Creativity of AI: Automatic Symbolic Option Discovery for Facilitating Deep Reinforcement Learning

Mu Jin[1], Zhihao Ma[1], Kebing Jin[1], Hankz Hankui Zhuo[1]*, Chen Chen[2], Chao Yu[1]

[1] School of Computer Science and Engineering, Sun Yat-Sen University
[2] Huawei Noah's Ark Lab
{jinm6, mazhh7, jinkb}@mail2.sysu.edu.cn, {zhuohank, yuchao3}@mail.sysu.edu.cn, chenchen9@huawei.com

Abstract

Despite of achieving great success in real life, Deep Reinforcement Learning (DRL) is still suffering from three critical issues, which are data efficiency, lack of the interpretability and transferability. Recent research shows that embedding symbolic knowledge into DRL is promising in addressing those challenges. Inspired by this, we introduce a novel deep reinforcement learning framework with symbolic options. This framework features a loop training procedure, which enables guiding the improvement of policy by planning with action models and symbolic options learned from interactive trajectories automatically. The learned symbolic options alleviate the dense requirement of expert domain knowledge and provide inherent interpretability of policies. Moreover, the transferability and data efficiency can be further improved by planning with the action models. To validate the effectiveness of this framework, we conduct experiments on two domains, Montezuma's Revenge and Office World, respectively. The results demonstrate the comparable performance, improved data efficiency, interpretability and transferability.

Introduction

Deep Reinforcement Learning (DRL) has achieved tremendous success in complex and high dimensional environments such as Go (Silver et al. 2016, 2017) and Atari Games (Mnih et al. 2015). It interacts with environments and improves its policy with the collected experience, by maximizing the long term reward. Recent criticism on DRL mostly focuses on low data efficiency, the lack of interpretability and transferability. The policy learnt from an environment often fails in another unseen environment. Due to the use of black-box neural networks for function approximation, the intrinsic lack of interpretability issue naturally raises in DRL, which disables the agent to explain its actions in a human-understandable way and earn people's trust in critical areas such as autonomous driving (Aradi 2020) and chemical engineering (Zhou, Li, and Zare 2017). Besides, DRL often requires a large amount of data to learn a satisfying policy in complex environments.

To alleviate these issues, researchers have put forward a variety of approaches. For transferability, Gamrian and Goldberg achieve substantially better sample efficiency and transfer behavior by separating the visual transfer task from the control policy (Gamrian and Goldberg 2019);

Ammanabrolu and Riedl explore the use of knowledge graphs as a representation of domain knowledge transfer for training text-adventure playing RL agents (Ammanabrolu and Riedl 2019). For interpretability, Jiang and Luo represent the policies in RL by first-order logic (Jiang and Luo 2019); Lyu et al. relates symbolic actions to options (Lyu et al. 2019); Hasanbeig et al. synthesises a human-interpretable automaton from trace data collected by exploring the environment (Hasanbeig et al. 2019).

For data efficiency, researchers investigate the use of learning a transition model(Chua et al. 2018; Nagabandi et al. 2018) or reward shaping methods(Hu et al. 2020, 2021). In addition to the above approaches, one promising way to solve these problems is to design a hierarchical decision making structure. Kulkarni et al. (Kulkarni et al. 2016) proposed Hierarchical Deep Reinforcement Learning (h-DRL) model to divide the MDP into two stages, associated with two agents. The first agent assigns a subgoal, which would be accomplished by the second agent. This approach successfully solves the sparse and delayed reward problem in the complicated Atari domain (e.g. Montezuma's Revenge).

Symbolic planning (Cimatti, Pistore, and Traverso 2008) is another effective approach enabling agent to make decisions in environments (Hanheide et al. 2017; Chen, Yang, and Chen 2016). It can be seen as a model-based approach, which relies on human experts to provide a symbolic transition model of the environment. The prior expert knowledge, represented in the form of symbols, can be utilized by logic-based approach such as PDDL (Gerevini 2020; Ghallab et al. 1998) or an action set solver CLINGO (Gebser, Kaufmann, and Schaub 2012) to generate a sequence of actions to reach goals. The provided symbolic model can provide inherent interpretability and improve data efficiency by planning. However, it is a great challenge to provide such symbolic model in complex and uncertain environments and thus the generality of symbolic planning is limited.

Researchers have investigated the combination of h-DRL and symbolic planning (Parr and Russell 1997; Ryan 2002; Hogg, Kuter, and Muñoz-Avila 2010; Leonetti, Iocchi, and Stone 2016; Yang et al. 2018; Lyu et al. 2019; Illanes et al. 2020; Sarathy et al. 2020). In this structure, the original MDP is divided into two levels. The higher level utilizes a symbolic

*Corresponding author

planner with a given action model to select an option and the lower-level interacts with the environment to accomplish the assigned option. This two-level structure helps alleviating the sparse reward issue, improves sampling efficiency and provides interpretability by the given symbolic action model. Specifically, this structure introduces prior knowledge by extracting semantic information from high dimensional states and manually designing symbolic transition model. However, if the action models are not available or incomplete in complex environments, it is likely that the combination would fail. The idea of learning action models in RL has emerged (Zhuo and Kambhampati 2017; Yang, Wu, and Jiang 2007; Ng and Petrick 2019; Martínez et al. 2016; James, Rosman, and Konidaris 2020) and is an important direction in the field of symbolic planning. However, they directly use a provided action model to plan an executable solution instead of learning an action model. Therefore, this approach requires a detailed symbolic representation of the environments and it is difficult to be applied in complex environments. Although another method (Sarathy et al. 2020) learns the action models automatically, they still need to manually define a major part of the models in advance. Besides, the planning goal in this approach is kept unchanged while is dynamically adapted to maximize the external reward in our framework.

