Diversity-Driven Extensible Hierarchical Reinforcement Learning

Submission ID: 2804

Abstract

Hierarchical reinforcement learning (HRL) has recently shown promising advances on speeding up learning, improving the exploration, and discovering intertask transferable skills. Most recent works focus on HRL with two levels, i.e., a master policy manipulates subpolicies, which in turn manipulate primitive actions. However, HRL with multiple levels is usually needed in many real-world scenarios, whose ultimate goals are highly abstract, while their actions are very primitive. Therefore, in this paper, we propose a diversity-driven extensible HRL (DEHRL), where an extensible and scalable framework is built and learned levelwise to realize HRL with multiple levels. DEHRL follows a popular assumption: diverse subpolicies are useful, i.e., subpolicies are believed to be more useful if they are more diverse. However, existing implementations of this diversity assumption usually have their own drawbacks, which makes them inapplicable to HRL with multiple levels. Consequently, we further propose a novel diversity-driven solution to achieve this assumption in DEHRL. Experimental studies evaluate DEHRL with five baselines from four perspectives in two domains; the results show that DEHRL outperforms the state-of-the-art baselines in all four aspects.

1 Introduction

Hierarchical reinforcement learning (HRL) recombines sequences of basic actions to form subpolicies (Sutton, Precup, and Singh 1999; Parr and Russell 1998; Dietterich 2000). It can be used to speed up the learning (Bacon, Harb, and Precup 2017), improve the exploration to solve tasks with sparse *extrinsic rewards* (i.e., rewards generated by the environment) (Şimşek and Barto 2004), or learn meta-skills that can be transferred to new problems (Frans et al. 2018). Although most previous approaches to HRL require hand-crafted subgoals to pretrain subpolicies (Heess et al. 2016) or extrinsic rewards as supervisory signals (Vezhnevets et al. 2016), the recent ones seek to discover subpolicies without manual subgoals or pretraining. Most of them are working in a top-down fashion, such as (Xu et al. 2017; Bacon, Harb, and Precup 2017), where a given agent first explores until it accomplishes a trajectory that reaches a positive extrinsic reward. Then, it tries to recombine the basic actions in the trajectory to build useful or reasonable subpolicies.

However, such top-down solutions are not practical in some situations, where the extrinsic rewards are sparse, and the action space is large (called *sparse extrinsic reward problems*); this is because positive extrinsic rewards are almost impossible to be reached by exploration using basic actions in such scenarios. Therefore, more recent works focus on discovering "useful" subpolicies in a bottom-up fashion (Lakshminarayanan et al. 2016; Kompella et al. 2017), which are capable of discovering subpolicies before reaching a positive extrinsic reward. In addition, the bottom-up strategy can discover subpolicies that better facilitate learning for an unseen problem (Frans et al. 2018). This is also called meta-learning (Finn, Abbeel, and Levine 2017), or more precisely meta-reinforcement-learning (Al-Shedivat et al. 2018), where subpolicies shared across different tasks are called meta-subpolicies (or meta-skills).

However, none of the above methods shows the capability to build extensible HRL with multiple levels, i.e., building subpolicies upon subpolicies, which is usually needed in many real-world scenarios, whose *ultimate goals* are highly abstract, while their basic *actions* are very primitive. We take the game called *OverCooked* and shown in Fig. 1 as an example. The ultimate goal of OverCooked is to let an *agent* fetch multiple *ingredients* (green box) in a particular sequence according to a *to-pick list* (yellow box), which is shuffled in every episode. However, the basic action of the agent is so primitive (thus called *primitive action* in the following) that it can just move one of its four legs towards one of the four directions at each step (marked by red arrows), and the agent's body can be moved only when all four legs are moved to the same direction. Consequently, although we can simplify the task by introducing subpolicies to learn to move the body towards different directions with four steps of primitive actions, the ultimate goal is still difficult to be reached, because the to-pick list changes every episode.

Fortunately, this problem can be easily overcome if HRL has multiple levels: by defining the previous subpolicies as subpolicies at level 0, HRL with multiple levels can build subpolicies at level 1 to learn to fetch different ingredients based on subpolicies at level 0; obviously, a policy based on subpolicies at level 1 is capable to reach the ultimate goal more easily than that based on subpolicies at level 0.

Motivated by the above observation, in this work, we propose a diversity-driven extensible HRL (*DEHRL*) approach, which is constructed and trained levelwise (i.e., each level shares the same structure and is trained with exactly the same algorithm), making the HRL framework extensible to build higher levels. DEHRL follows a popular *diversity assumption*: diverse subpolicies are useful, i.e., subpolicies are believed

Figure 1: Playing OverCooked with HRL of three levels.

to be more useful if they are more diverse. Therefore, the objective of DEHRL at each level is to learn corresponding subpolicies that are as diverse as possible, thus called diversity-driven.

However, existing implementations of this diversity assumption usually have their own drawbacks, which make them inapplicable to HRL with multiple levels. For example, (i) the implementation in (Daniel, Neumann, and Peters 2012) works in a top-down fashion; (ii) the one in (Haarnoja et al. 2018) cannot operate different layers at different temporal scales to solve temporally delayed reward tasks; and (iii) the implementation in (Gregor, Rezende, and Wierstra 2016; Florensa, Duan, and Abbeel 2017) is not extensible to higher levels.

