.1.

Policy parameterization. The low-level policy encodes the instruction with a GRU and feeds the result, along with the state, into a neural network that predicts the Q-value of each action. The high-level policy is also a neural network Q-function. Both use Double DQN [68] for training. The high-level policy uses sets of 80 and 240 instructions as the action space in the standard and diverse environments respectively, a set that sufficiently covers relationships between objects. We roll out the low-level policy for $T' = 5$ steps for every high-level instruction. For details, see Appendix B.3.

6 Experiments

To evaluate our framework, and study the role of compositionality in RL in general, we design the experiments to answer the following questions: (1) as a representation, how does language compare

to alternative representations, such as those that are not explicitly compositional? (2) How well does the framework scale with the diversity of instruction and dimensionality of the state (e.g. vision-based observation)? (3) Can the policy generalize in systematic ways by leveraging the structure of language? (4) Overall, how does our approach compare to state-of-the-art hierarchical RL approaches, along with learning flat, homogeneous policies?

To answer these questions, in Section 6.1, we first evaluate and analyze training of effective low-level policies, which are critical for effective learning of long-horizon tasks. Then, in Section 6.2, we evaluate the full HAL method on challenging temporally-extended tasks. Finally, we apply our method to the Crafting environment from Andreas et al. [2] to showcase the generality of our framework. For details on the experimental set-up and analysis, see Appendix D and E.

6.1 Low-level Policy

Role of compositionality and relabeling. We start by evaluating the fidelity of the low-level instruction-following policy, in isolation, with a variety of representations for the instruction. For these experiments, we use state-based observations. We start with a set of 600 instructions, which we paraphrase and substitute synonyms to obtain more than 10,000 total instructions which allows us to answer the first part of (2). We evaluate the performance all low-level policies by the average number of instructions it can successfully achieve each episode (100 steps), measured over 100 episodes. To answer (1), and evaluate the importance of compositionality, we compare against:

- a **one-hot encoded representation** of instructions where each instruction has its own row in a real-valued embedding matrix which uses the same instruction relabeling (see Appendix D.1)
- a **non-compositional latent variable representation** with identical information content. We train an sequence auto-encoder on sentences, which achieves 0 reconstruction error and is hence a *lossless* non-compositional representation of the instructions (see Appendix D.2)
- a **bag-of-words (BOW) representation** of instructions (see Appendix D.3)

In the first comparison, we observe that while one-hot encoded representation works on-par with or better than the language in the regime where the number of instructions is small, its performance quickly deteriorates as the number of instruction increases (Fig.4, middle). On the other hand, language representation of instruction can leverage the structures shared by different instructions and does not suffer from increasing number of instructions (Fig.4, right blue); in fact, an *improvement* in performance is observed. This suggests, perhaps unsurprisingly, that one-hot representations and state-based relabeling scale poorly to large numbers of instructions, *even when the underlying number of instructions does not change*, while, with instruction relabeling (HIR), the policy acquires better, more successful representations as the number of instructions increases.

In the second comparison, we observe that the agent is unable to make meaningful progress with this representation despite receiving identical amount of supervision as language. This indicates that the compositionality of language is critical for effective learning. Finally, we find that relabeling is critical for good performance, since without it (no HIR), the reward signal is significantly sparser.

Finally, in the comparison to a bag-of-words representation (BOW), we observe that, while the BOW agent’s return increases at a faster rate than that of the language agent at the beginning of training – likely due to the difficulty of optimizing recurrent neural network in language agent – the language agent achieves significantly better final performance. On the other hand, the performance of the BOW agent plateaus at around 8 instructions per episode. This is expected as BOW does not consider the *sequential nature* of an instruction, which is important for correctly executing an instruction.

Vision-based observations. To answer the second part of (2), we extend our framework to pixel observations. The agent reaches the same performance as the state-based model, albeit requiring longer convergence time with the same hyper-parameters. On the other hand, the one-hot representation reaches much worse relative performance with the same amount of experience (Fig.4, right).

Visual generalization. One of the most appealing aspects of language is the promise of *combinatorial generalization* [7] which allows for *extrapolation* rather than simple *interpolation* over the training set. To evaluate this (i.e. (3)), we design the training and test instruction sets that are systematically distinct. We evaluate the agent’s ability to perform such generalization by splitting the 600 instruction sets through the following procedure: (i) **standard**: random 70/30 split of the instruction set; (ii) **systematic**: the training set only consists of instructions that do not contain the words *red* in the first half of the instructions and the test set contains only those that have *red* in the first half of the

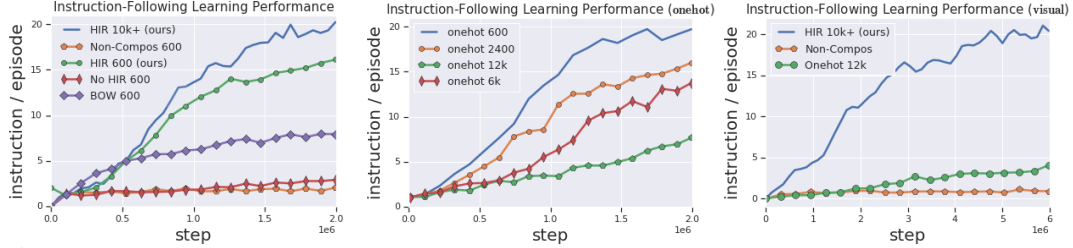


Figure 4: Results for low-level policies in terms of goals accomplished per episode over training steps for HIR. **Left:** HIR with different number of instructions and results with non-compositional representation and with no relabeling. **Middle:** Results for one-hot encoded representation with increasing number of instructions. 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. **Right:** Performance of image-based low-level policy compared against one-hot and non-compositional instruction representations.

instructions. We emphasize that the agent has *never* seen the words red in the first part of the sentence in training; in other words, the task is *zero-shot* as the training set and the test set are disjoint (i.e. the distributions do not share support). From a pure statistical learning theoretical perspective, the agent should not do better than chance on such a test set. Remarkably, we observe that the agent generalizes better with language than with non-compositional representation (table 1). This suggests that the agent recognizes the compositional structure of the language, and achieves systematic generalization through such understanding.