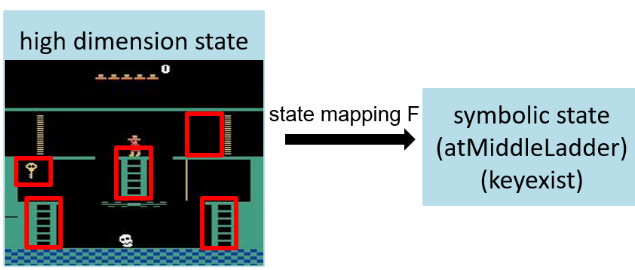


Figure 1: State Mapping Function in Montezuma's Revenge

Inspired by this, we introduce a novel framework named Symbolic Options for Reinforcement Learning (SORL) to tackle these problems mentioned above. We assume that there exists a function F, mapping high dimensional states to symbolic states and enabling us to learn symbolic action models and options. As shown in Figure 1, we extract the position of the man in red and the key from the high-dimensional state to obtain the corresponding symbolic state. When the agent walks from the middle ladder to the right ladder, the key still exists and the environment does not give any feedback (e.g. zero reward). This can be seen as a symbolic transition and we will generate the corresponding action model as shown in Figure 2. Then, we use a planner with the learned action models and a planning goal as inputs to generate a plan and use it to instruct the learning of agent.

Based on the assumption, SORL features a two-level structure, of which the higher level are a symbolic planner and a meta-controller, and the lower level is an RL agent interacting with the environment. The higher level utilizes the collected trajectories from the lower level to learn action models and symbolic options with minimum human knowledge. After that, the meta-controller chooses an option according to the

***action act*0:**
pre^+: (atMiddleLadder)(keyexist)
pre^-: $\emptyset$
eff^+: (atRightLadder) (increase (reward) 0)
eff^-: (atMiddleLadder)

Figure 2: An Action model in Montezuma's Revenge

plan generated from the planner with the learned action models and assigns it to the lower level. By interacting with the environments, the lower level learns a policy to reach the assigned option and sends the collected experience to the higher level. This cross-fertilization structure not only helps alleviating the sparse and delayed reward problem but also improves the interpretability and data efficiency. We summarize our contribution as below:

- Our work is the first one to learn action and option models automatically without being told any knowledge of these models and simultaneously learn RL policies.
- We propose a symbolic reinforcement learning framework capable of providing interpretability, improved data-efficiency and transferability.
- The symbolic option learned by SORL is more general, which can correspond to more than one action model.

Preliminaries

In this chapter, we establish relevant notation and briefly introduce key aspects of symbolic planning and reinforcement learning.

Symbolic Planning with PDDL

In PDDL language, states are represented as set of propositions and we call it symbolic states throughout the paper to distinguish them from states in RL. Propositions represent the properties of the world and in the symbolic state s, proposition $p \in s$ if p is true otherwise $(not\ p) \in s$. An action description called action model is a tuple $(name, pre^+, pre^-, eff^+, eff^-)$, where $name$ is the name of the action, (pre^+, pre^-) are the preconditions and (eff^+, eff^-) are the effects. As shown in Fig.2, the action model describes that when the agent walks from the middle ladder to the right ladder , the key keeps still and the reward remains unchanged. If $pre^+ \subset s$ and $s \cap pre^- = \emptyset$, then we can execute action a and obtain the next state $s' = ((s - eff^-) \cup eff^+)$.

The planning domain $D = (P, A)$ includes the proposition set P and the action set A, which describe the state space and the action space, respectively. A tuple (s, a, s') describes a symbolic transition from state s to state s' after executing action a. We define a planning problem denoted as a triple (I, P, A, G), of which I is an initial state and G is a goal state. The solution to this problem is called a plan π, which is a sequence of actions. After executing the plan, we can obtain a symbolic transition trace from I to G. To obtain such a plan with the maximum reward, we use a planner called Metric-FF

(Hoffmann 2002), which can handle planning problems with continuous metrics.

Reinforcement Learning

A Markov Decision Process (MDP) is defined as the tuple $(\widetilde{S}, \widetilde{A}, P^{\widetilde{a}}_{\widetilde{s}\widetilde{s}'}, r^{\widetilde{a}}_{\widetilde{s}}, \gamma)$ where $\widetilde{S}$ and $\widetilde{A}$ denote the state space and action space, respectively, $P^{\widetilde{a}}_{\widetilde{s}\widetilde{s}'}$ provides the transition probability of moving from state $\widetilde{s} \in \widetilde{S}$ to state $\widetilde{s'} \in \widetilde{S}$ after taking action $\widetilde{a} \in \widetilde{A}$, $r^{\widetilde{a}}_{\widetilde{s}}$ is the immediate reward obtained after performing action $\widetilde{a}$ at state $\widetilde{s}$ and $\gamma \in [0, 1)$ is a discount factor. The task of RL is to obtain a policy $\pi : \widetilde{S} \rightarrow \widetilde{A}$ that maximizes the expected return $V_\pi(\widetilde{s}) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t \mid \widetilde{s}_0 = \widetilde{s}]$ where r_t is the reward at time step t received by following π from state $\widetilde{s}_0 = \widetilde{s}$. The state-action value function is defined as follows: $Q_\pi(\widetilde{s}, \widetilde{a}) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t \mid \widetilde{s}_0 = \widetilde{s}, \widetilde{a}_0 = \widetilde{a}]$.

