Learning Higher Order Skills that Efficiently Compose

Anthony Z. Liu [1 2] **Dong-Ki Kim** [2] **Sungryull Sohn** [2] **Honglak Lee** [1 2]

Abstract

Hierarchical reinforcement learning allows an agent to effectively solve complex tasks by leveraging the compositional structures of tasks and executing a sequence of skills. However, our examination shows that prior work focuses on learning individual skills without considering how to efficiently combine them, which can lead to sub-optimal performance. To address this problem, we propose a novel framework, called *second-order skills (SOS)*, for learning skills to facilitate the efficient execution of skills in sequence. Specifically, second order skills (which can be generalized to higher orders) aim to learn skills from an extended perspective that takes into account the next skill required to accomplish a task. We theoretically demonstrate that our method guarantees more efficient performance in the downstream task compared to previous approaches that do not consider second-order skills. Also, our empirical experiments show that learning second-order skills results in improved learning performance compared to state-of-the-art in baselines across diverse benchmark domains.

1. Introduction

Real world compositional tasks require long-horizon planning and can be difficult for reinforcement learning (RL) approaches to learn successful policies to solve them. An effective approach for solving these long-horizon tasks is hierarchical reinforcement learning (HRL) (Sutton et al., 1999), which decouples the complex learning problem into multiple simpler subproblems. In HRL, low level policies (also referred to as skills) are learned to solve these subproblems while a high level policy composes the learned skills by selecting different skills based on the state to solve the downstream task. Prior work in HRL has studied the problem of learning skills and composing the learned skills independently (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022). These skills are often learned in an unsupervised setting, where samples are easy to gather, and then fine-tuned in a downstream task (Eysenbach et al., 2018; Gregor et al., 2016; Choi et al., 2021).

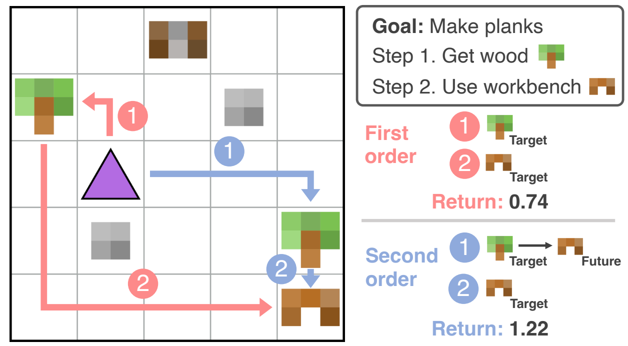

Figure 1. An example scenario from the Crafting domain (see Section 4.1), where the goal is to make planks by (1) getting a wood and then (2) using the workbench. Our **second-order** approach (in **blue**) achieves a higher return than the first-order baseline (in **red**) considered in prior work. **First-order**: the first-order skill greedily considers only the next skill to execute, leading the agent to grab the closest tree but move further from the workbench.

However, we highlight in this paper that when learning these skills in an unsupervised setting, it can be easy to learn skills that *do not compose together optimally* for the downstream task. This sub-optimality arises whenever there is ambiguity in how the agent should execute the skill, and when the agent only optimizes to maximizes the individual skill's reward. As a result, only maximizing individual skill reward can be detrimental when the agent must execute other skills in the future for the downstream task. Figure 1 shows an example of this sub-optimality issue, where the agent must first gather wood and then use a workbench to complete the task of making planks. Note that the example has ambiguity in which wood the agent should gather (i.e., either the left or right wood). If the agent looks ahead and considers the second step to complete the task, the agent can infer that gathering the wood closer to the workbench is a faster path than gathering the other wood.

With this insight, this paper introduces learning of a new

[1]University of Michigan, Ann Arbor, USA [2]LG AI Research, Ann Arbor, USA. Correspondence to: Anthony Z. Liu <anthliu@umich.edu>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

form of skills that maximize individual skill reward *while also maximizing future rewards for other skills*. We call these dual-maximizing skills *second-order skills* (SOS). Formally, every second-order skill is made of a *target skill* and a *future skill*, where the agent executes the target skill while maximizing future rewards for the future skill. This enables the agent to "look-ahead" to the future skill, enabling the agent to execute the target and future skill for a higher return than executing first-order skills for downstream tasks. Further, by iteratively applying second-order skills, we can extend the idea of second-order skills to nth order skills, which consist of n different skills with one target skill and $n-1$ different future skills. Analogous to how second-order skills are more efficient than first-order skills, composing nth order skills for downstream tasks is more efficient than composing $n-1$th order skills.

In summary, we make the following primary contributions:

- **Novel Skill Learning.** We define second-order skills (Section 3.1) and introduce the SOS algorithm for learning them using any off-policy RL algorithm (Section 3.2).

- **Theoretical Analysis.** We theoretically present that high level policies that execute second-order skills will be more efficient than only executing first order skills.

- **Empirical Evaluation.** We empirically show the effectiveness of learning second-order skills using two benchmark domains: the crafting domain (Andreas et al., 2016) and minigrid placement (Sohn et al., 2018; Liu et al., 2022).