	Standard train	Standard test	Standard gap	Systematic train	Systematic test	Systematic gap
Language	21.50 ± 2.28	21.49 ± 2.53	0.001	20.09 ± 2.46	8.13 ± 2.34	0.596
Non-Compositional	6.26 ± 1.18	5.78 ± 1.44	0.077	7.54 ± 1.14	0.76 ± 0.69	0.899
Random	0.17 ± 0.20	0.21 ± 0.17	-	0.11 ± 0.19	0.18 ± 0.22	-

Table 1: Final performance of the low-level policy on different training and test instruction distributions (20 episodes). Language outperforms the non-compositional language representation in both absolute performance and relative generalization gap for every setting. *Gap* is equal to one minus the ratio of between mean test performance and mean train performance; this quantity can be interpreted as the generalization gap. For instructions with language representation, the generalization gap increases by approximately 59.5% from standard generalization to zero-shot generalization while for non-compositional representation the generalization gap increases by 82.2%

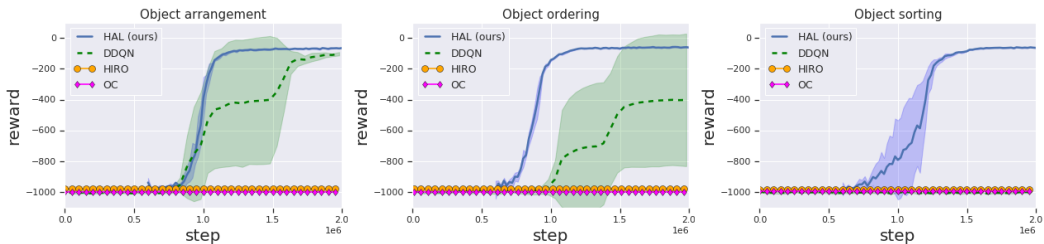


Figure 5: Results for high-level policy on tasks (a-c). Blue curves for HAL include the steps for training the low-level policy (a *single* low-level policy is used for all 3 tasks). In all settings, HAL demonstrates faster learning than DDQN. Means and standard deviations of 3 random seeds are plotted.

6.2 High-level policy

Now that we have analyzed the low-level policy performance, we next evaluate the full HAL algorithm. To answer (4), we compare our framework in the state space against a non-hierarchical baseline DDQN and two representative hierarchical reinforcement learning frameworks HIRO [42] and Option-Critic (OC) [5] on the proposed high-level tasks with sparse rewards (Sec.5). We observe that neither HRL baselines are able to learn a reasonable policy while DDQN is able to solve only 2 of the 3 tasks. HAL is able to solve all 3 tasks consistently and with much lower variance and better asymptotic performance (Fig.5). Then we show that our framework successfully transfers to high-dimensional

observation (i.e. images) in all 3 tasks without loss of performance whereas even the non-hierarchical DDQN fails to make progress (Fig.6, left). Finally, we apply the method to 3 additional diverse tasks (Fig.6, middle). In these settings, we observed that the high-level policy has difficulty learning from pixels alone, likely due to the visual diversity and the simplified high-level policy parameterization. As such, the high-level policy for diverse setting receives state observation but the low-level policy uses the raw-pixel observation. For more details, please refer to Appendix B.3.

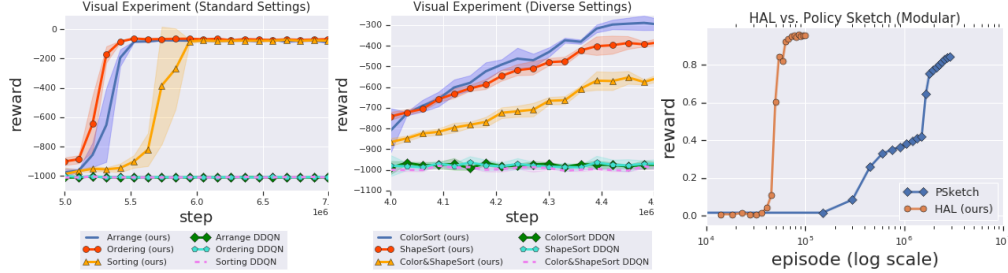


Figure 6: **Left:** Results for **vision-based** hierarchical RL. In all settings, HAL demonstrates faster and more stable learning while the baseline DDQN cannot learn a non-trivial policy. In this case, the vision-based low-level policy needs longer training time ($\sim 5 \times 10^6$ steps) so we start the x-axis there. Means and standard deviations of 3 seeds are plotted (Near 0 variance for DDQN). **Middle:** Results of HRL on the proposed 3 diverse tasks (d-e). In this case, the low-level policy used is trained on image observation for ($\sim 4 \times 10^6$ steps). 3 random seeds are plotted and training has not converged. **Right:** HAL vs policy sketches on the Crafting Environment. HAL is significantly more sample efficient since it is off-policy and uses relabeling among all modules.

6.3 Crafting Environment

To show the generality of the proposed framework, we apply our method to the **Crafting** environments introduced by Andreas et al. [2] (Fig. 6, right). We apply HAL to this environment by training *separate* policy networks for each module since there are fewer than 20 modules in the environment. These low-level policies receive binary rewards (i.e. one-bit supervision analogous to completing an instruction), and are trained jointly with HIR. Another high-level policy that picks which module to execute for a fixed 5 steps and is trained with regular DDQN. Note that our method uses a *different* form of supervision compared to policy sketch since we provide binary reward for the low-level policy – such supervision can sometimes be easier to specify than the entire sketch.

7 Discussion

We demonstrate that language abstractions can serve as an efficient, flexible, and human-interpretable representation for solving a variety of long-horizon control problems in HRL framework. Through re-labeling and the inherent compositionality of language, we show that low-level, language-conditioned policies can be trained efficiently without engineered reward shaping and with large numbers of instructions, while exhibiting strong generalizations. Our framework HAL can thereby leverage these policies to solve a range of difficult sparse-reward manipulation tasks with greater success and sample efficiency than training without language abstractions.