Option Framework

Hierarchical Reinforcement Learning (HRL) extends RL with temporally macro actions that represent high-level behaviors. The option framework (Sutton, Precup, and Singh 1999) models macro actions as options. In particular, an option o is defined as $(I_o(s), \pi_o(s), \beta_o(s))$, where initiation condition $I_o(s)$ determines whether the option o can be executed at state s, termination condition $\beta_o(s)$ determines whether option execution terminates at state s and $\pi_o(s)$ is a policy mapping state s to a low-level action. In this framework, an agent learns to choose an optimal option to be executed in the higher level, i.e. meta controller level and the lower level, i.e. controller level learns optimal policies to reach the option. An explicit assumption is that the set of options is predefined by human experts.

The SORL Framework

We define the reinforcement learning environment by a tuple $(I, G, P, A, F, \widetilde{S}, \widetilde{A}, \widetilde{R}, \widetilde{P}, \gamma)$ and divide it into three parts:

- First, we define a high-level symbolic planning problem by (I, G, P, A), where I is an initial state, and G is a goal state. P is a set of propositions represented by planning language PDDL with prior knowledge and it is used to describe symbolic states S, where $S \subseteq 2^P$. A is a set of action models that $S \times A \rightarrow S$ transfer a symbolic state to another. Each action model is learned by meta-controller through symbolic state pairs.
- Second, we define a state mapping function $F : \widetilde{S} \rightarrow 2^P$ mapping a high dimension state $\widetilde{s}$ to a symbolic state s.
- At last, we define an underlying decision-making problem by an MDP tuple $(\widetilde{S}, \widetilde{A}, \widetilde{R}, \widetilde{P}, \gamma)$. We denote a symbolic action and state as a and s respectively, while the primitive action and state as $\widetilde{a}$ and $\widetilde{s}$. Noted that $\widetilde{a}$ and $\widetilde{s}$ are gained from interacting with the environment.

Taking the Figure 1 as an example, in the game of the Montezuma's Revenge, the high level dimension state is the picture of the game scene and the symbolic state is composed of the propositions describing the location of the agent and the existence of the key.

This framework aims to learn action models, which can be utilized by logic-based planner to generate a sequence of options and achieve the maximal cumulative reward. As shown in the Figure 3, the SORL framework includes three components: (1) a planner for generating plans, (2) a meta-controller for generating action models, goals and choosing the goal option, and (3) an option set for interacting with the environment. The meta-controller first takes the symbolic state pairs and their external rewards as inputs and outputs action models and a goal. Noted that the state pairs set are empty in the beginning. Then the planner takes the action models as input and computes a plan. Next, the meta-controller receives the plan from the planner and chooses an option. Each option in option set can be regarded as an agent. The chosen agent keeps interacting with the environment until accomplishing the option or reaching the maximal steps, and the low-level state traces will be transformed into symbolic state pairs by the label function F and sent back to meta controller. The meta-controller continues learning action models and symbolic options from gained symbolic state pairs and external rewards. We repeat these procedures $num\_episodes$ times. With the proceeding of learning, our approach keeps updating action models and planning goals and the planner is able to generate plans achieving better rewards.

Option Set

Symbolic Option In this paper, we propose a novel option framework which is called *symbolic option*. A symbolic option is computed by symbolic state pairs gained from trajectories instead of manual setting in advance, requiring less prior knowledge in our approach. We define a symbolic option by $so = (pre, \pi, eff)$. pi is a low-level policy. pre is an union of preconditions, including pre^+ and pre^-. It's created and updated when the meta-controller generates action models. Similarly, eff is composed of eff^+ and eff^-, describing the effects of the symbolic option. As for a symbolic option so and a high-dimension state $\widetilde{s}$, we compute initiation condition $I_{so}(\widetilde{s})$ by Equation (1) and termination condition $\beta_{so}(\widetilde{s})$ by Equation (2). A symbolic option can be executed based on $\widetilde{s}$ only if $I_{so}(\widetilde{s}) = True$. Similarly, it terminates only if $\beta_{so}(\widetilde{s}) = True$.

$$I_{so}(\widetilde{s}) = \begin{cases} True & pre^+ \subset F(\widetilde{s}), F(\widetilde{s}) \cap pre^- = \emptyset \\ False & otherwise \end{cases} \tag{1}$$

$$\beta_{so}(\widetilde{s}) = \begin{cases} True & eff^+ \subset F(\widetilde{s}), eff^- \cap F(\widetilde{s}) = \emptyset \\ False & otherwise \end{cases} \tag{2}$$

It's noted that the inherent symbolic propositions of our symbolic option provide better interpretability compared to those approaches based on black-box neural networks. In terms of the low-level policy pi, it can be learned by interacting with the environments with the intrinsic rewards given by the meta-controller.

Global Option At the beginning of our algorithm, the option set contains no symbolic options but a global option $o_G = (I_G(s), \pi, \beta_G(s))$, where $I_G(\widetilde{s}) \equiv True$, $\beta_G(\widetilde{s}) = True$ if symbolic state changes and $\pi = random(\widetilde{A})$. We use $random(\widetilde{A})$ to indicate that the global option each time

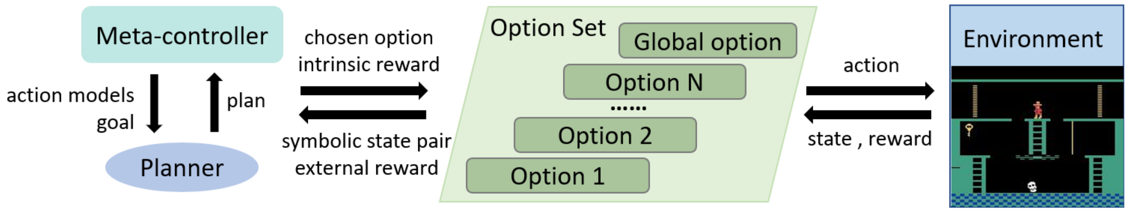


Figure 3: The SORL framework

chooses a random action $\widetilde{a} \in \widetilde{A}$. Intuitively, the global option is available for any state and it keeps randomly exploring until the symbolic state changes. Hence, in order to discover new action models, the meta-controller outputs the global option when the plan is empty or all action models in the plan has been executed.