2. Problem Statement

2.1. Hierarchical Reinforcement Learning

Our HRL setting considers a finite episodic MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, T)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the state transition probability, $\mathcal{R}$ is the reward function, γ is the discount factor, and T is the episodic horizon (Sutton and Barto, 1998; Sutton et al., 1999). A hierarchical agent aims to maximize the expected return for the downstream task by splitting the problem of learning into two: (1) Instead of learning behavior for the entire MDP, the agent learns a set of sub-behaviors $\mathcal{G}$ called *skills*. Specifically, the agent learns a low-level policy $\pi_{\text{LL}}(a|s,g)$ for each skill $g \in \mathcal{G}$, where $a \in \mathcal{A}$ and $s \in \mathcal{S}$, by maximizing the skill rewards $r_g(s,a)$. Each low-level policy also learns a termination policy $\beta_{\text{LL}}(\text{done}|s,a,g)$, predicting whether the current skill g is done (0 or 1) given the current state s and action a. (2) The HRL agent also learns a high-level policy $\pi_{\text{HL}}(g|s)$, which selects the best low-level policy (i.e., skill) to maximize the task return. The high-level policy executes the selected low-low policy g until it terminates.

2.2. Skill Pre-Training

We consider a variant of the HRL problem, where the agent learns the low-level skills $\mathcal{G}$ in isolation to the downstream task $\mathcal{R}$. This setting is considered in a variety of related work regarding unsupervised skill discovery (Eysenbach et al., 2018; Gregor et al., 2016) and goal conditioned RL (Nachum et al., 2018). These methods learn skills by training an RL policy on a given or inferred reward and use the learned skills to enable sample efficient learning for HRL. Following prior work (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022), we make two minor assumptions to make learning and deriving theoretical properties easier: (1) that skill rewards have already been inferred, and (2) that these skill rewards are non-negative (see Appendix A.2 for more details).

Ambiguity Issue. A critical issue in the unsupervised learning setting is the ambiguity problem, which occurs whenever a skill has multiple paths to successful execution. Specifically, when the agent executes a skill that has multiple paths of execution, the agent may arrive in a state that is sub-optimal for the downstream task. For example, Figure 1 shows the ambiguity in choosing which wood to collect, where getting left or right wood successfully terminates the skill but either choice results in a different performance. We address this ambiguity issue by introducing the concept of second-order skills in the next section.

3. Approach

This section formally defines second-order skills (SOS) and introduces a new algorithm for learning them using any off-policy RL algorithm. Intuitively, SOS learns a second-order skill $g_1 \rightarrow g_2$ that executes a skill g_1 while maximizing future return for a next skill g_2. We also present theoretical results, showing that when using second-order skills for HRL, SOS allows the agent to achieve a higher return than first-order skills on downstream tasks. Finally, we generalize our second-order approach to nth-order skills that can be defined by recursively applying second-orders to other second-order skills. Similarly, these nth-order skills can be learned using our SOS algorithm and can achieve higher theoretical HRL performance.

3.1. Higher-Order Skills and HRL

Second-Order Skills. Formally, given two skills g_1 and g_2, we define the second-order skill $g_1 \rightarrow g_2$ to be the skill that trains under the same MDP as skill g_1 but instead maximizes the following second-order reward function:

$$r_{g_1 \rightarrow g_2}(s,a) = \mathbb{E}_{s' \sim \mathcal{P}(s,a)} \left[\begin{cases} \widehat{Q}_{g_2}(s,a) & \text{if } s' \in S_{g_1} \\ 0 & \text{else} \end{cases} \right] \quad (1)$$

where S_{g_1} denotes a latent set of goal states determining when the agent has terminated the skill g_1 successfully, and $\widehat{Q}_{g_2}(\cdot,\cdot)$ denotes the estimated Q value for the skill g_2. Intuitively, Equation 1 states that when the agent executes g_1 successfully, it receives a reward of $\widehat{Q}_{g_2}(s,a)$, enabling the second-order skill $g_1 \rightarrow g_2$ to *look-ahead* toward g_2 while executing g_1. We show a visualization of value functions resulting from second-order skills reward in Section 4.4.

The notion of a successful skill is important when composing skills together for downstream tasks. The success function is generally unknown to the agent and must be estimated. In practice, we can estimate S_{g_1} using the reward r_{g_1} as the two are closely related:

$$S_{g_1} = \{s \in \mathcal{S} \mid r_{g_1}(s,a) > \rho\}, \tag{2}$$

where ρ is a hyperparameter.

nth-Order Skills. Because $g_1 \rightarrow g_2$ is itself a skill, nth-order skills can be defined by recursively applying second-order skills. For example, a third-order skill $g_1 \rightarrow g_2 \rightarrow g_3$ can be learned in the same way as second-order skills by first learning $g_2 \rightarrow g_3$ and then learning $g_1 \rightarrow (g_2 \rightarrow g_3)$.

nth-Order HRL. Similar to HRL, nth-order HRL learns both a low-level and high-level policy but the low-level policy learns nth-order skills. Specifically, we learn (1) A low-level policy for each nth order skill: $\pi_{\text{LL}}(a|s, g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n)$. The termination policy for the nth order skill is the same as the target skill's termination skill: $\beta_{\text{LL}}(\text{done}|s,a,g_1)$. (2) Then, the high-level policy is learned as $\pi_{\text{HL}}(g_1 \rightarrow g_2 \rightarrow \ldots \rightarrow g_n|s)$, which selects the nth-order skills for the low-level policy.

3.2. Learning Higher-Order Skills

This section details how second-order skills can be learned based on an off-policy RL algorithm. As with the definitions in Section 3.1, nth-order skills can be learned by recursively applying our learning algorithm on second-order skills.