While our method demonstrates promising results, one limitation is that the current method relies on instructions provided by a language supervisor which has access to the instructions that describe a scene. The language supervisor can, in principle, be replaced with an image-captioning model and question-answering model such that it can be deployed on real image observations for robotic control tasks, an exciting direction for future work. Another limitation is that the instruction set used is specific to our problem domain, providing a substantial amount of pre-defined structure to the agent. It remains an open question on how to enable an agent to follow a much more diverse instruction set, that is not specific to any particular domain, or learn compositional abstractions without the supervision of language. Our experiments suggest that both would likely yield an HRL method that requires minimal domain-specific supervision, while yielding significant empirical gains over existing domain-agnostic works, indicating an promising direction for future research. Overall, we believe this work represents a step towards RL agents that can effectively reason using compositional language to perform complex tasks, and hope that our empirical analysis will inspire more research in compositionality at the intersection of language and reinforcement learning.

Acknowledgments

We would like to thank anonymous reviewers for the valuable feedback. We would also like to thank Jacob Andreas, Justin Fu, Sergio Guadarrama, Ofir Nachum, Vikash Kumar, Allan Zhou, Archit Sharma, and other colleagues at Google Research for helpful discussion and feedback on the draft of this work.

References

- [1] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, 2018.
- [2] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches, 2016.
- [3] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language, 2017.
- [4] 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.
- [5] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.
- [6] Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to understand goal specifications by modelling reward, 2018.
- [7] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [8] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.
- [9] Nuttapon Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*, pages 1281–1288, 2005.
- [10] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. BabyAI: First steps towards grounded language learning with a human in the loop. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=rJeXCo0cYX>.
- [11] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches, 2014.
- [12] John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. Meta-learning language-guided policy learning. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=HkgSEnA5KQ>.
- [13] Christian Daniel, Gerhard Neumann, and Jan Peters. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*, pages 273–281, 2012.
- [14] Abhishek Das, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra. Neural modular control for embodied question answering. *arXiv preprint arXiv:1810.11181*, 2018.
- [15] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *Advances in neural information processing systems*, pages 271–278, 1993.

- [16] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.
- [17] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.
- [18] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. *International Conference on Learning Representations (ICLR)*, 2018.
- [19] Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems*, pages 3314–3325, 2018.
- [20] Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=r11q1hRqYQ>.
- [21] Lila Gleitman and Anna Papafragou. Language and thought. *Cambridge handbook of thinking and reasoning*, pages 633–661, 2005.
- [22] H Paul Grice. Logic and conversation. 1975, pages 41–58, 1975.
- [23] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.
- [24] 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.
- [25] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. *arXiv preprint arXiv:1511.04636*, 2015.
- [26] 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.
- [27] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.
- [28] 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.
- [29] Leslie Pack Kaelbling. Hierarchical learning in stochastic domains: Preliminary results. In *Proceedings of the tenth international conference on machine learning*, volume 951, pages 167–173, 1993.
- [30] Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning, 2017.
- [31] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [32] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [33] George Konidaris and Andrew G Barto. Building portable options: Skill transfer in reinforcement learning. In *IJCAI*, volume 7, pages 895–900, 2007.

- [34] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.
- [35] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- [36] Andrew Levy, Robert Platt, and Kate Saenko. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations*, 2019. URL <https://openreview.net/forum?id=ryzECoAcY7>.
- [37] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*, 2019.
- [38] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the twenty-first international conference on Machine learning*, page 71. ACM, 2004.
- [39] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.
- [40] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.
- [41] 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.
- [42] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning, 2018.
- [43] Ofir Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2019.
- [44] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *CoRR*, abs/1807.04742, 2018. URL <http://arxiv.org/abs/1807.04742>.
- [45] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. *arXiv preprint arXiv:1506.08941*, 2015.
- [46] Ronald Parr and Stuart J Russell. Reinforcement learning with hierarchies of machines. In *Advances in neural information processing systems*, pages 1043–1049, 1998.
- [47] 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.
- [48] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [49] 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.
- [50] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control, 2018.
- [51] Vitchyr H. Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019. URL <http://arxiv.org/abs/1903.03698>.

- [52] Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.
- [53] Ehud Reiter. A structured review of the validity of bleu. *Computational Linguistics*, 44(3): 393–401, 2018.
- [54] Himanshu Sahni, Toby Buckley, Pieter Abbeel, and Ilya Kuzovkin. Visual hindsight experience replay. *CoRR*, abs/1901.11529, 2019. URL <http://arxiv.org/abs/1901.11529>.
- [55] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pages 1312–1320, 2015.
- [56] 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.
- [57] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.
- [58] Olivier Sigaud and Freek Stulp. Policy search in continuous action domains: an overview. *arXiv preprint arXiv:1803.04706*, 2018.
- [59] 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.
- [60] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pages 212–223. Springer, 2002.
- [61] Elior Sulem, Omri Abend, and Ari Rappoport. Bleu is not suitable for the evaluation of text simplification. *arXiv preprint arXiv:1810.05995*, 2018.
- [62] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.
- [63] 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.
- [64] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [65] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*, volume 3, page 6, 2017.
- [66] 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*, pages 5026–5033. IEEE, 2012.
- [67] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017. URL <http://arxiv.org/abs/1711.00937>.
- [68] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.
- [69] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.
- [70] 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>.

A Language Model as the High-level Policy

Picking an instruction from a pre-defined set of instructions can limit the flexibility of the high-level policy and introduces scaling issues for more complex language instructions. In this section, we present a method for using a language model as the high-level policy, thereby removing the constraints of fixed instruction set and potentially enabling much more powerful and general-purpose high-level policy. Naively, directly using language as the action space is difficult for RL agent as language has extremely high-dimension and presents difficult exploration challenges. An insight that we can leverage is that many language statements, especially instructions, lie in a much lower dimension space. To that end, we can learn a low-dimension embedding space of the instructions and then set the action space of the high-level RL agent to be the embedding space. Note that naively training a latent variable model such as a VAE [32] is unlikely work well since the latent space does not have the structure necessary for being a good action space – an indirect example that demonstrates this fact is that the non-compositional representation for the low-level policy, which is also a latent variable model completely fails to learn a reasonable policy, even though it contains perfect information about the instruction. We hypothesize that to facilitate learning, the embedding space must be sufficiently *disentangled*. Learning a disentangled representation of language without any supervision is very much still an open problem and we present a candidate solution that can learn an well disentangled embedding space (for a relatively domain-specific language) that can efficiently be used as the action space for the high-level policy.