Symbolic State Pair and External Reward Given an option o_j under state $\widetilde{s}$, the lower level policy interacts with the environment and output a pair of symbolic states (s_1, s_2) and external reward r_e, denoted by $(s_1, s_2), r_e = ExecuteOption(\widetilde{s}, o_j)$. If the chosen option o_j isn't available for $\widetilde{s}$, i.e., $I_j(\widetilde{s}) = False$, both of the output pair and the reward are $None$. Otherwise, if the chosen option o_j is able to be executed, we let $s_1 = F(\widetilde{s})$ and the policy π_j first chooses an action $\widetilde{a}$ and we can obtain the next state $\widetilde{s'}$ and its reward $\widetilde{r}$ by interacting with the environments. Then the controller adds experience $(\widetilde{s}, \widetilde{a}, \widetilde{s'}, \widetilde{r})$ to the o_j's replay buffer. We keep executing action by following the low-level policy and update the states and rewards until until $\beta_j(\widetilde{s'}) = True$ or reaching the maximum steps, which means the option has been successfully executed or not. Finally, if the option o_j is successfully executed, we set the output symbolic state pair as (s_1, s_2) of which $s_2 = F(\widetilde{s'})$ and the external reward r_e be the accumulated sum of the environment rewards during interacting.

Meta-controller

In this section, we introduce our Meta-controller in detail. Meta-controller takes symbolic state pairs and their external rewards as input, and first generates action models and a planning goal and then chooses a an option according to the plan from planner.

Action Model Given symbolic state pairs and their rewards, meta-controller generates action models by $A, F_{A,O}, O = GenerateActionModels(R, O, sr)$. The function indicates it takes a dictionary R,an option set O and the success ratio set sr as inputs, and outputs a generated action set A, a mapping function $F_{A,O}$ and the updated option set O. Dictionary R includes mappings from a symbolic state pair to its external rewards. $F_{A,O}$ transfers action models to options. The success ratio set sr records the percentage of action models successful executed each 100 times.

As for a symbolic state pair $(s_1, s_2)_i \in R$, we can get a corresponding action model $a_i = (name, pre^+, pre^-, eff^+, eff^-)$. Noted that the action model and a symbolic state pair is a one-to-one match. Given a state pair $(s_1, s_2)_i$, the $name$ of a_i is the index of action models, denoted by act_i, and $pre^+ = \{p|p \in s_1\}$, $pre^- = \{p|p \notin s_1\}$. Next we let $eff^+ = s_2 - s_1$ and $eff^- = s_1 - s_2$, where $a - b$ is a set subtraction indicating set a subtracts the intersection of set a and set b. In order to generate a plan gaining a maximum reward, we use the metric constant $quality$ to denote the cumulative reward of the plan and add the proposition "$(increase \quad (quality) \quad \rho_i)$" into eff^+. Finally, we get an action which is called act_i, and we define the gained reward of act_i by ρ_i. To encourage the planner to generate a plan including the exploring action model, the reward ρ_i is composed of mean external reward and exploration reward, computed by Equation (3), where $R[(s_1, s_2)_i]$ is the external rewards list and r_E is the exploration rewards. The exploration rewards is computed by Equation (4), where c is a constant and $sr[i]$ is the success rate of act_i, which means exploration reward decreases as success rate increases.

$$\rho_i = mean(R[(s_1, s_2)_i]) + r_E \tag{3}$$

$$r_E = \begin{cases} c(1 - sr[i]) & act_i \; is \; being \; explored \\ 0 & otherwise \end{cases} \tag{4}$$

If there exists a symbolic option $o_j = (pre_j, \pi_j, eff_j)$ where $eff_j = eff$ after we attain an action model, we update pre_j^+ to a union of pre^+ and pre_j^+, and pre_j^- to a union of pre^- and pre_j^-. Otherwise, we create a new symbolic option $o_j = (pre, \pi_j, eff)$ and add it to the option set O. At last, we set the mapping function $F_{A,O}(act_i) = o_j$. During the exploration, we explore each action model sequentially, in other words, we repeat exploring act_i until the success rate of act_0 to act_{i-1} is higher than the threshold.

Planning Goal Next Meta-controller outputs a goal to guide planner, aiming at generating a plan with a maximal reward. The goal is a label function $quality > q$, where q is the cumulative external rewards of the plan gained in the last episode. Intuitively, the function constrains the planner to compute a plan with a largest reward compared with the past plans.