Universal Value Function Approximation. Following (Schaul et al., 2015a), we use a single neural network to approximate value functions for each first-order and second-order MDPs: $\psi : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$, which takes state, action, and embeddings for two skills (sub-skill and final skill). Note that if the sub-skill and final skill are equal, ψ outputs the values that correspond to the first-order skill, so our definition of second-order skills generalizes to first-order skills.

Off-Policy RL with Second-Order Skills. We give the pseudo-code for learning second-order skills using any off-policy RL algorithm (e.g., double DQN (Hasselt et al., 2015)) in Algorithm 1 in the appendix. Using the current Q estimate ψ, we estimate the second-order reward using the current Q estimate ψ.

3.3. Theoretical Properties

Optimality. We present the main theoretical result in Theorem 3.1, showing that second-order skills are more efficiently executed than first-order skills.

Theorem 3.1. *Let g_1 and g_2 be two skills with non-negative rewards r_{g_1} and r_{g_2}. Let $g_1 \rightarrow g_2$ be the second order skill for g_1 and g_2. Suppose there is a downstream task with a reward function $\mathcal{R}$ that requires skills g_1 and g_2 to be executed in sequence. Let π^*_x be the optimal policy for each skill, and trajectories τ_1 and τ_2 are generated by executing skills in sequence by using first-order and second-order skills, respectively:*

$$\begin{aligned}\tau_1 \sim \{&(s_1, \pi^*_{g_1}(s_1)), \ldots, (s_i, \pi^*_{g_1}(s_i)), \\ &(s_{i+1}, \pi^*_{g_2}(s_{i+1})), \ldots\}\end{aligned} \tag{3}$$

$$\begin{aligned}\tau_2 \sim \{&(s_1, \pi^*_{g_1\rightarrow g_2}(s_1)), \ldots, (s_j, \pi^*_{g_1\rightarrow g_2}(s_j)), \\ &(s_{j+1}, \pi^*_{g_2}(s_{j+1})), \ldots\}\end{aligned} \tag{4}$$

Then the return of second-order skill trajectory τ_2 is always more optimal or equal to the return of first-order skill trajectory τ_1:

$$R(\tau_2) \geq R(\tau_1), \tag{5}$$

where R refers to the expected return of a trajectory.

Our proof is shown in Appendix A.3.

Corollary 3.2. *Any downstream task that requires skills to be executed in some order can be more efficiently solved using second-order skills than first-order skills. Similarly, nth-order skills can solve these downstream tasks more efficiently than n'th-order skills for any $n' < n$.*

This can be shown by recognizing that any order of skill execution on a downstream task can be made more efficient by using the result in Theorem 3.1.

4. Experiments

4.1. Environments

Minigrid Placement. Inspired by block placement games, such as Sokoban (Kartal et al., 2021) and PushWorld (Kansky et al., 2023), we create a simple placement environment in the popular Minigrid Gymnasium environment (Chevalier-Boisvert et al., 2018) (see Figure 2). In this environment, an agent must place two balls (red and blue) into a goal area. However, this task is not trivial to solve as the agent must think ahead and put the first ball further into the goal space so that the agent is not blocked when placing the next ball into the remaining goal space. Two skills are given to the agent: placing the red/blue balls into the goal area. We expect the first-order skills to be short-sighted and immediately place the balls in the closest position, rendering

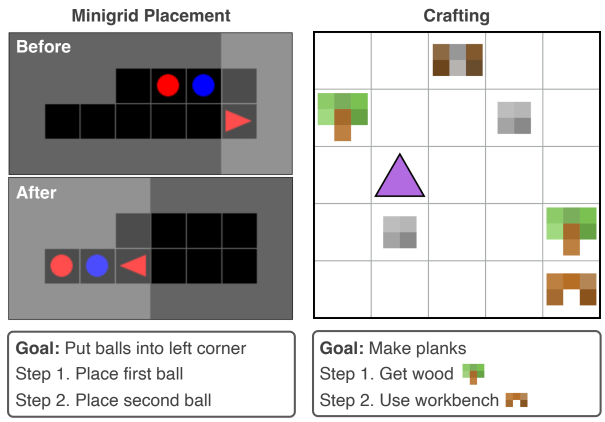

Figure 2. Visualizations of the Minigrid Placement and Crafting domain (e.g., `make plank` task) in this paper. A sequence of skills is required to complete a goal in each domain.

the HRL agent unable to solve this environment (since the agent can no longer place the next ball).

Crafting. We implement the Crafting domain from Andreas et al. (2016) in the Minigrid Gymnasium environment. This domain is inspired by minecraft, where the agent must craft various objects using various materials and tools in the environment (see Figure 2). We give the agent 6 low-level skills to learn: `get wood`, `get grass`, `get iron`, `use toolshed`, `use workbench`, `use factory`. Using these skills, the agent must then learn how to execute the following 8 downstream tasks (we use the `make` tasks from Andreas et al. (2016)): `make plank`, `make stick`, `make cloth`, `make rope`, `make bridge`, `make bed`, `make axe`, `make shears`. The agent is also given a sketch (or a plan) for which low-level skills to execute in order to solve each downstream task. For example, to solve `make stick`, the agent is given the sketch: `[get wood, use workbench]`. The full sketch details can be found in the appendix (Andreas et al., 2016).

