HIERARCHICAL W-LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Inspired by a model of the brain called projective simulation, which has attracted interest within the physics community in recent years, we develop a simple and general framework for hierarchical reinforcement learning. The proposed method extends the conventional action-value Q-function to a hierarchical W-function, enabling an agent to select actions according to a hierarchy of strategies. We first present a rigorous formulation of the hierarchical framework, together with the corresponding W-learning algorithm and the hierarchical policy gradient theorem. We then demonstrate the proposed method on a navigation task to illustrate the learning procedure and its underlying mechanism. Furthermore, we extend the hierarchical framework to several modern reinforcement learning algorithms, including hierarchical deterministic policy gradient (DPG), advantage actor-critic (A2C), proximal policy optimization (PPO), and soft actor-critic (SAC), and evaluate them on the continuous navigation task along with MuJoCo control benchmarks. Experimental results show that incorporating an appropriate hierarchical strategy can consistently improve learning performance over conventional reinforcement learning methods, provided that the hierarchy is well designed and the update parameters are properly chosen.

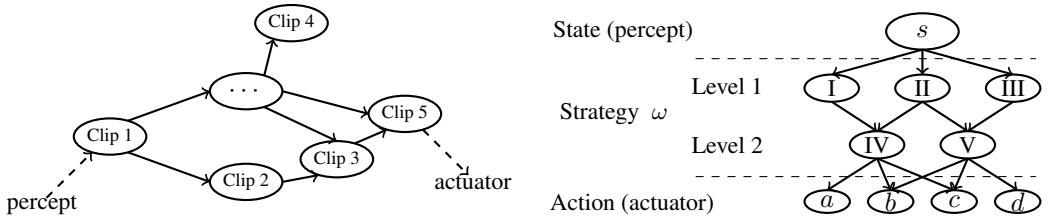
1 INTRODUCTION

The human brain is one of the most complicated and fascinating objects in nature, and many people believe that building a proper model of it may finally lead to artificial intelligence. In recent years, inspired by the study of path integrals in quantum mechanics, physicists have constructed a model for the brain called projective simulation (Briegel, 2012; Briegel & De las Cuevas, 2012; Paparo et al., 2014; Melnikov et al., 2017; Dunjko & Briegel, 2018; Boyajian et al., 2020; Flamini et al., 2024) which aims to keep track of the agent's memory and make the agent learn from the experience. The basic unit in the projective simulation is the clip units and the stimulation flowing between the clips is described by the transition probability.

The clip could represent any information stored in the brain such as an action, a state, or a series of actions and states. The learning procedure starts from a signal that stimulates a percept, and then the signal will go through the entire clip structure following the probability distribution describes by the arrows between clips. Finally, the signal will reach a clip that stimulates an actuator, leading to the action of the agent in the real physical world as shown in Fig.1a. After each percept-actuator cycle, the structure of the clips along with the transition probability, will update according to the feedback given by the environment where the agent is embedded.

There are various conceptual discussions about the creation and annihilation of the clips, as well as the dynamic evolution of the clip structure. Here, in this article, we will study the update of the transition probability in the context of reinforcement learning (Sutton & Barto, 1998) and focus on the hierarchical structure of those clips. Before diving into the rigorous construction, we first introduce the rules for drawing a network for the hierarchical clip structure. The network starts with the clip for the states s and ends at the clips which represent the action a of the agent. There are intermediate layers consisting of clips that represent the strategy ω. Moreover, there could be arrows between two levels that describe the agent's decision procedure while no arrows are allowed between two clips inside a single layer. A hierarchical network example is shown in Fig.1b. Here we should note that the design of the strategy is fixed before the training while the goal of training is to teach the agent what strategy to choose at each state. And, based on the real situation, the agent should be able to learn the action to take after fixing the strategy.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Projective simulation model

(b) The hierarchical structure

Figure 1: (a) Illustration of the projective simulation model. (b) Example of a hierarchical clip network with a two-level strategy structure consisting of 5 strategy clips, denoted by Roman numerals I-V. Nodes labeled by s represent states (percepts), while actions (actuators) are depicted as nodes a, b, c and d. The hierarchical architecture is illustrated by arrows indicating information flow between the different levels.