Chosen Option and Intrinsic Reward After the planner generates a plan $\Pi = (a_1, a_2, \dots, a_n)$, as for each action model a_i, the meta-controller selects a symbolic option from option set by $o_j = F_{A,O}(act_i)$, and we can get an series of options $(o_0, o_1, \dots, o_n)$. If all action models in Π successfully finish, which indicates the chosen symbolic options are

Algorithm 1: Planning and Learning algorithm for SORL

Input: proposition set P, state mapping function F, success ratio threshold λ

Initialization: option set $O \leftarrow \{o_G\}$, action models set $A \leftarrow \emptyset$, symbolic state pairs' external rewards dictionary $R \leftarrow \emptyset$, action models' success ratio set $sr \leftarrow \emptyset$, plan $\Pi_0 \leftarrow \emptyset$, $q \leftarrow 0$
for t=1,2, …, $num\_episodes$ **do**
  Initialize game, get start state $\widetilde{s_0}$, $I \leftarrow F(\widetilde{s_0})$, $\Pi^* \leftarrow \Pi_{t-1}$
  $A, F_{A,O}, O \leftarrow GenerateActionModels(R, O, sr)$
  $G \leftarrow (quality > q)$
  $\Pi_t \leftarrow metricFF.solve(I, P, A, G)$
  if $\Pi_t = \emptyset$ **then** $\Pi_t \leftarrow \Pi^*$
  $q \leftarrow 0$
  for $a_i \in \Pi_t$ **do**
    $o_j \leftarrow F_{A,O}[i]$, obtain current state $\widetilde{s}$
    $(s_1, s_2), r_e \leftarrow ExecuteOption(\widetilde{s}, o_j)$
    append r_e into $R[(s_1, s_2)]$, $q \leftarrow q + r_e$
  end for
  while env isn't terminal **do**
    obtain current state $\widetilde{s}$
    $(s_1, s_2), r_e \leftarrow ExecuteOption(\widetilde{s}, o_G)$
    if (s_1, s_2) not in R **then**
      $R[(s_1, s_2)] \leftarrow list(r_e)$
    else
      append r_e into $R[(s_1, s_2)]$
    end if
  end while
  train options in O and calculate sr
end for

executed sequentially and termination conditions are satisfied , then the meta-controller would choose the global option o_G to explore the environment thoroughly. For each option $o_i = (pre_i, \pi_i, \beta_i)$, we refer to (Lyu et al. 2019) to design intrinsic rewards:

$$r_i(\widetilde{s}) = \begin{cases} \phi & \beta_i(\widetilde{s}) = True \\ r & otherwise \end{cases} \tag{5}$$

where ϕ is a constant and r is the reward gained from the environments when reach state $\widetilde{s}$.

Planning and Learning

As shown in Algorithm 1, we firstly initialize an option set O only including o_G, an empty action model set A, an empty dictionary mapping symbolic state pairs to their external rewards R, an empty action models success ratio set sr and an empty plan Π_0. When an episode t begins, we first get a start state $\widetilde{s_0}$ from environment. Then we compute the symbolic initial state I by F and record the best plan Π^*, which is the plan generate in the last episode. Then meta-controller updates action models A, symbolic options set O, their mapping function $F_{A,O}$ and the planning goal G. Given current action models A and planning goal G, Metric-FF planner (Hoffmann 2002) generates a new plan Π_t whose quality is higher than the last plan Π_{t-1}. If Π_t is empty, which indicates Metric-FF couldn't find a solution to solve the problem, we let $\Pi_t = \Pi^*$.

As for each action model a_i in plan Π_t, meta-controller chooses a corresponding symbolic option o_j by $F_{A,O}$. Then the controller interacts with environment by performing Deep Q-Learning, executes the action chosen by o_j's inner policy and stores experience into o_j's replay buffer until o_j terminates. After that, we get o_j's initial symbolic state s_1 and a terminate symbolic state s_2 and an extrinsic reward r_e. In this way, we compute symbolic state pairs and their extrinsic rewards one by one and record these mappings by a dictionary R. Finally, quality q of plan Π_t is defined as the accumulated sum of extrinsic rewards.

If the environment isn't finished after executing Π_t, the meta-controller chooses the global option o_G to explore new symbolic states pairs in the environment. o_G stops exploring when the computed symbolic state changes and we calculate a symbolic state pair (s_1, s_2) and its external reward r_e. If (s_1, s_2) is a new symbolic state pair, we add it into R. This process repeats until the environment is terminated. Finally, when an episode ends, we train options in O and calculate success ratio for each action model.

Experiment

In this section, we evaluate our approach on two domains, Office World and Montezuma's Revenge in terms of data-efficiency, interpretability and transferability.

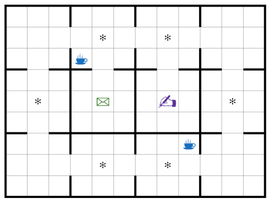
Office World

We first evaluate our approach on the Office World (Icarte et al. 2018) which is a simple multitask environment. In this environment, being initialized at a random location, the agent can move towards one of the four cardinal directions. Actions are valid only if the movement does not go through a wall. The agent can pick up cups of coffee or mails when it reaches the cell marked with blue cups or green envelops, respectively. He can deliver coffee or mail to the office by reaching the cell marked with a purple hand. The symbol $*$ means the place where the agent can not stay or reach.

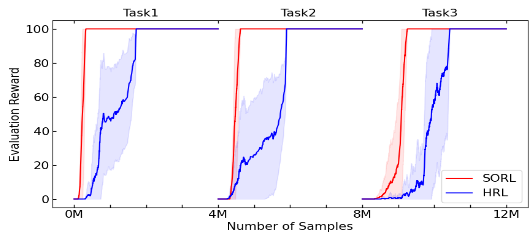
Setup In this environment, the start location of the agent is randomly initialized at every episode. The agent is required to finish three tasks. The first and the second are to deliver a cup of coffee or a piece of mail to the office while the third is to hand both objects to the office. We compared SORL to h-DQN, a goal based h-DRL approach (Kulkarni et al. 2016). Since the state and action space are finite, we choose to implement these two approaches with q-table in both high and low levels.

Results We evaluate our approaches in terms of data-efficiency, interpretability and transferability.