4.2. Implementation

We learn each environment using a double DQN (Hasselt et al., 2015) with prioritized experience replay (Schaul et al., 2015b), and dueling networks (Wang et al., 2015). We use a similar neural network in each environment: the network uses a 3-layer MLP over a concatenated representation of the state, and one hot encoding of skill(s) (one skill for first-order, and two skills for second-order). We give more details on the architecture in Appendix A.5.

4.3. Results

We trained skills on each environment (Minigrid Placement and Crafting) for 5 million and 10 million timesteps respectively. For the baseline, all timesteps were used to train first-order skills. For SOS, timesteps were used to train first-order and second-order skills in parallel. Then, we trained a high-level DQN policy which executes the pretrained skills on the downstream task. We show the results of the high-level policy in Table 1 and learning curve in Figure 3. The full per-task return for Crafting is shown in the Appendix, Figure 5.

Table 1. Second-Order Skills HRL DQN Return

		Placement	Crafting
DQN	1st-order	0 ± 0	0.67 ± 0.04
	2nd-order	$\mathbf{1 \pm 0}$	$\mathbf{0.82 \pm 0.03}$
Oracle	∞-order	1	1.02

On Placement, we found that first-order skills were unable to solve the downstream task at all (total return of 0) while second-order skills solved the task (total return of 1). This is because the first-order skills always greedily placed the balls in the nearest goal position, which blocks the agent from placing the next ball. The second-order skills allowed the agent to "look-ahead" and place the first ball in a further position, so that the next ball could be placed correctly.

On Crafting, we also found that first-order skills performed worse than second-order (0.67 ± 0.04 to 0.82 ± 0.03 return over 8 seeds). Similar to what we showed in the Crafting example, Figure 1, we found that second-order skills could find more optimal paths to complete the tasks. We also found that second-order skills enabled the agents to solve tasks at a higher rate (refer to Figure 5 in the Appendix), where the second-order skills can consistently solve the difficult long-horizon tasks better (`make bridge`, `make bed`, `make axe`, and `make shears`). We believe this is due to another property of second-order skills: executing second-order skills leaves agents in a state that it is unlikely to get stuck on, as they maximize the future return on other skills. This enables the high-level agent to succeed since the second-order skills are more likely to succeed.

4.4. Analysis

Second-Order Skills Visualization. We visualize the optimal value functions of second-order skills in Figure 4 in an imagined environment similar to Crafting, where an agent must travel to subgoals α then β. We can see from Figure 4 that $V^*_{\alpha\to\beta}$ can be thought of as a "reweighted" version of V^*_α, but with the values from V^*_β. From this Figure, we can also see that when an agent must visit α then β, the agent should (most of the time) visit α_2 rather than α_1, since it is closer to β and the value of $V^*_{\alpha\to\beta}$ is higher for α_2.

Oracle nth-Order Implementation. We also implemented oracle nth skills for skills orders 1 through 4. We evaluated

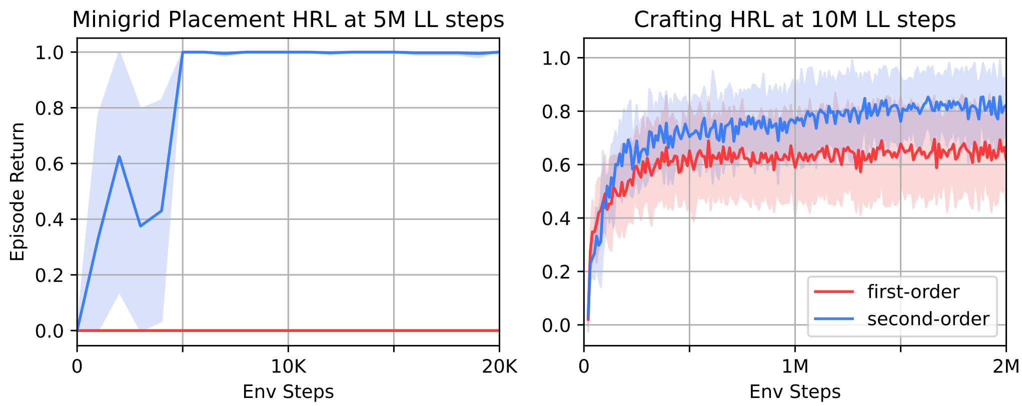

Figure 3. Episode returns of the HRL agent after training the low level skills for 5M and 10M environment steps for Placement and Crafting respectively. We plotted the mean and standard deviations over 8 seeds. For Crafting, we plot the mean return over all 8 Crafting tasks.

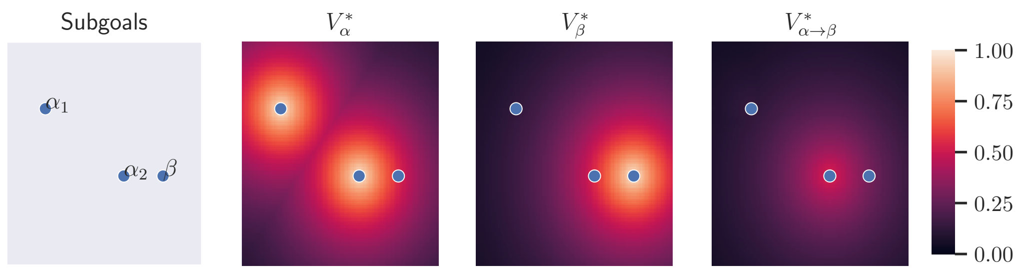

Figure 4. A visualization of a toy environment similar to Crafting where the agent must visit α and β. We visualize the optimal value functions for the skills α, β and second-order skill $\alpha \to \beta$. The values for second-order skill $\alpha \to \beta$, $V^*_{\alpha\to\beta}$ indicate that the agent should visit α_2 most of the time if the agent will visit β afterword.