Consequently, we further propose a novel diversity-driven solution to achieve this assumption in DEHRL: We first introduces a *predictor* at each level to dynamically predict the resulting state of each subpolicy. Then, the diversity assumption is achieved by giving higher *intrinsic rewards* to subpolicies that result in more diverse states; consequently, subpolicies in DEHRL converge to taking actions that result in most diverse states. Here, *intrinsic rewards* are the rewards generated by the agent.

We summarize the contributions of this paper as follows:

- We propose a diversity-driven extensible hierarchical reinforcement learning (DEHRL) approach. To our knowledge, DEHRL is the first learning algorithm that is built and learned levelwise with verified scalability, so that HRL with multiple levels can be realized end-to-end without human-designed extrinsic rewards.
- We further propose a new diversity-driven solution to implement and achieve the widely adopted diversity assumption in HRL with multiple levels.
- Experimental studies evaluate DEHRL with five baselines from four perspectives in two domains. The results show that, comparing to the baselines, DEHRL achieves the following advantages: (i) DEHRL can discover useful subpolicies more effectively, (ii) DEHRL can solve the sparse extrinsic reward problem more efficiently, (iii) DEHRL can learn better intertask-transferable meta subpolicies, and (iv) DEHRL has good portability.

2 Diversity-Driven Extensible HRL

This section introduces the new DEHRL framework as well as the integrated diversity-driven solution. The structure of DEHRL is shown in Fig. 2, where each level l contains a *policy* (denoted as π^l), a *predictor* (denoted as β^l), and an *estimator*. Above *policy* and *predictor* are two deep neural networks (i.e., parameterized functions) with π^l and β^l denoting their trainable parameters, while *estimator* only contains some operations without trainable parameters. For any two neighboring levels (e.g. level l and level $l-1$), three connections are shown in Fig. 2:

- Policy at the upper level π^l produces action a_t^l that is treated as an input for policy at the lower level π^{l-1};
- Predictor at the upper level β^l makes several predictions, which are passed to the estimator at the lower level $l-1$;
- Using the predictions from the upper level l, the estimator at the lower level $l-1$ generates an intrinsic reward b_t^{l-1} to train policy at the lower level π^{l-1};

2.1 Policy

As shown in Fig. 2, the policies for different levels act at different frequencies, i.e., policy π^l samples an action every T^l steps. Note that T^l is always an integer multiple of T^{l-1} and T^0 always equals to 1, so the time complexity of the proposed framework does not grow linearly as the level goes higher. T^l for $l>0$ are hyper-parameters. At level l, policy π^l takes as input the current state s_t and the action from upper level a_t^{l+1}, so that the output of π^l is conditional to a_t^{l+1}. Note that $a_t^{l+1} \in \mathbb{A}^{l+1}$, where $\mathbb{A}^{l+1}$ is the output action space of π^{l+1}. Thus, $\mathbb{A}^0$ should be set to the action space of the *environment* for policy π^0 directly taking actions on the environment, while $\mathbb{A}^l$ of $l>0$ are hyper-parameters. The policy takes as input both s_t and a_t^{l+1} to integrate multiple subpolicies into one model; a similar idea is presented in (Florensa, Duan, and Abbeel 2017). Detailed network structure of policy π^l is presented in supplementary material. Then, policy π^l produces the action $a_t^l \in \mathbb{A}^l$ by sampling from a parameterized categorical distribution:

$$a_t^l = \pi^l(s_t, a_t^{l+1}). \tag{1}$$

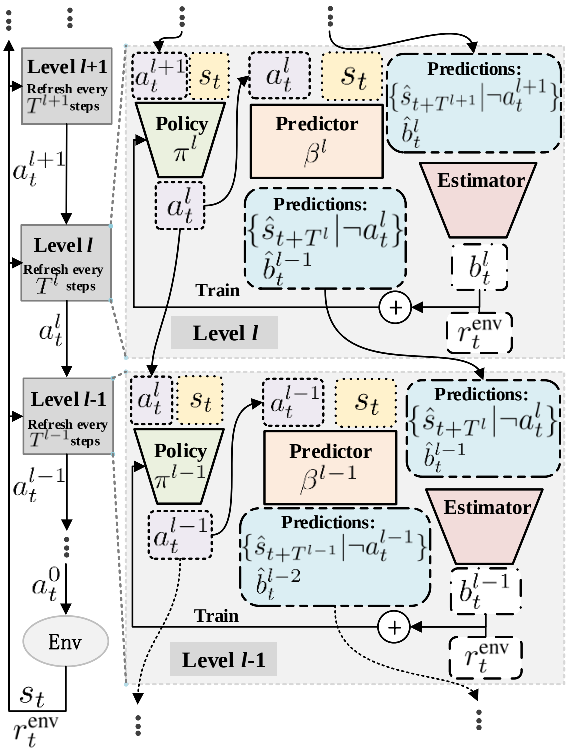

Figure 2: The framework of DEHRL.

The reward to train π^l combines the extrinsic reward from the environment r_t^{env} and the intrinsic reward b_t^l generated from the estimator at level l. When facing games with very sparse extrinsic rewards, where r_t^{env} is absent most of the time, b_t^l will guide policy at this level π^l to learn diverse subpolicies, so that the upper level policy π^{l+1} may reach the sparse positive extrinsic reward more easily. The policy π^l is trained with the PPO algorithm (Schulman et al. 2017), but our framework does not restrict the policy training algorithm to use. Following denotes the loss of training policy π^l:

$$L^l_{\text{policy}} = \text{PPO}\left(a_t^l, (r_t^{\text{env}} + \lambda b_t^l)|\pi^l\right), \tag{2}$$

where π^l means that the gradients of this loss are only passed to the parameters in π^l, and λ is a hyper-parameter set to 1 all the time (the section of estimator below will introduce a normalization of intrinsic reward b_t^l, making λ free from careful tuning).