The traditional reinforcement learning describes the behavior of the agent as a probability distribution over the actions based on the state, and the goal of the training procedure is to determine the optimal policy which maximizes the expected reward following it. Here the hierarchical clip structure focuses more on the internal structure of the agent, and at the same time, the evolution of such structure is determined by the accumulated reward. There is already a significant amount of work on the study of the hierarchical reinforcement learning (Sutton et al., 1999b; Dietterich, 2000; Hengst et al., 2002; Botvinick, 2008; Bacon et al., 2017; Zhou et al., 2019; Hutsebaut-Buysse et al., 2022) and here we focus on the hierarchical generalization of Q-learning (Watkins et al., 1989; Watkins & Dayan, 1992). In our work, the action-value function Q is extended to the strategy-value called the W function in order to reflect the hierarchical structure of the strategies, also we can see that there is no need to construct a termination function and the update rule is a simple repetition of the update rule for Q-learning at each layer. Moreover, in the W-learning, the agent could be taught to learn both the policy-over-option and the intra-option, therefore the method is more flexible and could be applied to deal with more dynamical tasks. In fact, it is possible to learn all the tasks provided that the strategy could be represented in terms of a hierarchical clip structure.

In this paper, we consider the experimental implementation of such a method in a navigation task and compare the results with the corresponding Q-learning, Actor-Critic and REINFORCE (with baseline) algorithms. Then the hierarchical algorithms, including hierarchical DPG, A2C, PPO and SAC, are also investigated on various tasks in the continuous domain. From the experiment, we see that hierarchical learning performs better than the standard learning if the strategy is properly designed and it is also interesting to note that the assignment of the update parameters between different levels will also play an important role in practice.

2 Hierarchical Structures

In reinforcement learning, an agent interacts with its environment over a sequence of discrete time steps. At each time step t, the agent is in a state $s_t \in \mathcal{S}$, where $\mathcal{S}$ denotes the state space, and selects an action a_t according to the policy $\pi(\cdot \mid s_t)$. After executing the action a_t, the agent transitions to the next state s_{t+1} and receives a reward r_{t+1}. To evaluate the performance of the agent following a given policy π, one can define the state-value function $V_\pi(s)$

$$V_\pi(s) = \mathbb{E}_\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s \right\}, \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor of the rewards. From the above definition, one can see that the value function tells us the expectation of the total reward during the whole process following the policy π if the initial state is set as s. Moreover, one can also define the action-value function as

$$Q(s, a) = \mathbb{E}_\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, a_0 = a \right\} \tag{2}$$

which gives us the expectation of the accumulated reward if the initial state and action are set as $s_0 = s, a_0 = a$.

We now study the hierarchical structure of the policies, where the strategy at the lth-level is denoted as ω_l. The policy π can be expressed as

$$\pi_\theta(a \mid s) = \sum_\omega \pi_{\theta_{l-1}}(a \mid \omega_{l-1}) \cdots \pi_{\theta_i}(\omega_{i+1} \mid \omega_i) \cdots \pi_{\theta_0}(\omega_1 \mid s) \tag{3}$$

where $\pi_{\theta_i}(\omega_{i+1} \mid \omega_i)$ tells us the probability of choosing the strategy ω_{i+1} at the level $i+1$ given that the strategy of the i th level is chosen as ω_i. The parameters θ_i are introduced here to describe the behavior of the strategies for some fixed basis[1]. We should note that the strategy ω_i at the level i and the parameter θ_i are in fact sets of vectors $\theta_i = (\boldsymbol{\theta}_i^1, \cdots \boldsymbol{\theta}_i^k)$ and $\omega_i = (\omega_i^1, \cdots, \omega_i^k)$ where k is the number of possible strategies at the level i. Based on the definition of the strategy and the corresponding hierarchical structure, we can write down the strategy-value function as

$$W(s, \omega, a) := \mathbb{E}_\pi \left\{ \sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s, \omega_0 = \omega, a_0 = a \right\}, \tag{4}$$

which gives us the expectation of the total reward if the initial state, strategy and action are chosen as s, ω, a. Moreover, given the strategy-value function, one can also define the W value at the level $l-1$ as