Table 2. nth-Order HRL Theoretical Return

		Placement	Crafting
Oracle	1st-order	0	0.89 ± 0.00
	2nd-order	1	1.01 ± 0.00
	3rd-order	1	1.02 ± 0.00
	4th-order	1	1.02 ± 0.00

them on the Placement and Crafting environments for 10000 episodes and show the mean reward ($\pm$ standard error) in Table 2.

We find that in Placement, since there are only 2 skills needed, 2nd-order skills can immediately solve the task. However, in Crafting, the maximum number skills needed to solve a single downstream task is 4. Even so, there are diminishing returns from 2nd-order to 3rd-order to skills. 3rd and 4th order skills are indistinguishable. This implies that for many tasks, second-order skills gives us sufficient approximations of policies for nth-order skills.

5. Related Work

Unsupervised Skill Discovery. Unsupervised skill discovery aims to learn skills without the necessity of manually defining and tuning reward functions for desired behaviors. One of the most common approaches to the unsupervised skill discovery problem is to maximize the mutual information (MI) between skill latent variables and states (Gregor et al., 2016; Achiam et al., 2018; Eysenbach et al., 2018; Hansen et al., 2019; Sharma et al., 2019; Choi et al., 2021; Zhang et al., 2021), leading to the discovery of diverse and distinguishable behaviors. The works has been extended to use different posterior models (Hansen et al., 2019), spectral normalization (Choi et al., 2021), and variants of the MI identity (Zhang et al., 2021) to further improve the skill performance. However, these methods share a limitation: they do not consider the behavioral optimality when the learned skills are combined for down-stream tasks. Our work propose a novel framework designed to learn skills that can be executed efficiently when combined together.

Option-Based HRL. The option framework studies discovering temporally extended high-level actions, called options, to achieve efficient learning (Sutton et al., 1999; Precup and Sutton, 2000). By reducing the effective number of decision makings, options improve an agent's learning efficiency for solving a long-horizon task (Omidshafiei et al., 2018). Recent frameworks show that an agent can learn both option and termination policies end-to-end without requiring prior knowledge (Bacon et al., 2017; Riemer et al., 2018; Abdulhai et al., 2022). Our main idea of extending the perspective of skills by considering the next skill also applies to option learning settings.

Goal-Conditioned HRL. Another related work is the goal-conditioned setting, where a high-level policy generates a sequence of subgoals for a low-level policy to follow (Kulkarni et al., 2016). Nachum et al. (2018) focused on improving learning efficiency using off-policy correction or hindsight mechanisms. Chane-Sane et al. (2021); Lo et al. (2022); Zhang et al. (2020) proposed efficient subgoal sampling methods to avoid challenges of sparse reward when the subgoal is hard or impossible to reach. Savinov et al. (2018); Laskin et al. (2020); Huang et al. (2019); Hoang et al. (2021) proposed to model the relationship between subgoals in a graph form to enable navigating between subgoals via planning. However, these approaches learn goal-conditioned policy by optimizing to reach *any* state that belongs to the given subgoal, which may induce a sub-optimal behavior (Nachum et al., 2019). Thus, our work propose a novel way to improve skill learning to enable more efficient composition of skills in down-stream tasks.

6. Conclusion

In this paper, we have introduced second-order skills (SOS), which learns second-order skills to efficiently compose for solving downstream tasks. Our key idea is to extend the perspective skills by looking ahead to next skills that will be executed after the current. We evaluated our method of learning second-order skills on the gridworld benchmark domains and showed that SOS performs more effectively than the first-order baseline.

References

Marwa Abdulhai, Dong-Ki Kim, Matthew Riemer, Miao Liu, Gerald Tesauro, and Jonathan P. How. Context-specific representation abstraction for deep option learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):5959–5967, Jun. 2022. doi: 10.1609/aaai.v36i6.20541. URL https://ojs.aaai.org/index.php/AAAI/article/view/20541.

Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Jayant Joshi, Ryan C. Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego M Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, F. Xia, Ted Xiao, Peng Xu, Sichun Xu, and Mengyuan Yan. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2022.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *ArXiv*, abs/1611.01796, 2016.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 1726–1734. AAAI Press, 2017.

Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In *International Conference on Machine Learning*, pages 1430–1440. PMLR, 2021.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for gymnasium, 2018. URL https://github.com/Farama-Foundation/Minigrid.

Jongwook Choi, Archit Sharma, Honglak Lee, Sergey Levine, and Shixiang Shane Gu. Variational empowerment as representation learning for goal-conditioned reinforcement learning. In *International Conference on Machine Learning*, pages 1953–1963. PMLR, 2021.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *ArXiv*, abs/1802.06070, 2018.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *ArXiv*, abs/1611.07507, 2016.

Steven Hansen, Will Dabney, Andre Barreto, Tom Van de Wiele, David Warde-Farley, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

H. V. Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2015.

Christopher Hoang, Sungryull Sohn, Jongwook Choi, Wilka Carvalho, and Honglak Lee. Successor feature landmarks for long-horizon goal-conditioned reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 26963–26975, 2021.

Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. In *Advances in Neural Information Processing Systems (NeurIPS*, pages 1940–1950, 2019.

Ken Kansky, Skanda Vaidyanath, Scott Swingle, Xinghua Lou, Miguel Lázaro-Gredilla, and Dileep George. Pushworld: A benchmark for manipulation planning with tools and movable obstacles. *ArXiv*, abs/2301.10289, 2023.

Bilal Kartal, Nick Sohre, and Stephen J. Guy. Data driven sokoban puzzle generation with monte carlo tree search. In *Artificial Intelligence and Interactive Digital Entertainment Conference*, 2021.

Tejas D. Kulkarni, Karthik R. Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 3682–3690, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Michael Laskin, Scott Emmons, Ajay Jain, Thanard Kurutach, Pieter Abbeel, and Deepak Pathak. Sparse graphical memory for robust planning, 2020.

Anthony Z. Liu, Sungryull Sohn, Mahdi Qazwini, and Honglak Lee. Learning parameterized task structure for generalization to unseen entities. *ArXiv*, abs/2203.15034, 2022.

Chunlok Lo, Gabor Mihucz, Adam White, Farzane Aminmansour, and Martha White. Goal-space planning with subgoal models. *arXiv preprint arXiv:2206.02902*, 2022.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Neural Information Processing Systems*, 2018.

Ofir Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? *arXiv preprint arXiv:1909.10618*, 2019.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *International Conference on Machine Learning*, 2017.

Shayegan Omidshafiei, Dong-Ki Kim, Jason Pazis, and Jonathan P. How. Crossmodal attentive skill learner. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, page 139–146. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Doina Precup and Richard S. Sutton. *Temporal Abstraction in Reinforcement Learning*. PhD thesis, 2000.

Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/cdf28f8b7d14ab02d12a2329d71e4079-Paper.pdf.

Nikolay Savinov, Alexey Dosovitskiy, and Vladlen Koltun. Semi-parametric topological memory for navigation. In *International Conference on Learning Representations (ICLR)*, 2018.

Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015b.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

Sungryull Sohn, Junhyuk Oh, and Honglak Lee. Multitask reinforcement learning for zero-shot generalization with subtask dependencies. *ArXiv*, abs/1807.07665, 2018.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112: 181–211, 1999.

Ziyun Wang, Tom Schaul, Matteo Hessel, H. V. Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, 2015.

Jesse Zhang, Haonan Yu, and Wei Xu. Hierarchical reinforcement learning by discovering intrinsic options. *arXiv preprint arXiv:2101.06521*, 2021.

Tianren Zhang, Shangqi Guo, Tian Tan, Xiaolin Hu, and Feng Chen. Generating adjacency-constrained subgoals in hierarchical reinforcement learning. *Advances in Neural Information Processing Systems*, 33:21579–21590, 2020.

A. Appendix

A.1. Second-Order Skills Pseudocode

Algorithm 1 Learning Second-Order Skills Pseudocode

given
  A distribution of skills or goals $\mathcal{G}$.
  An off-policy RL algorithm $\mathbb{A}$ ▷ DQN, DDPG, etc.
    with Q network $\psi : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \times \mathcal{G} \to \mathbb{R}$. ▷ ψ takes a state, action, goal, and final goal
  A reward function $r : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$.
  A skill success threshold $\rho \geq 0$.
Initialize $\mathbb{A}$ and Q network ψ.
Initialize replay buffer $\mathcal{D}$.
for episode $= 1, \dots, M$ **do**
  Sample a goal g and final goal g' from $\mathcal{G}$.
  for $t = 1, \dots, T$ **do**
    Sample and execute action a_t using policy $\mathbb{A}$:
      $a_t \leftarrow \pi(s_t \| g \| g')$. ▷ $\|$ denotes concatenation
    Let $r_t \leftarrow r(s_t, a_t, g)$.
    Let t denote whether the transition is terminal.
    Store the transition $(s_t \| g \| g', a_t, r_t, s_{t+1} \| g \| g', t)$ into $\mathcal{D}$.
    if $update$ **then**
      Sample a mini-batch B from replay buffer $\mathcal{D}$.
      Let $B_{\text{SOS}} = [\,]$. ▷ Convert B to a second order mini-batch
      for $(s \| g \| g', a, r, s' \| g \| g', t)$ in B **do**
        if $g \neq g'$ **then** ▷ If second order
          $r \leftarrow \mathbb{I}[r > \rho] \psi(s \| g' \| g', a)$.
          $t \leftarrow t \vee (r > \rho)$.
        Append $(s \| g \| g', a, r, s' \| g \| g', t)$ to B_{SOS}.
      Perform an optimization step on ψ with B_{SOS} using $\mathbb{A}$

A.2. Discussion of Second-Order Reward

Non-Negativity. Following prior work (Oh et al., 2017; Andreas et al., 2016; Ahn et al., 2022), we assume that skill rewards are non-negative. Sparse or goal-reaching reward, commonly used in prior work and are used here, are already non-negative, as they give positive rewards when a goal or sub-goal is reached, and zero otherwise.

We also note that many of skill rewards (Eysenbach et al., 2018; Achiam et al., 2018) can also be translated into a non-negative reward using a monotonic function such as $\tan$, or simply flooring the reward such as $r' = \max(0, r)$. However, as this non-trivially changes the skills learned, we leave investigating how to translate reward for second-order skills to future work.