2.2 Predictor

As shown in Fig. 2, the predictor at level l (i.e., β^l) takes as input the current state s_t and the action taken by the policy at level l (i.e., a_t^l) as a one-hot vector. The integration of s_t and a_t^l is accomplished by approximated multiplicative interaction (Oh et al. 2015), so that any predictions made by predictor β^l is conditioned on the action input of a_t^l. The predictor makes two predictions, denoted as $\hat{s}_{t+T^l}$ and $\hat{b}_t^{l-1}$, respectively. Thus, the forward function of predictor β^l is:

$$\hat{s}_{t+T^l}, \hat{b}_t^{l-1} = \beta^l(s_t, a_t^l). \tag{3}$$

Detailed network structure of predictor β^l is presented in supplementary material. The first prediction $\hat{s}_{t+T^l}$ in (3) is trained to predict the state after T^l steps with following loss function:

$$L^l_{\text{transition}} = \text{MSE}(s_{t+T^l}, \hat{s}_{t+T^l}|\beta^l), \tag{4}$$

where MSE is the mean square error, and β^l indicates that the gradients of this loss are only passed to the parameters in β^l. The second prediction $\hat{b}_t^{l-1}$ in (3) is trained to approximate the intrinsic reward at the lower level b_t^{l-1}, with the loss function

$$L^l_{\text{intrinsic reward}} = \text{MSE}(b_t^{l-1}, \hat{b}_t^{l-1}|\beta^l), \tag{5}$$

where β^l means that the gradients of this loss are only passed to the parameters in β^l. The next section about estimator will show that intrinsic reward b_t^{l-1} is also related to the action a_t^{l-1} sampled according to policy at the lower level π^{l-1}. Since a_t^{l-1} is not fed into the predictor β^l, intrinsic reward $\hat{b}_t^{l-1}$ is actually an estimation of the expectation of b_t^{l-1} under the current π^{l-1}:

$$\hat{b}_t^{l-1} = \mathbb{E}_{\pi^{l-1}}\{b_t^{l-1}\}. \tag{6}$$

The above two predictions will be used in the estimator, described in the following section.

The predictor is active at the same frequency as that of the policy. Each time the predictor β^l is active, it produces several predictions feeding to the estimator at level $l-1$, including $\hat{b}_t^{l-1}$ and $\{\hat{s}_{t+T^l}|\neg a_t^l\}$, where $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ denotes a set of predictions when feeding predictor β^l with actions other than a_t^l.

2.3 Estimator

As shown in Fig. 2, taking as input $\hat{b}_t^{l-1}$ and $\{\hat{s}_{t+T^l}|\neg a_t^l\}$, the estimator produces intrinsic reward b_t^{l-1}, which is used to train policy π^{l-1}, as described in the policy section. The design of the estimator is motivated as follows:

- If the currently selected subpolicy $\pi^{l-1}(s_t, a_t^l)$ for the upper-level action a_t^l differs from the subpolicies for other actions (i.e., $\{\pi^{l-1}(s_t, \neg a_t^l)\}$), then the intrinsic reward b_t^{l-1} should be high;
- The above difference can be measured via the distance between the states resulting from subpolicy $\pi^{l-1}(s_t, a_t^l)$ and subpolicies $\{\pi^{l-1}(s_t, \neg a_t^l)\}$. Note that since a_t^l is selected every T^l steps, these resulting states are the ones at $t + T^l$;

In the above motivation, the resulting state of subpolicy $\pi^{l-1}(s_t, a_t^l)$ is the real state environment returned after T^l steps (i.e., s_{t+T^l}), while the resulting states of subpolicies $\{\pi^{l-1}(s_t, \neg a_t^l)\}$ have been predicted by the predictor at the upper level l (i.e., $\{\hat{s}_{t+T^l}|\neg a_t^l\}$), as described in the last section. Thus, intrinsic reward b_t^{l-1} is computed as follows:

$$b_t^{l-1} = \sum_{s \in \{\hat{s}_{t+T^l}|\neg a_t^l\}} D(s_{t+T^l}, s), \tag{7}$$

where D is the distance chosen to measure the distance between states. In practice, we combine the L1 distance and

Table 1: The settings of DEHRL.

$\mathbb{A}^0$	$\mathbb{A}^1$	$\mathbb{A}^2$	T^0	T^1	T^2
16	5	5	1	1*4	1*4*12

the distance between the center of mass of the states, to obtain information on color changes as well as objects moving. A more advanced way to measure the above distance is to match features in states and to measure the movements of the matched features, or to integrate the inverse model in (Pathak et al. 2017) to capture the action-related feature changes. However, the above advanced ways are not investigated here, due to the scope of the paper. Equation (7) gives a high intrinsic reward, if s_{t+T^l} is far from $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ overall. In practice, we find that punishing s_{t+T^l} from being too close to the one state in $\{\hat{s}_{t+T^l}|\neg a_t^l\}$ that is closest to s_{t+T^l} is a much better choice. Thus, we replace the sum in (7) with the minimum, and find it consistently gives the best intrinsic reward estimation.