$$W(s, \omega_1, \ldots, \omega_{l-1}) = \sum_a \pi_{\theta_{l-1}}(a \mid \omega_{l-1}) W(s, \omega_1, \ldots, \omega_{l-1}, a) \tag{5}$$

which tells the expectation of the total reward if the initial state and strategy is set as s and $\omega = (\omega_1, \ldots, \omega_{l-1})$. The hierarchical W value between the $l-1$ and $l-2$ has the recursion relation

$$W(s, \omega_1, \ldots, \omega_{l-2}) = \sum_{\omega_{l-1}} \pi_{\theta_{l-2}}(\omega_{l-1} \mid \omega_{l-2}) W(s, \omega_1, \ldots, \omega_{l-2}, \omega_{l-1}). \tag{6}$$

We can also define the weight function $\mu_{\omega,\theta}(s) = \pi_{\theta_{l-2}}(\omega_{l-1} \mid \omega_{l-2}) \cdots \pi_{\theta_0}(\omega_1 \mid s)$, which gives the probability that the agent chooses the strategy ω at the state s. Moreover, according to the definition of $V(s)$ and $W(s, \omega, a)$, we have the relation

$$W(s, \omega, a) = r(s, \omega, a) + \gamma \sum_{s'} p(s' \mid s, \omega, a) V(s') \tag{7}$$

where $r(s, \omega, a) = \mathbb{E}\{r_{t+1} \mid s_t = s, \omega_t = \omega, a_t = a\}$ and $p(s' \mid s, \omega, a)$ is the probability of $s_{t+1} = s'$ when the state, strategy and the action of the t step is chosen as (s, ω, a). By summing over all the strategies ω, weighted by μ, we will obtain the relation between V and Q, familiar from the relation introduced in the standard reinforcement learning literature.

The goal of the learning procedure is to determine the optimal policy π^* that has the maximum total value function $V^*(s)$ which satisfies the Bellman optimality equation. And the stochastic approximation of Bellman operator is to follow the update rule

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha(t)(r_{t+1}(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)) \tag{8}$$

where the parameter $\alpha(t)$ is the learning rate and γ is called the discount factor. It has been proven that the Q value will converge to the maximum Q^*. For the hierarchical structure of the policy, based on the formula equation 8, here we introduce the update rule for the W function written as

$$W_k(s_t, \omega_t) \longleftarrow W_k(s_t, \omega_t) + \alpha_k(t)(g_k(s_t, \omega_t, a_t, r_{t+1}) + \gamma_k \max_\omega W_k(s_{t+1}, \omega) - W_k(s_t, \omega_t)) \tag{9}$$

where the parameters $\alpha_k(t)$ are learning rate for the k layer in the strategy hierarchical structure and, for simplicity, $W(s, \omega_1, \ldots, \omega_k)$ is denoted as $W_k(s, \omega)$. g_k is the parameter that determines the update for the higher level W function and it depends on the state, action a_t, strategy ω_t and the reward r_{t+1}. Later in the experiment, we will see that it plays a crucial role for the performance of the agent. Although in practice it is more convenient to determine the value W, we should note

[1]More precisely, the policy $\pi_\theta(a \mid s)$ is given by the soft-max distribution $\pi_\theta(a \mid s) = \frac{\exp \boldsymbol{\theta}^t \cdot \mathbf{x}(a,s)}{\sum_b \exp \boldsymbol{\theta}^t \cdot \mathbf{x}(b,s)}$, where $\mathbf{x}(b, s)$ are the corresponding basis expressed as a vector.

that there is no one-to-one correspondence between W and the policy π together with the strategies. Given the learned W function, one can determine the strategy layer by layer following the rule

$$\omega_i^* = \arg\max_{\omega_i} W(s, \omega_1^*, \ldots, \omega_{i-1}^*, \omega_i) \tag{10}$$

therefore the policy is determined as $(s, \omega_1^*, \ldots, \omega_{l-1}^*, a^*)$, and the algorithm can be described as below.