The non-negativity assumption is required since second-order skills may not be beneficial in specific cases when rewards are negative. Specifically, consider two skills g_1 and g_2, and a second-order skill $g_1 \to g_2$. Then, under Equation (1), $g_1 \to g_2$ could learn behavior that accumulates negative reward for g_2 before g_1 is completed.

ρ-Threshold Value. Recall that we estimate the success of a skill using the approximation in Equation (2). The choice of the threshold ρ is set as a hyperparameter. Note that in sparse reward settings, $\rho = 0$ is sufficient, as the reward r is greater than zero only when success is achieved. ρ can also be learned by taking setting ρ to be the Xth percentile reward observed by the agent.

In the most general scenario, success can be modeled using some learned distribution:

$$S_{g_1} \sim p_\theta(\cdot \mid s, a) \tag{6}$$

We leave generalizations of success and second-order skills to future work.

A.3. Proof of Theorem 3.1

Before we give the proof of Theorem 3.1, we state our formal definition of a reward function that requires skills to be executed in sequence.

Definition A.1. A reward function $\mathcal{R}$ is a reward function that requires skills g_1 and g_2 to be executed in sequence if it follows two properties:

1. $\mathcal{R}$ only gives reward if g_2 is completed (i.e. $s \in S_{g_2}$) *after* g_1 is completed (i.e. $s \in S_{g_1}$). (With the implicit assumption that history of completion is tracked in the MDP).

2. $\mathcal{R}$ is "aligned" with g_2. We specify that behavior for maximizing the the skill reward for g_2, r_{g_2}, also maximizes $\mathcal{R}$ when g_1 is completed. Formally: Let πg_2 be a policy for g_2. For some state s, if g_1 is completed ($s \in S_{g_1}$), and τ_π is the trajectory generated by π on s, then

$$R_{r_{g_2}}(\tau_\pi) > R_{r_{g_2}}(\tau_{\pi'}) \Rightarrow R_{\mathcal{R}}(\tau_\pi) > R_{\mathcal{R}}(\tau_{\pi'}) \tag{7}$$

where R_x is the expected return for task x.

Following this definition, we give the proof for Theorem 3.1 below:

Proof. (For ease of reading, we will restate the conditions.) We have skills g_1 and g_2 with non-negative rewards r_{g_1} and r_{g_2}. We have a downstream task $\mathcal{R}$ which requires g_1 and g_2 to be executed in sequence (defined above).

Let π^*_x be the optimal policy for a skill x, (including second-order $g_1 \to g_2$). For notation, we denote $\tau_1(s)$ as the first-order-only trajectory generated by first completing g_1 using $\pi^*_{g_1}$, then completing g_2 using $\pi^*_{g_2}$, and $\tau_2(s)$ as the second-order trajectory generated by first completing g_1 using $\pi^*_{g_1 \to g_2}$, then completing g_2 using $\pi^*_{g_2}$. I.e.

$$\begin{aligned} \tau_1(s_1) &\sim \{(s_1, \pi^*_{g_1}(s_1)), \dots, (s_i, \pi^*_{g_1}(s_i)), (s_{i+1}, \pi^*_{g_2}(s_{i+1})), \dots\} \\ \tau_2(s_1) &\sim \{(s_1, \pi^*_{g_1 \to g_2}(s_1)), \dots, (s_j, \pi^*_{g_1 \to g_2}(s_j)), (s_{j+1}, \pi^*_{g_2}(s_{j+1})), \dots\} \end{aligned} \tag{8}$$

We will show that for all $s \in \mathcal{S}$, $R_{\mathcal{R}}(\tau_2(s)) \geq R_{\mathcal{R}}(\tau_1(s))$. We break this into two cases:

(1) If g_1 is already completed: $s \in S_{g_1}$. Then the two trajectories are equal, since they both execute $\pi^*_{g_2}$, so $R_{\mathcal{R}}(\tau_2(s)) = R_{\mathcal{R}}(\tau_1(s))$.

(2) If g_2 is not completed: $s \notin S_{g_1}$.

Let s'_1 denote the state reached after executing $\pi^*_{g_1}$ on s, and let s'_2 denote the state reached after executing $\pi^*_{g_1 \to g_2}$ on s. Note that $s'_1 \in S_{g_1}$ and $s'_2 \in S_{g_1}$.

Then, we can show that $R_{\mathcal{R}}(\tau_2(s'_2)) \geq R_{\mathcal{R}}(\tau_1(s'_1))$. We show by the contrapositive. Suppose that there is some s'_1 and s'_2 such that $R_{\mathcal{R}}(\tau_2(s'_2)) < R_{\mathcal{R}}(\tau_1(s'_1))$.

Then by the contrapositive on the Definition in Equation (7), $R_{r_{g_2}}(\tau_2(s'_2)) < R_{r_{g_2}}(\tau_1(s'_1))$. However, this contradicts with our definition of the second-order skill $g_1 \to g_2$ and the optimal policy for the skill $\pi^*_{g_1 \to g_2}$. Since, the path to s'_1 is strictly higher rewarding than s'_2 according to the second-order reward definition in Equation (1).