Estimating the intrinsic reward with distances of high dimensional states comes with the problem that the changes in distance that we want the intrinsic reward to capture is extremely small, comparing to the mean of the distances. Thus, we use the estimation of the expectation of intrinsic reward b_t^{l-1} (i.e., $\hat{b}_t^{l-1}$ described in last section) to normalize b_t^{l-1}:

$$b_t^{l-1} = b_t^{l-1} - \hat{b}_t^{l-1}. \quad (8)$$

In practice, this normalization gives a stable algorithm without need to tune λ according to the distance that we choose or the converge status of the predictor at the upper level. We jointly optimize the loss functions of (2), (4), and (5).

3 Experiments

We conduct experiments to evaluate DEHRL and five baselines based on two games, OverCooked (shown in Fig. 1) and MineCraft. All methods are implement to perceive the state via raw image input, which are down-sampled to 84×84 and gray-scaled. The important hyper-parameters of DEHRL are summarized in Table 1, while other details (e.g., neural network architecture and hyper-parameters in the policy training algorithm) are provided in the supplementary material. To keep a fair comparison, submodels (e.g., convolution layers) that are owned by both DEHRL and baselines are designed to share the same structures. Easy-to-run codes have been released to further clarify the details and facilitate future research.

3.1 Subpolicy Discovery

We first evaluate DEHRL in OverCooked to see if it discovers diverse subpolicies more efficiently, comparing to the state-of-the-art baselines towards option discovery (Florensa, Duan, and Abbeel 2017; Bacon, Harb, and Precup 2017).

As shown in Fig. 1, an agent in OverCooked can move one of the four legs towards one of the four directions at each step, so its action space is 16. Only after all four legs are moved towards a same direction, The body of an agent can be moved towards this direction, and all four legs are then reset. There are four different ingredients at the corners of the kitchen (marked by green box). An ingredient is automatically picked up when the agent reaches it. The left-down corner shows a list of ingredients that the chief needs to pick in sequence to complete a dish (marked by red box), called to-pick list.

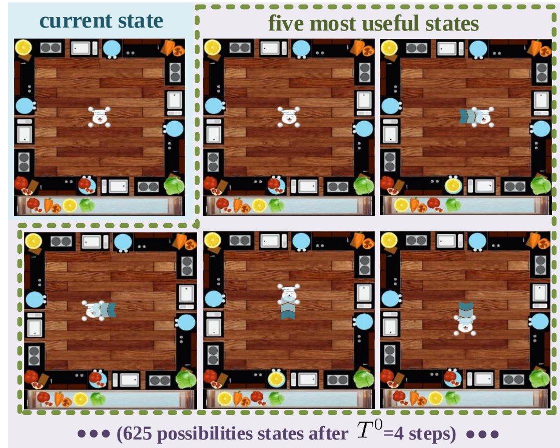

Figure 3: State examples of Overcooked.

Without Extrinsic Rewards. We first aim to discover useful subpolicies without extrinsic rewards. Since there are four legs, and every leg of an agent has five possible states (staying or moving towards four directions), there are totally $5^4 = 625$ possible states for every $T^0 = 4$ steps. As shown in Fig. 3, five of them are the most useful states (i.e., the ones that are most diversity to each other), whose four legs have the same state, making the body of the chief move towards four directions or stay still.

Consequently, a good implementation of diversity assumption should be able to learn subpolicies at level 0 that can result in the five most useful states (called *five useful subpolicies*) efficiently and comprehensively. Therefore, given $\mathbb{A}^1 = 5$ (i.e., discovering only five subpolicies), and the number of steps is 10 millions, the five subpolicies learned by DEHRL at level 0 are exactly the five useful subpolicies. Furthermore, *SNN* (Florensa, Duan, and Abbeel 2017) is a state-of-the-art implementation of diversity assumption, which is thus tested as a baseline under the same setting. However, only one of the five useful subpolicies is discovered by SNN. We then repeat experiments three times with different training seeds, and the results are the same. Furthermore, we loose the restriction by setting $\mathbb{A}^1 = 20$ (i.e., discovering 20 subpolicies) and the number of steps is 20 millions. With no surprise, the five useful subpolicies are always included in the 20 discovered subpolicies; however, the 20 subpolicies discovered by SNN still contain only one useful subpolicy.

The superior performance of DEHRL comes from the diversity-driven solution, which gives higher intrinsic rewards to subpolicies that result in more diverse states; consequently, subpolicies in DEHRL converge to taking actions that result in most diverse states. And the failure of SNN may be because: The objective of SNN is to maximize mutual information, so it only guarantees to discover subpolicies resulting in different states, but these different states are not guarantees to be most diverse to each other. Similar failures are found in other state-of-the-art solutions (e.g., (Gregor, Rezende, and Wierstra 2016)), we thus omit the analysis due to space limit.

As for finding useful subpolicies at higher level, due to the failures at level 0, none of the state-of-the art solutions can generate useful subpolicies at higher level. However, useful subpolices can be learned by DEHRL at higher level. Fig. 4 visualizes five subpolicies learned by DEHRL at level

Table 2: The different settings of extrinsic rewards in OverCooked.

reward-level	*goal-type: any* (easy)	*goal-type: fix* (middle)	*goal-type: random* (hard)
1 (easy)	Get any ingredient.	Get a particular ingredient.	Get the first ingredient shown in the shuffling* to-pick list.
2 (hard)	Get 4 ingredients in any order.	Get 4 ingredients in a particular order.	Get 4 ingredients in order according to the shuffling* to-pick list.