- **Data-efficiency** In order to validate the data-efficiency, we train these two approaches in the three tasks and compare the corresponding performance at the same interaction steps. To demonstrate the transferability, we train the agent in task 3 along with the options learned in tasks 1 and 2. To implement our approach, we design the propo-



(a) Office World



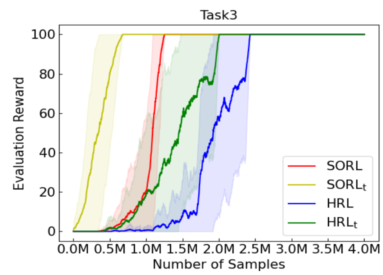
(b) Learning Curve

task	proposition
task1	(haveCoffee), (deliveredCoffee)
task2	(haveEmail), (deliverdEmail)
task3	(haveCoffee), (deliveredCoffee) (haveEmail), (deliverdEmail)

(c) Propositions

task	action	description	option	effect
1	act0	get coffee	opt0	(haveCoffee)
	act1	deliver coffee	opt1	(not (haveCoffee)) (deliveredCoffee)
2	act0	get mail	opt2	(haveMail)
	act1	deliver mail	opt3	(not (haveMail)) (deliveredMail)
3	act0	get coffee	opt0	(haveCoffee)
	act1	get mail	opt1	(haveMail)
	act2	deliver coffee and mail	opt4	(not (haveCoffee)) (not (haveMail)) (deliveredCoffee) (deliveredMail)

(d) Learned Action Models and Options



(e) Learning Curve

Figure 4: Experimental Results in the Office World

sitions as shown in Fig. 4(c). As shown in Fig.4(b) , from Task1 to Task3, SORL can get rewards faster than HRL.

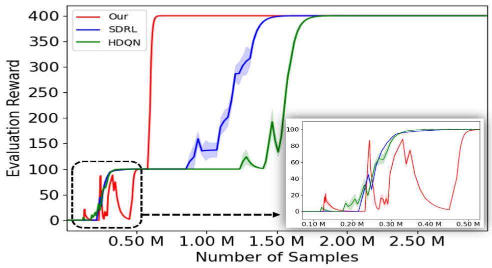
- **Interpretability** Fig.4(d) shows action models and symbolic options learned in each task. Those action models describe the reason of making decisions at each step in a human understandable way. For example, we can explicitly know $act1$ in task1 can be executed when the agent gets coffee and does not deliver it to the office, and the agent would deliver the coffee to the office and get a reward of 100 when $act1$ is executed.
- **Transferability** By utilizing the options learned in tasks 1 and 2, we test the tranferability of SORL and H-DQN in Task 3 and denote them as $SORL_t$ and HRL_t. As shown in Fig4(c), the performance of SORL and HRL is improved when transferring the learned knowledge. It verifies that compared to SORL, the converging speed of $SORL_t$ improves dramatically with only half of samples. We conjecture that the SORL is able to transfer the learned knowledge into other unseen environments.

Montezuma's Revenge

Montezuma's Revenge is an Atari game with sparse and delayed rewards. It requires the player to navigate through several rooms while collecting treasures. We conduct our experiments based on the first room shown in Fig.5(a). In this room, the player only obtains positive rewards when it fetches the key (+100) or opens a door (+300). Otherwise, the player would not receive any reward signal. The optimal solution is to climb down the ladders to obtain the key, then return back to the platform and open a door, resulting in a maximum reward (+400).

Setup We compare our approach with HRL (Kulkarni et al. 2016) and SDRL (Lyu et al. 2019) as baselines, where SDRL is an approach that combines symbolic planning and RL with excellent results in complex environments with sparse rewards. SORL can automatically learn the action models while they are pre-defined by experts in SDRL. Besides, the option model can correspond to multiple action models in SORL while one in SDRL. We implement these approaches under an option-based h-DRL framework. In terms of the low level, we follow the network architecture used in (Kulkarni et al. 2016) and train this network with double-Q learning (van Hasselt, Guez, and Silver 2016) and prioritized experience replay (Schaul et al. 2016). Besides, both SORL and SDRL use a planner to generate high level policy while HRL utilizes a neural network. The intrinsic reward follows 5 with $\phi = 100$. The maximum steps in an episode and the threshold of success rate are set to be 500 and 0.95, respectively. To describe the environment, we abstract four local propositions (e.g., MiddleLadder, RightDoor, LeftLadder and RightLadder) and an object (Key).

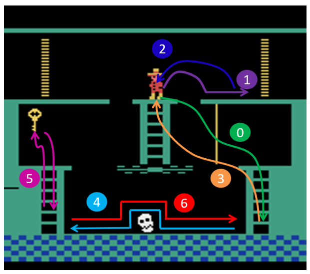
Results We present the experimental results in Fig.5. It is evident that SORL can achieve the maximum reward (+400) in 0.7M samples while both SDRL and HDQN need more than 1.5M samples, indicating the superior data-efficiency of SORL. However, to pick up the key (reward +100), SORL needs to interact with the environment with more than 0.3M steps, at which SDRL and HDQN fall into the local optimum. This is because SORL randomly explores symbolic options and it is easier to find options closer to the starting point. After finding these options, SORL would train them sequentially instead of directly learning options on the path of getting the key. One option model corresponds to one action model in SDRL while several action models in SORL. The ability of reusing the learned symbolic options enables SORL to converge faster than SDRL. Take Fig.5(c) as an example,



(a) Learning Curve

act		action model
act1	pre	(atMiddleLadder)(keyexist)
	eff	(not (atMiddleLadder))(atRightLadder) (increase (quality) 0)
act5	pre	(atLeftLadder) (keyexist)
	eff	(not (keyexist)) (increase (quality) 100)
act10	pre	(atMiddleLadder) (not (keyexist))
	eff	(not (atMiddleLadder)) (atRightDoor) (increase (quality) 300)