A.1 Learning a Disentangled Representation of Language

Unlike modeling the density of language with architectures such as GPT or Transformers, we want to learn an disentangled representation of language via an embedding space. To this end, we leverage an InfoGAN-like auto-regressive architecture that turns an embedding to a sentence, and use a mutual information based objective to encourage disentanglement. Further, since the generated tokens are discrete, we explore several method for efficiently training the generator with gradients and find that using vector quantization [67] empirically outperforms unbiased methods such as the Gumbel-softmax trick [27].

Notations:

1. The latent vector $\tilde{z} = (z, c) \in \mathbb{R}^{10}$ is a concatenation of a noise vector $z \in \mathbb{R}^4$ sampled from an isotropic Gaussian and a latent code $c \in \mathbb{R}^6$ sampled uniformly from the hypercube $[-1, 1]^6$. With a little abuse of notations, we use \tilde{z}_c to denote the latent code components of the latent vector \tilde{z} .
2. $\mathcal{V} = \{v_i\}_{i=1}^{N_V}$ is the set of all vocabularies with size N_V
3. $\mathcal{E} = \{e_i\}_{i=1}^{N_V}$ is a set embedding vector for each vocabulary where $e \in \mathbb{R}^{d_e}$. Further, for stability, each e_i is normalize to be unit vector such that $\|e_i\|_2 = 1$ (constrained to a unit sphere).
4. $\mathbf{x} = [v_1, v_2, \dots, v_m]$ is real instruction datum, assuming all instructions are of length m (padded with $\langle EOS \rangle$ if shorter).
5. \mathbf{x}_i is the i^{th} example in a collection of sampled instructions $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N_X}$.
6. $G(\tilde{z}; \theta_G)$ is the generator that maps a latent code to a vector $\mathbf{x}_{gen} \in \mathbb{R}^{m \times d_e}$. Each row of $\mathbf{x}_{gen}^{(i)}$ can be mapped to some

$$e(\mathbf{x}_{gen}^{(i)}) = \arg \min_{e_j} \|e_j - \mathbf{x}_{gen}^{(i)}\|_2$$

7. $D(\mathbf{x}; \theta_D)$ maps a sample to a scalar; D is constrained to be 1-Lipschitz.
8. $Q(\mathbf{x}_{gen}; \theta_Q)$ maps a generated sample to its latent code \tilde{c} . Q shares the parameters all layers except for the last layer with D
9. sg denotes stop gradient operator

Then the **training objective of the generator** is defined as:

$$\ell_{VQ}(\tilde{z}) = \frac{1}{m} \sum_{i=1}^m \|\text{sg}[G(\tilde{z})^{(i)}] - e(G(\tilde{z})^{(i)})\|_2^2 + \beta \|G(\tilde{z})^{(i)} - \text{sg}[e(G(\tilde{z})^{(i)})]\|_2^2 \quad (2)$$

$$\ell_G = \mathbb{E}_{\tilde{z}}[-D(G(\tilde{z})) + \text{huber}(Q(G(\tilde{z})), \tilde{z}_c) + \ell_{VQ}(\tilde{z})] \quad (3)$$

$$(\mathcal{E}^*, \theta_G^*) = \arg \min_{\mathcal{E}, \theta_G} \ell_G \quad (4)$$

β is a hyperparameter for the *commitment loss* (second term in the VQ objective) and huber denotes the Huber loss. Note that Huber loss is used instead of ℓ_2 loss which corresponds to true maximum likelihood objective as ℓ_2 loss is very unstable.

Then the **training objective of the discriminator** is defined as:

$$\ell_D = \mathbb{E}_{\tilde{z}}[\max(0, 1 + D(G(\tilde{z}))) + \mathbb{E}_{\mathbf{x} \sim \mathcal{U}(\mathcal{X})}[\max(0, 1 - D(\mathbf{x}))]] \quad (5)$$

$$\theta_D^* = \arg \min_{\theta_D \text{ s.t. } D \in C^{1,1}} \ell_D \quad (6)$$

Note that the discriminator does not optimize \mathcal{E} . Including \mathcal{E} causes instability likely from competition between the generator and discriminator. The generator *cannot* cheat by forcing all embeddings to be the same (which will fool the discriminator that does not have control over the embedding) because of the mutual information constraint below.

The **training objective of the decoder** is defined as:

$$\ell_Q = \mathbb{E}_{\tilde{z}}[\text{huber}(Q(G(\tilde{z})), \tilde{z}_c)] \quad (7)$$

$$\theta_Q^* = \arg \min_{\theta_Q} \ell_Q \quad (8)$$

For the components shared with D, the 1-Lipschitz constraint is also enforced.

A.2 Embedding as Action

After training the architecture above, we have a function G that maps a noise vector \tilde{z} to a sentence. We train a SAC [23] agent π_h that outputs action $\mathbf{a} \in \mathbb{R}^{10}$. This action is mapped to an instruction $\mathbf{g} = G(\mathbf{a})$ which is in turn interpreted and executed by the low-level policy π_l . The policy is trained as a regular reinforcement learning agent on the high-level tasks.

A.3 Architecture Details

The generator network consists of a GRU cell with 128 hidden units. There are 50 available tokens for all the sentences in the environment and we choose the embedding size to be 64 so the embedding vector is of dimension 50×32 . The hidden state at each time step of the generator is linearly projected to dimension 32.

The discriminator architecture is a feed-forward 1D convolutional neural network with residual connections. The components shared by the discriminator and the decoder consists of 2 identical residual blocks, each of which consists of an identity block plus the output of convolution with window size 1 and output channel 32, leaky relu, convolution with window size 3 and output channel 64, and leaky relu. All of the convolution layers are spectrally normalized [40]. Following these are branches of two identical residual blocks – one for the discriminator and the other for the decoder. Finally, the outputs of both are reduced to a single scalar.

A.4 Experiments

We trained the above language model with 10000 randomly sampled instructions and trained a SAC agent using the resulting generator on the ColorSort task from the diverse setting. The diverse setting has much more possible instructions compared to the fixed object setting and can benefit more from having access to more instructions at the high-level policy. The comparison between language model and fixed instruction set (Discrete) is shown below in Figure 7.