Algorithm 1: Hierarchical W Learning

Algorithm parameters: learning rate and discount factors $\alpha_i, \gamma_i \in (0, 1]$
Initialize $W(s, \omega, a)$ for all s, ω and a

Loop for each episode:
  Initialize s_0 and $t = 0$
  Loop for each step of episode:
    Choose the strategy ω_t from s_t using the policy derived from W
    Then determine and execute the action a_t, observe r_{t+1}, s_{t+1}
    Choose the strategy ω_{t+1} using the W function
    For all k: $W_k(s_t, \omega_t) \longleftarrow W_k(s_t, \omega_t) + \alpha_k(g_k + \gamma_k \max_\omega W_k(s_{t+1}, \omega) - W_k(s_t, \omega_t))$
    $t = t + 1$
  until s_t is terminal

As we can see, in the context of reinforcement learning, the key is to figure out an optimal policy as a probability distribution $\pi^*(s, \omega, a)$, which gives the maximum expected reward $V^*(s)$ throughout the training process starting from the state s. However, although the method for updating the function W provides us a convenient way to obtain an optimal policy, it is still necessary to study and keep track of the update of the policy in a straightforward way. In our setup, given the basis, the policy update now becomes the update for the parameters θ, written as

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \alpha_i(t) \frac{\partial V(s)}{\partial \theta_i} \tag{11}$$

where $\partial_\theta V$ is the gradient for the value function giving the direction for the θ leading to a greater value and $\alpha_i(t)$ is the learning rate for the policy. The superscripts t are used to label the training step.

Theorem 1 (Hierarchical Policy Gradient)**.** *In the context of hierarchical reinforcement learning described by the conditional probability $\pi(\omega_{i+1} \mid \omega_i)$, given all the strategy-value functions $W(s, \omega_1, \cdots \omega_i)$, the derivative of the value function $V(s)$ takes the form of*

$$\frac{\partial V(s)}{\partial \theta_i} = \sum_{s', \omega_1, \ldots, \omega_i} \sum_{k=0}^{\infty} \gamma^k \Pr(s \to s', k, \pi) W(s', \omega_1, \ldots, \omega_i) \frac{\partial \pi_{\theta_i}(\omega_1, \ldots, \omega_i \mid s')}{\partial \theta_i} \tag{12}$$

where $\Pr(s \to s', k, \pi)$ is the probability that the agent is at the state s' after k steps.

In practice, during the training, one can use the sample at S_t, and Ω_t^i to approximate the sum over s and ω in equation B.3 thus we have

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \alpha_i(t) W(S_t, \Omega_t^i) \frac{\partial \ln \pi_{\theta_i}(\Omega_t^i \mid S_t)}{\partial \theta_i} \tag{13}$$

where S_t and Ω_t^i are the value of the action the strategy at the step t. Moreover, one can also introduce the baseline $b_i(s)$ at each level to establish the hierarchical REINFORCE, REINFORCE with Baseline and Actor-Critic learning algorithms. Such generalization is straightforward and we present the algorithms in appendix B.3. The convergence for the update procedure equation 9 and equation 13 is shown in the appendix A.

In addition to the stochastic case, we can also express the policy in a deterministic way where strategies and actions are given by functions μ parameterized by θ, written as

$$\omega_1 = \mu_{\theta_1}(s), \ \ldots, \omega_i = \mu_{\theta_i}(\omega_{i-1}), \ \ldots, \ a = \mu_{\theta_l}(\omega_l). \tag{14}$$

Then the W functions take the form of

$$W^{\mu}(s, \omega_1, \ldots, \omega_l, a) = W(s,\ \mu_1(s), \ldots, \mu_{l-1\ldots1},\ \mu_{l\cdots1}(s)) \tag{15}$$

where $\mu_{i\ldots1}(s) = \mu_{\theta_i} \circ \mu_{\theta_{i-1}} \cdots \mu_{\theta_2} \circ \mu_{\theta_1}(s)$, and for the value function, we have

$$V_d^{\mu}(s) = W(s, \mu_1(s)) = \cdots = W(s, \mu_1(s), \ldots, \mu_{i\ldots1}(s)). \tag{16}$$

in which the deterministic value function is denoted as $V_d(s)$. Following similar procedure shown in Silver et al. (2014), the gradient of $V_d(s)$ is given by the theorem below