(b) Examples of Learned Action Models



(c) learned Symbolic Options

action	description	option
act0	move from MiddleLadder to RightLadder without Key	opt0
act1	move from MiddleLadder to RightDoor without Key	opt1
act2	move from RightDoor to MiddleLadder without Key	opt2
act3	move from RightLadder to MiddleLadder without Key	opt3
act4	move from RightLadder to LeftLadder without Key	opt4
act5	pick up the Key	opt5
act6	move from LeftLadder to RightLadder without Key	opt6
act7	move from LeftLadder to RightLadder with Key	opt6
act8	move from RightLadder to MiddleLadder with Key	opt3
act9	move from RightLadder to LeftLadder with Key	opt4
act10	move from MiddleLadder to RightDoor with Key	opt1

(d) Learned Action Models

Figure 5: Experimental Results in Montezuma's Revenge

the $opt1$ representing the move from middle ladder to right door, is firstly trained at the beginning when the player does not get the key. After the player picks up the key, SORL only needs a small amount of data to fine-tune $opt1$ when the player moves from the middle ladder to right door with a key. However, both SDRL and HDQN start training the options after the player moves to the middle ladder with a key, consuming more interaction resources. Different from option-based HDQN and SDRL, SORL can learn the initial and termination condition of symbolic options automatically. The action models used in SDRL need to be constructed by human in advance while they are learned from the trajectories in SORL, saving labour resources. We present some of the learned action models in Fig.5(b) and the effects of symbolic options in Fig.5(c). Fig. 5(b) describes the preconditions and effects of each action model and we can see that if the player is at LeftLadder and the key exists, then the player can obtain a key and reward (+100) by executing action5. Fig. 5(c) shows the learned options in SORL and the order of options actually does not match the optimal order because SORL randomly explore the environment and options 0-3 are easier to learn. We describe the meaning of all learned action models and their corresponding options in Fig. 5(d). It is easy to see that act7 to act10 correspond to the options explored before, so these options can be reused to improve the data-efficiency.

Conclusions

In this paper, we propose a novel framework SORL which can automatically learn action models and symbolic options from the trajectories and the symbolic planner can instruct RL to explore efficiently in environments with sparse and delayed rewards. Compared with other approaches, the experimental results demonstrate the better sampling efficiency of our approach. Moreover, SORL requires less prior knowledge and provides interpretability and transferability by the learned action models and symbolic options. In the future, it would be interesting to investigate possibility of learning more expressive planning models, such as learning Hierarchical Task Networks (Zhuo and Yang 2014) and PDDL models (Zhuo et al. 2010), as well as different learning mechanisms, such as transfer learning (Zhuo and Yang 2014; Shen et al. 2020).

Acknowledgement

This research was funded by the National Natural Science Foundation of China (Grant No. 62076263), Guangdong Natural Science Funds for Distinguished Young Scholar (Grant No. 2017A030306028), Guangdong special branch plans young talent with scientific and technological innovation (Grant No. 2017TQ04X866), Pearl River Science and Technology New Star of Guangzhou and Guangdong Province Key Laboratory of Big Data Analysis and Processing.

References

Ammanabrolu, P.; and Riedl, M. 2019. Transfer in Deep Reinforcement Learning Using Knowledge Graphs. In Ustalov, D.; Somasundaran, S.; Jansen, P.; Glavas, G.; Riedl, M.; Surdeanu, M.; and Vazirgiannis, M., eds., *Proceedings of the Thirteenth Workshop on Graph-Based Methods for Natural Language Processing, TextGraphs@EMNLP 2019, Hong Kong, November 4, 2019*, 1–10. Association for Computational Linguistics.

Aradi, S. 2020. Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles. *CoRR*, abs/2001.11231.

Chen, K.; Yang, F.; and Chen, X. 2016. Planning with Task-Oriented Knowledge Acquisition for a Service Robot. In Kambhampati, S., ed., *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, 812–818. IJCAI/AAAI Press.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*.

Cimatti, A.; Pistore, M.; and Traverso, P. 2008. Automated Planning. In van Harmelen, F.; Lifschitz, V.; and Porter, B. W., eds., *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, 841–867. Elsevier.

Gamrian, S.; and Goldberg, Y. 2019. Transfer Learning for Related Reinforcement Learning Tasks via Image-to-Image Translation. In Chaudhuri, K.; and Salakhutdinov, R., eds., *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, 2063–2072. PMLR.

Gebser, M.; Kaufmann, B.; and Schaub, T. 2012. Conflict-driven answer set solving: From theory to practice. *Artif. Intell.*, 187: 52–89.

Gerevini, A. E. 2020. An Introduction to the Planning Domain Definition Language (PDDL): Book review. *Artif. Intell.*, 280: 103221.

Ghallab, M.; Knoblock, C.; Wilkins, D.; Barrett, A.; Christianson, D.; Friedman, M.; Kwok, C.; Golden, K.; Penberthy, S.; Smith, D.; Sun, Y.; and Weld, D. 1998. PDDL - The Planning Domain Definition Language.

Hanheide, M.; Göbelbecker, M.; Horn, G. S.; Pronobis, A.; Sjöö, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janícek, M.; Zender, H.; Kruijff, G. M.; Hawes, N.; and Wyatt, J. L. 2017. Robot task planning and explanation in open and uncertain worlds. *Artif. Intell.*, 247: 119–150.