* The to-pick list is shuffled every episode.

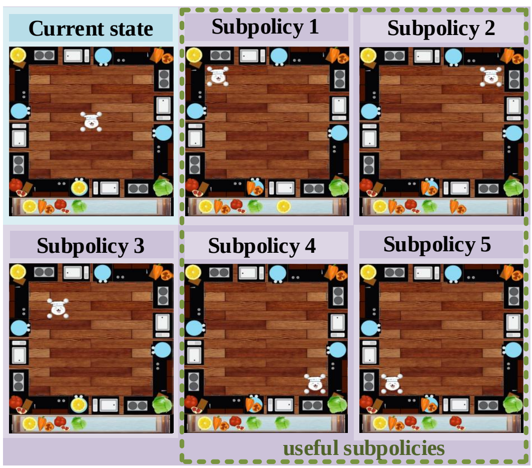

Figure 4: Subpolicies learned at level 1 in DHEL.

1, where four of them (marked by green box) result in getting four different ingredients, which are the useful subpolicies at level 1. Obviously, these four useful subpolicies are very helpful for the harder task of fetching a sequence of ingredients at level 2. Consequently, we can assert that DEHRL is much more effective than the baselines in discovering useful subpolicies at all levels without extrinsic rewards.

With Extrinsic Rewards. Although DEHRL can work in a bottom-up fashion where no extrinsic reward is required to discover useful subpolicies, DEHRL actually also have very superior performance in the scenarios when extrinsic rewards are given. Therefore, we compare DEHRL with a state-of-the-art top-down method, called *option-critic* (Bacon, Harb, and Precup 2017), where extrinsic rewards are essential. As shown in Table 2, six different extrinsic reward settings are given to OverCooked, resulting in different difficulties.

To measure the performance quantitatively, two metrics, *final performance score* and *learning speed score*, which are based on reward per episode, are imported from (Schulman et al. 2017). Generally, the higher the reward per episode, the better the solution. Specifically, final performance score averages reward per episode over last 100 episodes of training to measure the performance at final stages; while learning speed score averages extrinsic reward per episode over entire training period to quantify the learning efficiency.

The results of final performance score and learning speed score are shown in Table 3 and Fig. 5, respectively. In Table 3, we find that DEHRL can solve the problems in all six settings, while option-critic can only solve the two easier ones (with the last four harder cases scored 0). The failure of option-critic is because the extrinsic reward gets more sparse (i.e., sparse extrinsic reward problem) in the last four harder cases. The results in Fig. 5 also exhibits the similar finding.

Table 3: Final performance score of DEHRL and baselines on OverCooked with six different extrinsic reward settings.

reward-level *goal-type*	*1* *any*	*1* *fix*	*1* *random*	*2* *any*	*2* *fix*	*2* *random*
DEHRL	**1.00**	**1.00**	**1.00**	**0.95**	**0.93**	**0.81**
Option-critic	**1.00**	**1.00**	0.00	0.00	0.00	0.00
PPO	0.98	0.97	0.56	0.00	0.00	0.00
State Novelty	**1.00**	0.96	0.95	0.00	0.00	0.00
Transition Novelty	**1.00**	**1.00**	**1.00**	0.00	0.00	0.00

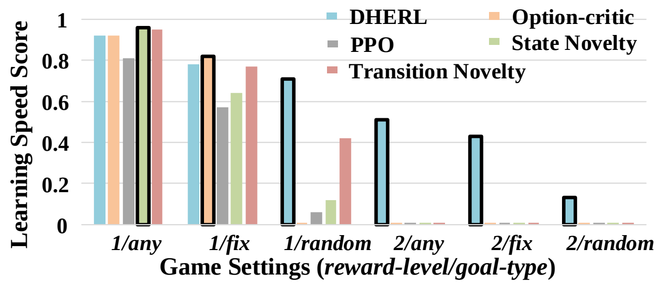

Figure 5: Learning speed score of DEHRL and the baselines on OverCooked with six different extrinsic reward settings.

Consequently, we state that DEHRL can also achieve better performance than the state-of-the-art baseline when extrinsic rewards are given.

3.2 Solving Sparse Extrinsic Reward Problem

As option-critic fails in sparse extrinsic reward problem, to illustrate the advantage of our method in solving this problem, we further DEHRL with two state-of-the-art methods with better exploration strategies, namely *state novelty* (Şimşek and Barto 2004) and *transition novelty* (Pathak et al. 2017). In addition, as previous mentioned, our framework is based on *PPO* algorithm (Schulman et al. 2017), so we include PPO as a baseline as well. The evaluations are also based on final performance scores and learning speed scores which are shown in Table 3 and Fig. 5, respectively.