Currently the language model agent performs comparably to the discrete agent. This may be due the fact that learning a disentangled generative model of language is difficult – in particular, we observed that the model tends to drop modes which can affect the final policy negatively. We expect the result to significantly improve with better language models or even visual-linguistic models. Further, the fixed instruction baseline shown above under-performs the one used in the main text because it *does not* use prioritized replay but instead uses uniform replay buffer. We observed decreased performance for SAC when prioritized replay is used; as such, we decided to use a uniform replay buffer for fair comparison.

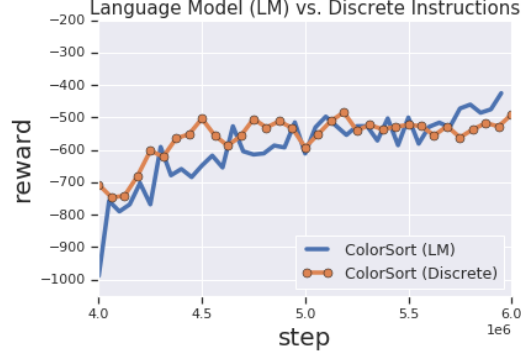


Figure 7: Results of an agent using a language model vs an agent picking from a fixed set of instructions.

B Environment, Model Architectures, and Implementation

In this section, we describe the environment and tasks we designed, and various implementation details such as the architectures of the policy networks.

B.1 CLEVR-Robot Environment

We want an environment where we can evaluate the agent’s ability to learn policies for long-horizon tasks that have compositional structure. In robotics, manipulating and rearranging objects is a fundamental way through which robots interact with the environment, which is often cluttered and unstructured. Humans also tend to decompose complex tasks into smaller sub-tasks in the environment (e.g. putting together a complicated model or write a piece of program). While previous works have studied the use of language in navigation domains, we aim to develop an environment where the agent can physically interact with and change the environment. To that end, we designed a new environment for language and manipulation tasks in MuJoCo where the agents must interact with the objects in the scene. To succeed in this environment, the agents must be able to handle a variety of objects with diverse visual and physical properties. Since our environment is inspired by the CLEVR dataset, we name our environment *CLEVR-Robot Environment*.

We will refer to all the elements in the environment collectively as the *world state*. The environment can contain up to 5 object. Each object is represented by a vector o_i that is the concatenation of *3d coordinate*, \mathbf{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 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 o_i ’s neighbors in 4 cardinal directions *left*, *right*, *front* and *behind*. The criterion for o_j to be the neighbor of o_i in certain direction is if $\|\mathbf{p}_j - \mathbf{p}_i\| \leq r_{\max}$ and the angle between $\mathbf{p}_j - \mathbf{p}_i$ and the cardinal vector of that direction is smaller than β_{\max} . After every interaction between the agent and the environment, o_i and the relation graph are updated to reflect the current world state.

The agent takes the form of a point mass that can teleport around, which is a mild assumption for standard robotic arms (Other agents are possible as well). 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.

B.2 Tasks

The high-level policy’s reward function can be tailored towards the task of interests. We propose three difficult benchmark tasks with extremely sparse rewards.

B.2.1 Five Object Settings (Standard)

In this setting, we have a fixed set of 5 spheres of different colors *cyan, purple, green, blue, red*.

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 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 exact arrangement constraints are: (1) red ball to the right of purple ball; (2) green ball to the right of red ball; (3) green ball to the right of cyan ball; (4) purple ball to the left of cyan ball; (5) cyan ball to the right of purple ball; (6) red ball in front of blue ball; (7) red ball to the left of green ball; (8) green ball in front of blue ball; (9) purple ball to the left of cyan ball; (10) blue ball behind the red ball

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 and 0.0 only if all statements are satisfied. The ordering of color is: *cyan, purple, green, blue, red* from left to right.

The third task is **object sorting**. In this task, the agent needs to sort 4 object around a central object; furthermore, 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 constraint is violated, and 0.0 only if all constraints are satisfied. Environment is reset until at most 1 constraint is satisfied. Images of the end goal for each high-level tasks are show in Figure 3.

B.2.2 Diverse Object Settings

Here, instead of 5 fixed objects, we introduce 3 different shapes *cube, sphere* and *cylinder* in combinations with 5 colors. Both colors and shapes can repeat but the same combination of color and shape does not repeat. In this setting, there are $\binom{15}{5} = 3003$ possible object configurations. In this setting, we define the color hierarchy to be *red, green, blue, cyan, purple* from left to right and the shape hierarchy to be *sphere, cube, cylinder* from left to right. Sample goal states of each task are shown in 3.

The first task is **color ordering** where the agent needs to manipulate the objects such that their colors are in ascending order.

The second task is **shape ordering** where the agent needs to manipulate the object such that their shapes are in ascending order.

Finally, the last task is **color & shape ordering** where the agent needs to manipulate the object such that the color needs to be in ascending order, and within each color group the shapes are also in ascending order.

Like in the fixed object setting, the agent only receives 0 reward when the objects are completely ordered; otherwise, the reward is always -10.

B.3 Implementation details

Language supervisor. 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 [28] 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 (*one-hop*) between the objects. We plan to scale up to full CLEVR complexity and possibly beyond in future works.

State based low-level policy. When we have access to the ground truth states of the objects in the scene, we use an object-centric representation of states by assuming $s_t = \{o_i\}_{i=1}^{k_t}$, where $o_i \in \mathbb{R}^{d_o}$ is the state representation of object i , and k_t is the number of objects (which can change over time). We also assume $a_t = \{\alpha_i\}_{i=1}^{k_t}$, where each $\alpha_i \in \mathbb{R}^{d_\alpha}$ acts on individual object o_i .