Theorem 2 (Hierarchical Deterministic Policy Gradient)**.** *Suppose that the MDP satisfies that the conditions $p(s'|s,\omega)$, $\nabla_\omega p(s'|s,\omega)$, $\mu_{\theta_i}(\omega)$, $\nabla_{\theta_i}\mu_{\theta_i}(\omega)$, $r(s,\omega)$, $\nabla_\omega r(r,\omega)$ are continuous in all parameters and variables s, ω, a, s' and θ_i; these imply that $\nabla_\omega W(s,\omega)$ and the hierarchical deterministic policy gradient exist. Then,*

$$\nabla_{\theta_i} V_d(s) = \int_{\mathcal{S}} ds' \sum_{t=0}^{\infty} \gamma^t \Pr(s \to s', t, \mu) \nabla_{\theta_i}\mu_{\theta_i}(s') \nabla_{\omega_i} W(s', \omega_1, \ldots, \omega_i)\Big|_{\omega_i = \mu_{i\ldots1(s)}}. \tag{17}$$

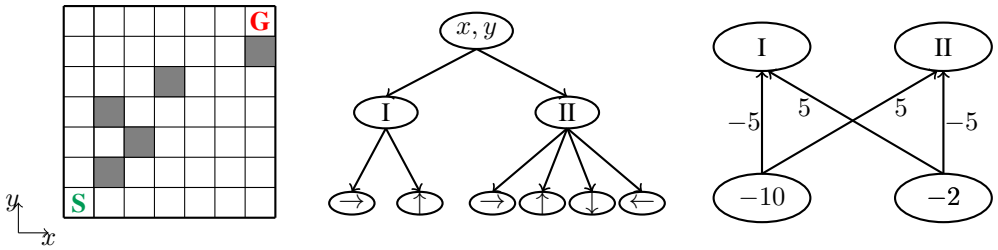
Moreover, given the initial state distribution density $p_1(s)$, one can define the discounted state distribution by $\rho^{\pi}(s') := \int \sum_{t=1}^{\infty} \gamma^{t-1} p_1(s) p(s \to s', t, \pi) ds$. The performance object is written as $J(\pi_\theta) = \mathbb{E}_{s\sim\rho^{\pi}}[V_d(s)]$, therefore, the parameters θ update along the direction of $\partial_\theta(J(\pi_\theta))$, written as

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \alpha_i(t) \mathbb{E}_{s\sim\rho^{\mu}}\Big[\nabla_{\theta_i}\mu_{\theta_i}(s) \nabla_{\omega_i} W(s, \omega_1, \ldots, \omega_i)\Big]. \tag{18}$$

Proof of the theorem and the corresponding hierarchical deterministic policy gradient (HDPG) algorithm are shown in appendix B.2 and appendix B.3.

3 Experiment

3.1 Showcase in the Grid World Navigation



(a) Gridworld (b) Hierarchy Strategy (c) Feedback Mechanism

Figure 2: Illustration of the navigation in the grid world and the strategy

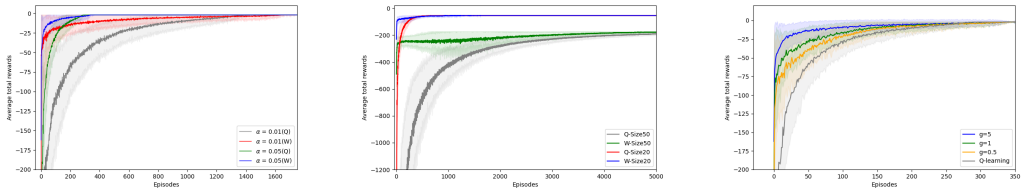
Now, as an example, we consider the navigation task to illustrate how the hierarchical W learning works. In our case, the agent moves in a 7-by-7 grid world with obstacles, depicted as gray boxes in Fig.2a. It starts from the bottom left corner and the goal is set at the top right corner, denoted as the green S and red G, respectively. Moreover, during the training, the agent will get the reward of -10 if it hits the boundary or obstacles and will receive +20 if it reaches the goal. The agent receives -2 points for moving into an empty box. For convenience, we label the position of the box as (x, y) following the convention of Cartesian coordinates and set the starting point S as $(1, 1)$.