The results show that, better than option-critic, three new baselines are all able to solves the task in third setting. However, they all still fails in the last three settings. This asserts that the proposed framework in DEHRL can solve the sparse extrinsic reward problem more effectively.

we also note that although achieving better performance in harder settings, DEHRL performs slightly worse than state novelty and transition novelty on the first setting, and slightly worse than option-critic in the second setting.The reason is that DEHRL has to first learn subpolicies that move towards four directions at level 0 and then learn reaching food with policy at level 1. However, the first two settings are so easy that the agent only needs to reach one ingredient at one

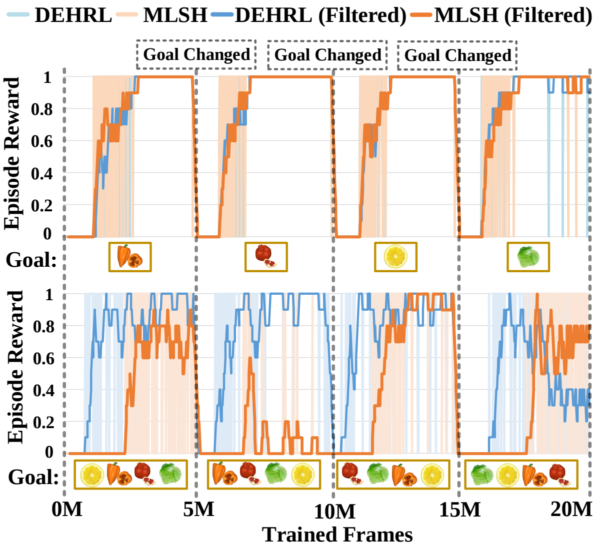

Figure 6: Meta HRL performances of DEHRL and MLSH in OverCooked with reward-level=1 (upper) and reward-level=2 (downer).

corner; consequently, not all four directions are essential in such easy cases and some of the learning processes in DEHRL are wasted.

3.3 Meta HRL

HRL recently shows a promising ability to learn meta-subpolicies that better facilitate adaptive behavior for new problems (Solway et al. 2014; Frans et al. 2018). We compare DEHRL against the state-of-the-art MLSH framework in (Frans et al. 2018)to investigate such performance.

We first test both DEHRL and MLSH in OverCooked with *reward-level=1* and *goal-type=random*. In order to make MLSH work properly, instead of changing the goal every episode as originally designed in *goal-type=random*, the goal in MLSH is changed every 5M steps; to keep fair comparison, top-level hierarchy of DEHRL is also reset every 5M steps (same as MLSH). The results of the episode extrinsic reward curves in 20M steps (the goal is changed five times) are shown in Fig. 6 (upper part). As expected, the episode extrinsic reward drops every time the goal is changed, since the top-level hierarchies of both methods are re-initialized. And the increase speed of the episode extrinsic reward after each reset can measure the performance of methods in learning meta-subpolicies that facilitate adaptive behaviors for a new goal. Consequently, we find that DEHRL and MLSH have similar meta-learning performances under this setting.

In addition, we repeat the above experiment with *reward-level=2* and the results are shown in Fig. 6 (lower part). We find that DEHRL produces a better meta-learning ability than MLSH. This is because DEHRL is capable to learn the subpolicies at level 1 to fetching four different ingredients, while MLSH can only learn subpolicies to moving towards four different directions (similar to the subpoicies at level 0 of DEHRL). Obviously, based on the better intertask-transferable subpolicies learned at level 1, DEHRL will resolve the new goal more easily and quickly than MLSH. Therefore, this finding shows that DEHRL can learn meta subpolicies at higher levels, which are usually better intertask-transferable than those learned by the baseline.

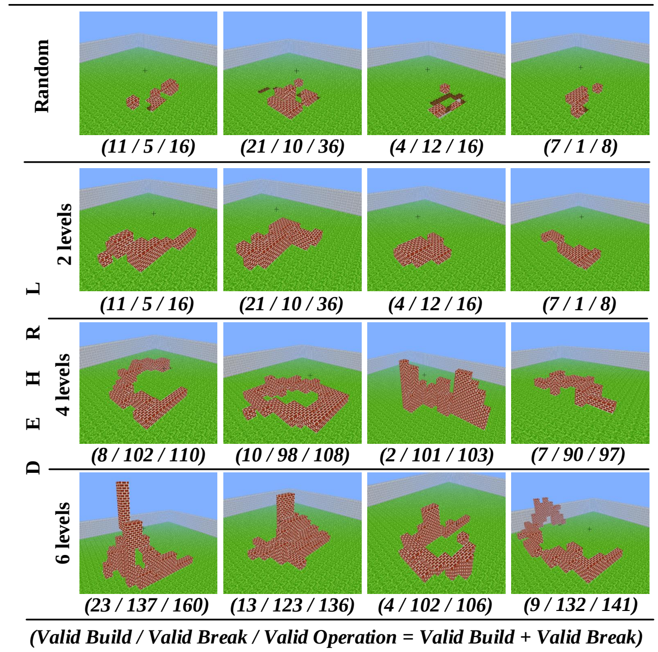

Figure 7: Worlds built by playing MineCraft without extrinsic reward.

3.4 Application of DEHRL in MineCraft

The above evaluations are all based on the OverCooked domain. However, DEHRL are actually able to be applied to many other domains and will also achieve very impressive performances. To show the portability of DEHRL, we further apply DEHRL in a popular video game called *MineCraft*, where the agent has much freedom to act and build.

In our settings, the agent has the first-person view via raw pixel input. At the beginning of each episode, the world is empty except one layer of *GRASS* blocks that can be broken. We allow the agent to play 1000 steps in each episode; then the world is reset. At each step, ten actions are available, i.e., moving towards four directions, rotating the view towards four directions, break and build a block[1], and jump. Due to space limit, more detailed settings and more visualizations about the agent and this game are provided in the supplementary material.