We implement a specialized universal value function approximator [55] for this case. 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 $\{o_i\}_{i=1}^k$, we first create pair-wise concatenation of the objects, $\mathcal{O} = \{o_i \| o_j\}_{j=1, i=1}^{k,k}$. Then we transform every pair-wise vectors with a single neural network f_1 into $\mathcal{Z} = \{f_1(o_i \| o_j)\}_{j=1, i=1}^{k,k}$. A recurrent neural network with GRU [11], f_2 , embeds the instruction g into a real valued vector $\tilde{g} = f_2(g)$. We use the embedding to attend over every pair of objects to compute weights $\{w_i = \langle \tilde{g}, z_i \rangle | z_i \in \mathcal{Z}\}$. We then compute a weighted combination of all p_i where the weights are equal to the softmax weights $\exp(w_i) / \sum_{j=1}^{k^2} \exp(w_j)$. This combination transforms the elements of \mathcal{Z} into a single vector \tilde{z} of fixed size. Each o_i is concatenated with \tilde{g} and \tilde{z} into $o'_i = (o_i \| \tilde{g} \| \tilde{z})$. Then each o'_i is transformed with the another neural network f_3 whose output is of dimension d_α . The final output $Q = \{f_3(o_i \| \tilde{g} \| \tilde{z})\}_{i=1}^k$ is in $\mathbb{R}^{k \times d_\alpha}$ which represents all state-action values of the state. Illustration of the architecture is shown in Figure 8.

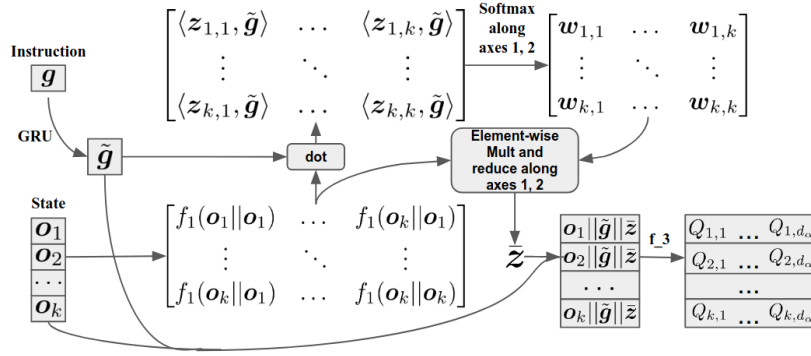


Figure 8: Computation graph of the state-based low level policy.

Image based low-level policy. In reality, we often do not have access to the state representation of the scene. For many application, a natural alternative is images. We assume $s_t \in [0, 1]^{W \times H \times C}$ is the available image representation of the scene (in all experimnts, $W=64, H=64, C=3$). Further, we need to adopt a more general action space since we no longer have access to the state representation (e.g. coordinates of the location). To that end, we discretize the 2D space into 10×10 grids and an action involves picking an starting location out of the 100 available grid and a direction out of the 8 cardinal direction to push. This induces an 800 dimensional discrete action space.

It is well-known that reinforcement learning from raw image observation is difficult, and even off-policy methods require millions of interaction on Atari games where the discrete action space is small. Increasing the action space would understandably make the already difficult exploration problem harder. A potential remedy is found in the fact that these high-dimensional action space can often be factorized into semantically meaningful groups (e.g. the pushing task can be break down into discrete bins of the x and y axes as well as a pushing direction). Previous works attempted to leverage this observation by using auto-regressive models or assuming conditional independence between the groups [39, 64]. We offer a new approach that aims to make the least assumptions. Following the group assumption, we assume there exists $m = 3$ groups and each group consists of k_m discrete *sub-actions* (i.e. $\mathcal{A}_m = \{a_m^{(1)}, a_m^{(2)}, \dots, a_m^{(k_m)}\}$). Following this definition, we can build a bijective

look-up map ζ between \mathcal{A} to tuples of sub-actions:

$$\mathcal{B} = \prod_{n=1}^m \{1, 2, \dots, k_n\} \quad (9)$$

$$\mathcal{A} \xrightarrow{\zeta} \{(\mathbf{a}_1^{(i_1)}, \dots, \mathbf{a}_m^{(i_m)}) \mid \forall (i_1, \dots, i_m) \in \mathcal{B}\} \quad (10)$$

We overload the notion $\mathbf{a}_m^{(i)}(s)$ to the *action feature* of $\mathbf{a}_m^{(i)}$ conditioned on the state and goal and $\zeta(\mathbf{a}, s)$ to be a tuple of the corresponding action features. Then the value function each action can be represented as:

$$Q(s, \mathbf{a}) = f_\psi(\zeta(\mathbf{a}, s)) \quad (11)$$

where f_ψ is a single neural network parameterized by ψ that is shared between all \mathbf{a} . This model does not require picking an order like the auto-regressive model and does not assume conditional independence between the groups. Most importantly, the number of parameters scales sublinearly with the dimension of the actions. The trade-off of this model is that it can be memory-expensive to model the full joint distribution of actions at the same time; however, we found that this model performs empirically well for the pushing tasks considered in this work. We will refer to this operation as *Tensor Concatenation*.

The overall architecture for the UVFA is as follows: the image input is fed through 3 convolution layers with kernel size $\{8, 5, 3\}$, stride $\{2, 2, 1\}$, and channel $\{46, 128, 64\}$; each convolution block is FiLM'd [48] with the instruction embedding. Then the activation is flattened spatially to 256×64 and projected to 28×64 . This is further split into 3 action group of sizes $\{10 \times 64, 10 \times 64, 8 \times 64\}$ and fed through tensor concatenation. f_ψ is parameterized by a 2-layer MLP with 512 hidden units at each layer and output dimension of 1 which is the Q-value.

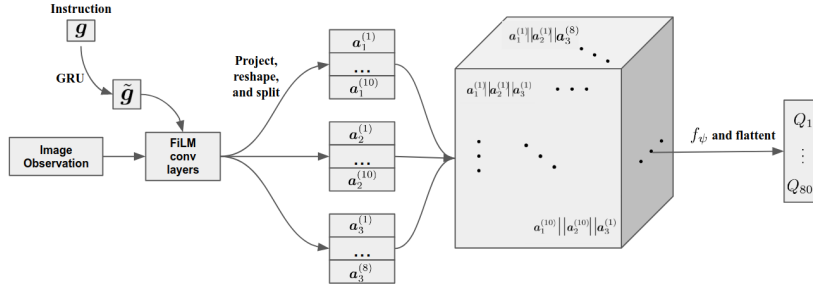


Figure 9: Computation graph of the vision-based low level policy.