Hasanbeig, M.; Jeppu, N. Y.; Abate, A.; Melham, T.; and Kroening, D. 2019. DeepSynth: Automata Synthesis for Automatic Task Segmentation in Deep Reinforcement Learning. *arXiv preprint arXiv:1911.10244*.

Hoffmann, J. 2002. Extending FF to Numerical State Variables. In van Harmelen, F., ed., *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002, Lyon, France, Jusly 2002*, 571–575. IOS Press.

Hogg, C.; Kuter, U.; and Muñoz-Avila, H. 2010. Learning Methods to Generate Good Plans: Integrating HTN Learning and Reinforcement Learning. In Fox, M.; and Poole, D., eds., *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press.

Hu, Y.; Hua, Y.; Liu, W.; and Zhu, J. 2021. Reward Shaping Based Federated Reinforcement Learning. *IEEE Access*, 9: 67259–67267.

Hu, Y.; Wang, W.; Jia, H.; Wang, Y.; Chen, Y.; Hao, J.; Wu, F.; and Fan, C. 2020. Learning to Utilize Shaping Rewards: A New Approach of Reward Shaping. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Icarte, R. T.; Klassen, T. Q.; Valenzano, R. A.; and McIlraith, S. A. 2018. Advice-Based Exploration in Model-Based Reinforcement Learning. In Bagheri, E.; and Cheung, J. C. K., eds., *Advances in Artificial Intelligence - 31st Canadian Conference on Artificial Intelligence, Canadian AI 2018, Toronto, ON, Canada, May 8-11, 2018, Proceedings*, volume 10832 of *Lecture Notes in Computer Science*, 72–83. Springer.

Illanes, L.; Yan, X.; Icarte, R. T.; and McIlraith, S. A. 2020. Symbolic Plans as High-Level Instructions for Reinforcement Learning. In Beck, J. C.; Buffet, O.; Hoffmann, J.; Karpas, E.; and Sohrabi, S., eds., *Proceedings of ICAPS*, 540–550. AAAI Press.

James, S.; Rosman, B.; and Konidaris, G. 2020. Learning Portable Representations for High-Level Planning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, 4682–4691. PMLR.

Jiang, Z.; and Luo, S. 2019. Neural logic reinforcement learning. In *International Conference on Machine Learning*, 3110–3119. PMLR.

Kulkarni, T. D.; Narasimhan, K.; Saeedi, A.; and Tenenbaum, J. 2016. Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation. In Lee, D. D.; Sugiyama, M.; von Luxburg, U.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, 3675–3683.

Leonetti, M.; Iocchi, L.; and Stone, P. 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. *Artif. Intell.*, 241: 103–130.

Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: Interpretable and Data-Efficient Deep Reinforcement Learning Leveraging Symbolic Planning. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, 2970–2977. AAAI Press.

Martínez, D.; Alenyà, G.; Torras, C.; Ribeiro, T.; and Inoue, K. 2016. Learning Relational Dynamics of Stochastic Domains for Planning. In Coles, A. J.; Coles, A.; Edelkamp, S.; Magazzeni, D.; and Sanner, S., eds., *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016*, 235–243. AAAI Press.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Nagabandi, A.; Kahn, G.; Fearing, R. S.; and Levine, S. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7559–7566. IEEE.

Ng, J. H. A.; and Petrick, R. P. A. 2019. Incremental Learning of Planning Actions in Model-Based Reinforcement Learning. In Kraus, S., ed., *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, 3195–3201. ijcai.org.

Parr, R.; and Russell, S. J. 1997. Reinforcement Learning with Hierarchies of Machines. In Jordan, M. I.; Kearns, M. J.; and Solla, S. A., eds., *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, 1043–1049. The MIT Press.

Ryan, M. R. K. 2002. Using Abstract Models of Behaviours to Automatically Generate Reinforcement Learning Hierarchies. In Sammut, C.; and Hoffmann, A. G., eds., *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, 522–529. Morgan Kaufmann.

Sarathy, V.; Kasenberg, D.; Goel, S.; Sinapov, J.; and Scheutz, M. 2020. SPOTTER: Extending Symbolic Planning Operators through Targeted Reinforcement Learning. *arXiv preprint arXiv:2012.13037*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized Experience Replay. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Shen, J.; Zhuo, H. H.; Xu, J.; Zhong, B.; and Pan, S. J. 2020. Transfer Value Iteration Networks. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, 5676–5683. AAAI Press.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T. P.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nat.*, 529(7587): 484–489.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of Go without human knowledge. *Nature*, 550(7676): 354–359.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.*, 112(1-2): 181–211.

van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep Reinforcement Learning with Double Q-Learning. In Schuurmans, D.; and Wellman, M. P., eds., *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2094–2100. AAAI Press.

Yang, F.; Lyu, D.; Liu, B.; and Gustafson, S. 2018. PEORL: Integrating Symbolic Planning and Hierarchical Reinforcement Learning for Robust Decision-Making. In Lang, J., ed., *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 4860–4866. ijcai.org.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artif. Intell.*, 171(2-3): 107–143.

Zhou, Z.; Li, X.; and Zare, R. N. 2017. Optimizing chemical reactions with deep reinforcement learning. *ACS central science*, 3(12): 1337–1344.

Zhuo, H. H.; and Kambhampati, S. 2017. Model-lite planning: Case-based vs. model-based approaches. *Artif. Intell.*, 246: 1–21.

Zhuo, H. H.; and Yang, Q. 2014. Action-model acquisition for planning via transfer learning. *Artif. Intell.*, 212: 80–103.

Zhuo, H. H.; Yang, Q.; Hu, D. H.; and Li, L. 2010. Learning complex action models with quantifiers and logical implications. *Artif. Intell.*, 174(18): 1540–1569.