Since the typical use of DEHRL is based on the intrinsic reward only, the existing works (Tessler et al. 2017) that requires human-designed extrinsic reward signals to train subpolicies are not applicable to be used as baselines. Consequently, we compare the performance of DEHRL with three different numbers of levels in MineCraft to a framework with a random policy. Fig. 7 shows the building results, where we measure the performance by world complexity. As we can see, DEHRL builds more complex worlds than the random policy. Furthermore, with the increase of the number of levels, DEHRL tends to build more complex worlds.

The complexities of the worlds is quantified by *Valid Operation*, which is computed by the following equation:

$$Valid\ Operations = Valid\ Breaks + Valid\ Builds,$$

[1]Only one kind of block (i.e., *BRICK*) can be built, and any blocks except *STONE* can be broken.

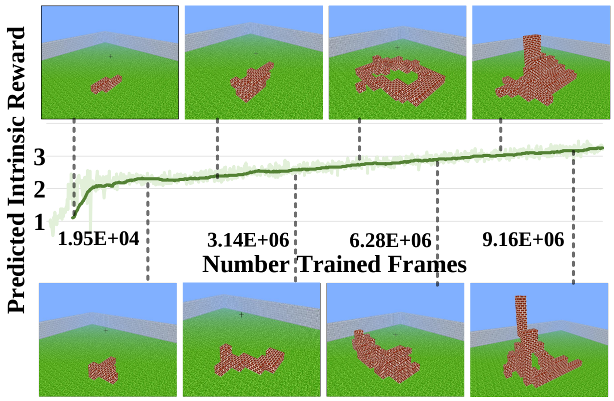

Figure 8: Predicted intrinsic reward.

where *Valid Builds* is the number of blocks that have been built and not brobroken at the end of an episode; *Valid Breaks* is the the number of blocks that are originally in the world have been broken. Consequently, blocks that are built but broken later will not be counted into *Valid Build* or *Valid Break*. The quantitative results in Fig. 7 are definitely consistent with our previous Intuitive feeling.

Finally, since the predicted intrinsic reward ($\hat{b}_t^l$) is an indication of the diversity of current subpolicies, we plot the averaged $\hat{b}_t^l$ over all levels and visualize the world built by DEHRL at the point in Fig. 8, so that the relationship between $\hat{b}_t^l$ and the built world is further illustrated. In summary, based on this preliminary exploration, we can assert that DEHRL has the potential to achieve very promising results in MineCraft, which proves the portability of DEHRL.

4 Related Work

Discovering diverse subpolicies in HRL. The diversity assumption is prevailing in the recent works of option discovery. Among them, SAC-LSP (Haarnoja et al. 2018) is the most recent work, but whether it can operate different layers at different temporal scales is an open problem. HiREPS (Daniel, Neumann, and Peters 2012) is also a popular approach, but it works in a top-down fashion. Thus, it is not clear that whether the above two methods can be applied to sparse extrinsic reward tasks. In contrast, SNN (Florensa, Duan, and Abbeel 2017) is designed to handle sparse extrinsic reward tasks, which achieves this diversity assumption explicitly by information-maximizing statistics. Besides, it is promising to apply SNN in HRL of multiple levels. However, SNN suffers from various failures when the possible future states are enormous, making it impractical on domains with large action space and unable to further learn higher-level subpolicies. Similar failure cases are observed in (Gregor, Rezende, and Wierstra 2016).

Extensible HRL. Recently, there has been some attempts in increasing the levels of HRL. Such works include MAXQ (Dietterich 2000), which requires completely searching the subtree of each subtask, leading to high computational time. In contrast, AMDP (Gopalan et al. 2017) explores only the relevant branches. However, AMDP concentrates on the planning problem. Deeper levels are also supported in (Silver and Ciosek 2012), but its scalability is not clear. DDO (Fox et al. 2017) and DDCO (Krishnan et al. 2017) discover higher-level subpolicies from demonstration trajectories. However, our work focuses on learning those purely end-to-end without human designed extrinsic reward. Studies from neuroscience (Rasmussen, Voelker, and Eliasmith 2017) also involve a modular structure that supports deeper-level HRL. However, there is no guarantee or verification on whether the structure can learn useful or diverse subpolicies.

Meta HRL. Neuroscience research (Solway et al. 2014) proposes that the optimal hierarchy is one that best facilitates adaptive behavior in face of new problems. Its idea is accomplished with verified scalability in MLSH (Frans et al. 2018), where meta HRL is proposed. However, MLSH keeps reinitializing the policy at top level, once the environment resets the goal. This brings several drawbacks, such as the inability to be extended to higher levels and requiring aux information from the environment about when the goal has been changed. In contrast, our method does not introduce either of the above restrictions. Regardless of the difference, we compare with MLSH in our experiments under the settings of MLSH where aux information on goal resetting is provided. As such, the meta HRL ability of our approach is investigated.

Improved exploration with predictive models. Since we introduce the transition model to generate intrinsic rewards, our method is also related to improving exploration with predictive models, typically introducing sample models (Fu, Co-Reyes, and Levine 2017), generative models (Held et al. 2017), or deterministic models (Pathak et al. 2017) as transition models to predict future states. However, the transition model in our DEHRL is introduced to encourage developing diverse subpolicies, while those in the above works are introduced to improve exploration. The idea in (Şimşek and Barto 2004) is to measure how much novelty a state introduces, thus called state novelty. In contrast, the idea in state-of-the-art (Pathak et al. 2017) is to measure the expectation of a transition, thus called transition novelty for distinction. Our method is compared with above state novelty (Şimşek and Barto 2004) and transition novelty (Pathak et al. 2017) in our experiments.