Both policy networks are trained with HIR. Training mini-batches 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. For simplicity, we use a Double DQN [68] to train the high-level policy. We use an instruction set that consists of 80 instructions ($|\mathcal{I}| = 80$ for standard and $|\mathcal{I}| = 240$ for diverse) that can sufficiently cover all relationships between the objects. We roll out the low-level policy for 5 steps for every high-level instruction ($T' = 5$). Training mini-batches are uniformly sampled from the replay buffer. The state-based high-level policy uses the same observation space as the low-level policy; the image-based high-level policy uses extracted visual features from the low-level policy and then extracts salient spatial points with spatial softmax [35]. The convolutional weights are frozen during training. This design choice makes natural sense since humans also use a single visual cortex to process all initial visual signals, but training from scratch is certainly possible, if not ideal, should computation budget not matter. For the diverse visual tasks, we found that using the convolutional features with spatial softmax could not yield a good representation for the downstream tasks. Due to time constraints, the experiments shown for the diverse high-level tasks use the ground truth state, namely position, one-hot encoded colors, and shapes for the high-level policy; however, the low-level policy only has access to the image. We believe a learned convolutional layer would

solve this problem. Finally, we note that the high-level policy picks each sentence independent and therefore does not leverage the structure of language. While generating language is generally hard, a generative model would have more natural resemblance to *thought*. We think this is an extremely important and exciting direction for future work.

C Algorithms

In this section we elaborate on our proposed algorithm and lay out finer details.

C.1 Overall algorithm

The overall hierarchical algorithm is as follows:

Algorithm 1 Overall Hierarchical Training

- 1: **Inputs:** Low level RL algorithm \mathcal{A}_l ; High-level RL algorithm \mathcal{A}_h ; Environment \mathcal{E} ; other relevant inputs of Algorithms 2 and 3
 - 2: $\pi_l(\mathbf{a}|\mathbf{s}, \mathbf{g}) \leftarrow$ low-level policy trained with \mathcal{A}_l and other appropriate inputs (Algorithm 2)
 - 3: $\pi_h(\mathbf{g}|\mathbf{s}) \leftarrow$ high-level policy trained with \mathcal{A}_h , $\pi_l(\mathbf{a}|\mathbf{s}, \mathbf{g})$ and other appropriate inputs (Algorithm 3)
 - 4: **return** $\pi_l(\mathbf{a}|\mathbf{s}, \mathbf{g})$ and $\pi_h(\mathbf{g}|\mathbf{s})$
-

C.2 Training the low-level policy

For both state-based and vision-based experiments, we use DDQN as \mathcal{A}_l . For the state-based experiments, the agent receives binary reward based on whether the action taken completes the instruction; for the vision-based experiments, we found it instrumental to add an object movements bonus, i.e. if the agents change the position of the objects by some minimum threshold, the agent receives a 0.25 reward. This alleviates the exploration problem in high dimensional action space (\mathbb{R}^{800}), but the algorithm is capable of learning without the exploration bonus. (Algorithm 2). We adopt the similar setting as HER where unit of time consists of epochs, cycles and episode. Each cycle consists of 50 episode and each episode consists of 100 steps. While we set the number of epoch to 50, in practice we never actually reach there. We adopt an epsilon greedy exploration strategy where at the beginning of every cycle we multiply the exploration by a factor of 0.993, starting from 1 but at the beginning we 10 cycles to populate the buffer. The minimum epsilon is 0.1. We use $\gamma = 0.9$ and replay buffer of size 2e6. The target network we use is a 0.95 moving average of the model parameters, updated at the beginning of every cycle. Every episode, we update the network for 100 steps with the Adam Optimizer at minibatch of size 128 randomly sampled from the replay buffer.

C.3 Training the high-level policy

\mathcal{A}_h is also a DDQN. One single set of hyperparameter is used for standard experiments and another for the diverse experiments. DDQN is trained for 2e6 steps, uses uniform replay buffer of size 1e5, linearly anneals epsilon from 1 to 0.05 in 3e5 steps, Adam and batch size 256. $T' = 5$ for all our experiments, but the experience the network sees is equivalent of 1 step. For the diverse settings, due to time constraint, we use priority replay buffer of size 1e6 for all diverse experiment including the DDQN baselines. (Algorithm 3) We use the Adam Optimizer [31] with initial learning rate of 0.0001. Discount factor $\gamma = 0.9$. Learning starts after 100 episodes each of which lasts for 100 steps (100 high-level actions).

C.4 Relabeling Strategy

HER [4] 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 add all transition tuples in \mathbb{V}_t to the replay buffer \mathbb{B} (Algorithm 2). For

Algorithm 2 RL with hindsight instruction relabeling (HIR)

```
1: Inputs: off-policy RL algorithm  $\mathcal{A}_l$ ; instruction relabeling strategy  $\mathcal{S}$ ; language supervisor  $\Omega$ ;  
   Environment  $\mathcal{E}$ ; number of relabeled future  $K$   
2: Initialize replay buffer  $\mathbb{B}$  and  $\pi_l(a|s, g)$   
3: for episode  $i = 1$  to  $M$  do  
4:    $s_0 \leftarrow \text{reset } \mathcal{E}$   
5:    $g \sim \mathcal{U}(\{g \in \Omega(s_0) \mid \Psi(s_0, g) = 0\})$   
6:    $\tau \leftarrow []$   
7:   for step  $t = 0$  to  $T$  do  
8:      $\mathbb{U}_t \leftarrow \{g \in \Omega(s_t) \mid \Psi(s_t, g) = 0\}$   
9:      $a_t \sim \pi_{\mathcal{A}}(a|s_t, g)$   
10:     $s_{t+1} \leftarrow \text{Take action } a_t \text{ from } s_t$   
11:     $r_t \leftarrow \Psi(s_{t+1}, g)$   
12:     $\mathbb{V}_t \leftarrow \mathbb{U}_t \setminus \{g \in \Omega(s_{t+1}) \mid \Psi(s_{t+1}, g) = 0\}$   
13:    add  $(s_t, a_t, g, r_t, s_{t+1}, a_{t+1}, \mathbb{V}_t)$  to  $\tau$   
14:    if  $r_t = 1$  then  
15:       $g \sim \mathcal{U}(\{g \in \Omega(s_{t+1}) \mid \Psi(s_{t+1}, g) = 0\})$   
16:    end if  
17:  end for  
18:  for step  $t = 0$  to  $T$  do  
19:     $\mathbb{B} \leftarrow \mathbb{B} \cup \{(s_t, a_t, g, r_t, s_{t+1}, a_{t+1})\}$   
20:    for  $g' \in \mathbb{V}_t$  do  
21:       $\mathbb{B} \leftarrow \mathbb{B} \cup \{(s_t, a_t, g', 1, s_{t+1}, a_{t+1})\}$   
22:    end for  
23:     $\mathbb{W} \leftarrow \mathcal{S}(\tau, t, K)^\dagger$   
24:    for  $(g', r') \in \mathbb{W}$  do  
25:       $\mathbb{B} \leftarrow \mathbb{B} \cup \{(s_t, a_t, g', r', s_{t+1}, a_{t+1})\}$   
26:    end for  
27:  end for  
28:  Update  $\pi_l(a|s, g)$  with  $\mathcal{A}_l$  using minibatch from  $\mathbb{B}$   
29: end for  
30: return  $\pi_l(a|s, g)$   
31:  $\dagger$  Details in Appendix C.4;  $\mathcal{U}(\cdot)$  denotes uniform sampling from the given set.
```