Thus, $R_{\mathcal{R}}(\tau_2(s'_2)) \geq R_{\mathcal{R}}(\tau_1(s'_1))$, and it directly follows that $R_{\mathcal{R}}(\tau_2(s)) \geq R_{\mathcal{R}}(\tau_1(s))$, $\square$

A.4. Hyperparameters

Table 3. Minigrid Placement Hyperparameters

discount γ	0.9
learn start	20000
epsilon greedy	0.1
learning rate	1e-4
gradient norm clip	10
replay frequency	4
target update frequency	4000
batch size	32
prioritized replay weight	0.4
prioritized replay exponent	0.5
max memory buffer size	1000000
success threshold ρ	0

Table 4. Crafting Hyperparameters

discount γ	0.9
learn start	20000
epsilon greedy	0.1
learning rate	1e-4
gradient norm clip	10
replay frequency	8
target update frequency	4000
batch size	128
prioritized replay weight	0.4
prioritized replay exponent	0.5
max memory buffer size	6000000
success threshold ρ	0.5

A.5. Network Architecture

For the Crafting and Placement environments, we use a similar architecture: Each position in the grid is encoded into an object ID and mapped into a learned encoding. The grid encoding is flattened, then concatenated with a one-hot encoding of the direction of the agent, and a one-hot encoding of the agents skill(s) (two skills if second-order). Lastly, this encoding is fed into a 3-layer MLP.

For crafting, we additionally concatenate a one-hot encoding of the task sketch (a list of the subgoals needed to complete the task).

A.6. Crafting Task Results

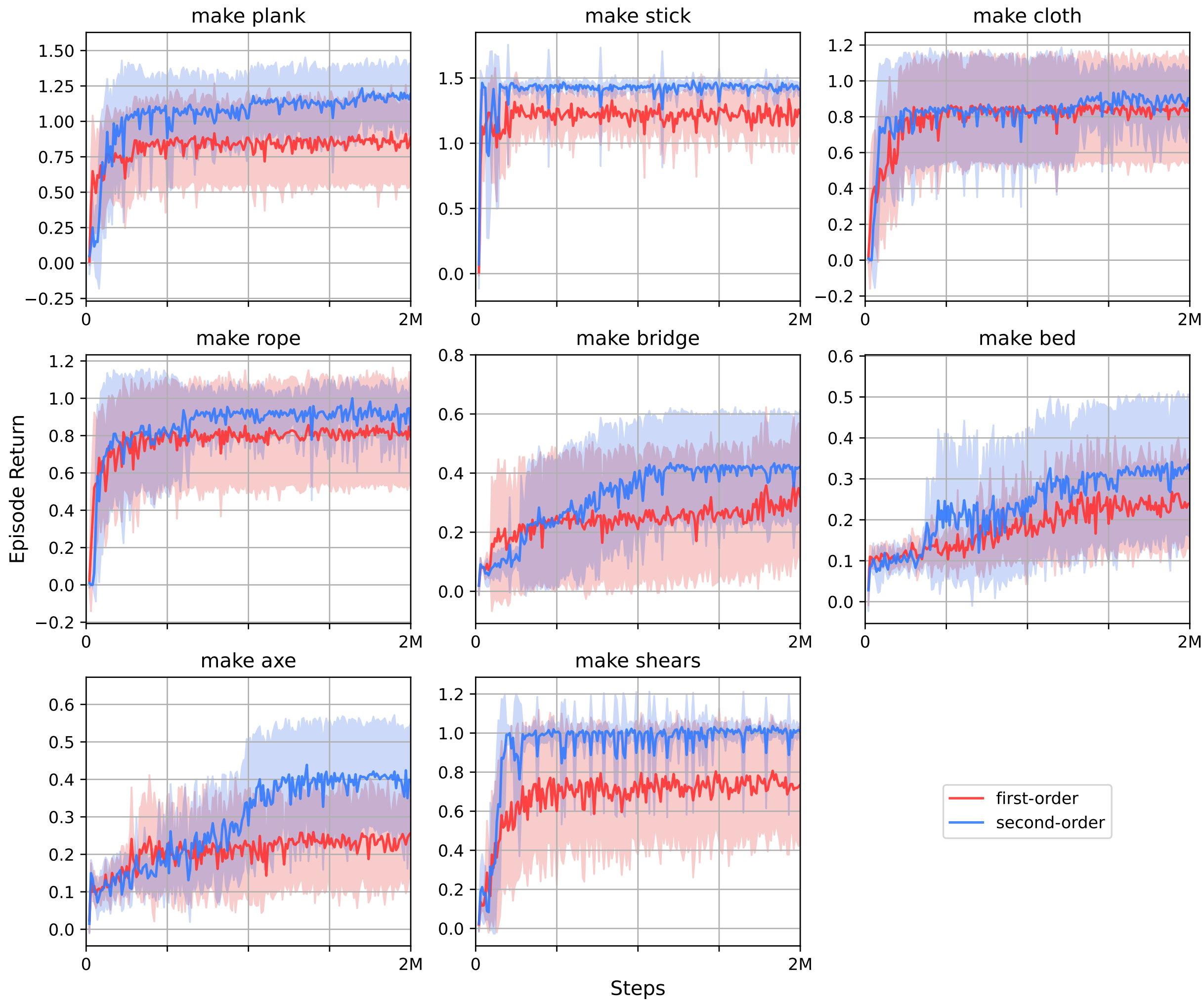

Figure 5. Episode returns for each task of the HRL agent on Crafting after training the low level skills for 10M environment steps. We plotted the mean and standard deviations over 8 seeds.