5 Summary and Outlook

We have proposed DEHRL towards building extensible HRL that learns useful subpolicies over multiple levels. We have shown that such subpolicies over multiple levels are useful to solve tasks with highly abstract goals, while the action space is primitive. We have also shown that such levels of subpolicies can better facilitate the learning of new tasks and produce more complex behavior when no extrinsic reward is provided. Finally, we also show the good portability of DEHRL using MineCraft. There are several interesting directions to explore for DEHRL in future. For example, though our framework can be easily extended, T^l and $\mathbb{A}^l$ at each level still need to be manually set. A promising direction is to develop algorithms that generate or dynamically adjust these settings. Furthermore, measuring the distance between states is another important direction to explore, where algorithms that better understand the states may lead to improvements of DEHRL. Finally, though promising results are shown on MineCraft, experiments in this section is still preliminary to foster more discussions. So an interesting future work is to explore the possibility to introduce extrinsic rewards, making the built world is well-structured or appreciated by a human.

References

Al-Shedivat, M.; Bansal, T.; Burda, Y.; Sutskever, I.; Mordatch, I.; and Abbeel, P. 2018. Continuous adaptation via meta-learning in nonstationary and competitive environments.

Bacon, P.-L.; Harb, J.; and Precup, D. 2017. The option-critic architecture. *AAAI*.

Daniel, C.; Neumann, G.; and Peters, J. 2012. Hierarchical relative entropy policy search. In *Artificial Intelligence and Statistics*.

Dietterich, T. G. 2000. Hierarchical reinforcement learning with the maxq value function decomposition. *JAIR*.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-agnostic meta-learning for fast adaptation of deep networks.

Florensa, C.; Duan, Y.; and Abbeel, P. 2017. Stochastic neural networks for hierarchical reinforcement learning. *ICLR*.

Fox, R.; Krishnan, S.; Stoica, I.; and Goldberg, K. 2017. Multi-level discovery of deep options. *arXiv preprint arXiv:1703.08294*.

Frans, K.; Ho, J.; Chen, X.; Abbeel, P.; and Schulman, J. 2018. Meta learning shared hierarchies. *ICLR*.

Fu, J.; Co-Reyes, J.; and Levine, S. 2017. Ex2: Exploration with exemplar models for deep reinforcement learning. In *NIPS*.

Gopalan, N.; desJardins, M.; Littman, M. L.; MacGlashan, J.; Squire, S.; Tellex, S.; Winder, J.; and Wong, L. L. 2017. Planning with abstract markov decision processes. In *ICAPS*.

Gregor, K.; Rezende, D. J.; and Wierstra, D. 2016. Variational intrinsic control. *ICLR Workshop*.

Haarnoja, T.; Hartikainen, K.; Abbeel, P.; and Levine, S. 2018. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*.

Heess, N.; Wayne, G.; Tassa, Y.; Lillicrap, T.; Riedmiller, M.; and Silver, D. 2016. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv:1610.05182*.

Held, D.; Geng, X.; Florensa, C.; and Abbeel, P. 2017. Automatic goal generation for reinforcement learning agents. *arXiv preprint arXiv:1705.06366*.

Kompella, V. R.; Stollenga, M.; Luciw, M.; and Schmidhuber, J. 2017. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*.

Krishnan, S.; Fox, R.; Stoica, I.; and Goldberg, K. 2017. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on Robot Learning*.

Lakshminarayanan, A. S.; Krishnamurthy, R.; Kumar, P.; and Ravindran, B. 2016. Option discovery in hierarchical reinforcement learning using spatio-temporal clustering. *arXiv preprint arXiv:1605.05359*.

Oh, J.; Guo, X.; Lee, H.; Lewis, R. L.; and Singh, S. 2015. Action-conditional video prediction using deep networks in atari games. In *NIPS*.

Parr, R., and Russell, S. J. 1998. Reinforcement learning with hierarchies of machines. In *NIPS*.

Pathak, D.; Agrawal, P.; Efros, A. A.; and Darrell, T. 2017. Curiosity-driven exploration by self-supervised prediction. In *ICML*.

Rasmussen, D.; Voelker, A.; and Eliasmith, C. 2017. A neural model of hierarchical reinforcement learning. *PloS one*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., and Ciosek, K. 2012. Compositional planning using optimal option models. *arXiv preprint arXiv:1206.6473*.

Şimşek, Ö., and Barto, A. G. 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *ICML*.

Solway, A.; Diuk, C.; Córdova, N.; Yee, D.; Barto, A. G.; Niv, Y.; and Botvinick, M. M. 2014. Optimal behavioral hierarchy. *PLoS computational biology*.

Sutton, R. S.; Precup, D.; and Singh, S. 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*.

Tessler, C.; Givony, S.; Zahavy, T.; Mankowitz, D. J.; and Mannor, S. 2017. A deep hierarchical approach to lifelong learning in minecraft. In *AAAI*.

Vezhnevets, A.; Mnih, V.; Osindero, S.; Graves, A.; Vinyals, O.; Agapiou, J.; et al. 2016. Strategic attentive writer for learning macro-actions. In *NIPS*.

Xu, D.; Nair, S.; Zhu, Y.; Gao, J.; Garg, A.; Fei-Fei, L.; and Savarese, S. 2017. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*.