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 provides sufficient learning signal. Detailed steps are shown below (Algorithm 4). In our experiments, we use $K = 4$.

In addition to relabeling, if an object is moved (using 800 dimensional action space), we add to replay buffer a transition where the instruction is the name of the object such as “*large rubber red ball*” with associated reward of 1.0. We found this helps the agent to learn the concept of objects. We refer to this operation as *Unary Relabeling*.

D Experimental Details

D.1 One-hot encoded representation

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 vector is embedded with a 2 layer MLP with 64 hidden units at each layer.

Algorithm 3 Training high-level policy

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

Algorithm 4 Future Instruction Relabeling Strategy (\mathcal{S})

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

D.2 Non-compositional representation

To faithfully evaluate the importance of compositionality, we want a representation that carries the identical information as the language instruction but without the explicit compositional property (but perhaps still to some degree compositional). To this end, we use a Seq2Seq [62] autoencoder with 64 hidden units to compress the 600 instructions into real-valued continuous vectors. The original tokens are fully recovered which indicates that the compression is *lossless* and the latent’s information content is the same as the original instruction. This embedding is used in place of the GRU instruction embedding. We also observed that adding regularization to the autoencoder decreases the performance of the resulting representation. For example, decreasing the bottleneck size leads to worse performance, as does adding dropout. Figure 4 uses an autoencoder with dropout of 0.5 while 1 uses one with no dropout. As shown in the experiments, the performance without dropout is better than the one with. We hypothesize that adding regularization decreases the compositionality of the representation.

D.3 Bag-of-Words representation

Bag-of-words is a popular technique in natural language processing where a sentence or corpus of texts are treated as a histogram of tokens – i.e. the order of the tokens are discarded and only the frequency at which a token occurs is considered. For each instruction, we count the occurrence of

each token available in the vocabulary and normalize the count by the length of the instruction. This vector is then embedded with a 2 layer MLP with 64 hidden units at each layer.

D.4 Non-hierarchical baseline

We use the Double DQN implementation from OpenAI baselines³. We use a 2 layer MLP with 512 hidden units at each layer with the same action dimension as the low-level policy.

D.5 HRL baselines

In general, we note that it is difficult to compare different HRL algorithms in an apples-to-apples manner. HIRO assumes a continuous goal space so we modified the goal to be an \mathbb{R}^{10} vector representing the locations of each object rather than language. In this regime, we observed HIRO was unable to make good progress. We hypothesize that the highly sparse reward might be the culprit. It is also worth noting that HIRO uses a goal space in \mathbb{R}^2 for navigation (which is by itself a choice of abstraction because the actual agent state space is much higher) while ours is of higher dimensionality. The Tensorflow implementation of HIRO we used can be found at the official Tensorflow repository⁴. (This is the implementation from the original author).

Option-critic aims to learn everything in a complete end-to-end manner which means it does not use the supervision from language. It is unsurprising that the sparse tasks do not provide sufficient signal for OC. In other words, our method enjoys the benefit of a flexible but fixed abstraction while OC needs to learn such abstraction. We tried 8, 16, and 32 options for OC but our method has much more sub-policies due to the combinatorial nature of language. The OC implementation in Tensorflow we used is open-sourced⁵.

D.6 Hardware Specs and Training time

All of our experiments are performed on a single Nvidia Tesla V100. We are unable to verify the specs of the virtual CPU. The low-level policy for state-based observation takes about 2 days to train and, for image-based observation, 6 days. The high-level policies for the state-based observation takes about 2 days to train and 3 days for image-based observations (wall-clock time). The implementations are not deliberately optimized for performance as the major bottleneck is actually the language supervisor so it is very likely the time could be dramatically shortened.

E More Experimental Results and Discussions

E.1 Low-level policy for a diverse environment

Figure 10 shows the training instruction per episode on the diverse environment. We see that the performance is worse than a fixed number of objects with the same amount of experience. This is perhaps not surprising considering the visual tasks are much more diverse and hence more challenging.

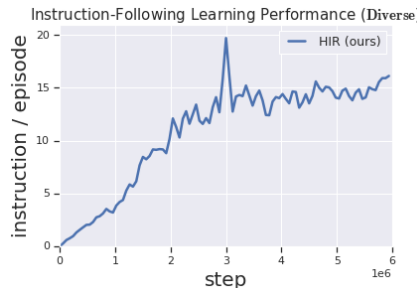


Figure 10: Results training the low-level policy on the diverse environment.

³<https://github.com/openai/baselines/tree/master/baselines/deepq>

⁴<https://github.com/tensorflow/models/tree/master/research/efficient-hrl>

⁵<https://github.com/yadrimz/option-critic>

E.2 Proposed environment, sparse reward, and structured exploration

While DDQN worked on 2 cases in the state-based environment, it is unable to solve any of the problems in visual domain. We hypothesize that the pixel observations and increase in action space ($20\times$ increase) makes the exploration difficult for DDQN. The difficulty of the tasks – in particular the 3 standard tasks – is reflected in the fact that the reward from non-hierarchical random action is stably 0 with small variance, meaning that under the sparse reward setting the agent almost never visits the goal states (states with non-zero reward). On the other hand, the random exploration reward is much higher for our method as the exploration in the space of language is *structured* as the exploration is aware of the task-relevant structure of the environment.