In the context of reinforcement learning, now the state s is described by a pair of integers (x, y) and the action a becomes the movement following the direction: right, left, up and down. In order to establish the hierarchical structure, here we introduce two strategies $\omega_1 = (\omega_1^1, \omega_1^2) = (\mathrm{I}, \mathrm{II})$ for the agent: strategy I is to move towards the goal while strategy II is to avoid the obstacles. Moreover, based on the observation that the starting point and the goal are located at two diagonal corners of the grid world, strategy I is then designed in the way that it will only lead to the right and up actions while the strategy II will lead to all four actions in order to enable the agent to get rid of various kinds of obstacles as shown in Fig.2b.

For the update rule, similar to the Q-learning, we set the low level update parameters equal to the rewards, i.e. $g_2 = r_{t+1}$. For the higher level update, we set the parameter as

$$g_1(\text{ I}, r = -2) = g_1(\text{ II}, r = -10) = 5 \qquad g_1(\text{ I}, r = -10) = g_1(\text{ II}, r = -2) = -5 \tag{19}$$

based on the feedback machinery illustrated in Fig.2c where the absolute value is set as $|g_1| = 5$. That is, when the agent hits an obstacle and receives a reward of -10, strategy I receives feedback of -5, and strategy II receives feedback of $+5$. Conversely, if the agent goes into an empty box and receives the reward of -2, we assign a $+5$ feedback to the strategy I and -5 to the strategy II, which implies that the agent would be more likely to choose the strategy I when the next state is empty therefore, it will be guided to the goal in a faster way.



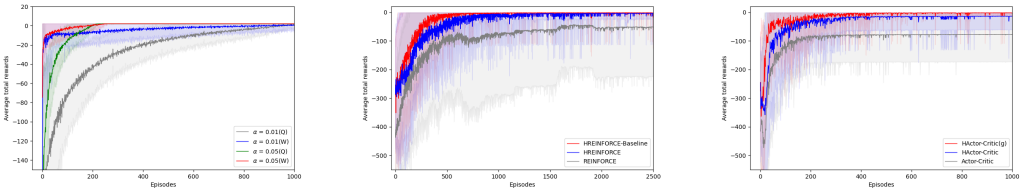
(a) Q and W-learning (b) 20×20, 50×50 grid world (c) $|g_1|$ = 5, 1 and 0.5

Figure 3: Both of Q and W-learning results for different learning rates are shown in (a). More experiments on the 20 by 20 and 50 by 50 grid world are shown in (b). Training for $|g_1|$=0.5, 1, and 5 is shown in (c). The reward is plotted by taking the average of 200 runs.

To investigate the effects of the hierarchical structure in a sensible way, we plot the behavior of the reward for the Q-learning without the strategies and put them together with the W-learning results in Fig.3a. There are two groups of curves in the figure, one for the learning rate $\alpha = 0.05$ and the other for $\alpha = 0.01$. From the figure, we see that the reward curves for W and Q learning will reach the optimal level at around the same number of episodes for the same learning rate but the reward curve for W-learning will grow more rapidly. Then we investigate how training behaves as the grid world size increases. The obstacle configurations are the same as the previous 7-by-7 case, so the effect of grid world size becomes dominant. It shows in Fig.3b that W-learning performs better at the early state then converge to the same value as Q-learning. Moreover, one can also improve the strategy to make it state dependent on a function $F(x, y)$. Detail of the construction and the learning results are shown in the appendix D.

Next we explore more about the effects of the parameters g_1. At first we vary the absolute value $|g_1|$, from 5 and 1 to 0.5, it shows in Fig.3c that the bigger absolute value of g_1 tends to have a faster convergence speed. Apart from changing the absolute value of the parameter, we can also vary their signs. For a given absolute value $|g_1|$, in our case $|g_1| = 5$, there are sixteen kinds of combinations of the parameters which can be sorted into three groups corresponding to three types of curves. Following the convention introduced in appendix C, we label the parameters with a combination of four $\pm$ signs. For example, the case we have studied in equation 19 can be denoted as $(-++-)$. The first group of parameters behaves the same as $(-++-)$, containing $(++--), (+++-)$ and $(+-++)$. The second group follows the curve for Q-learning, consisting of $(+--+), (--++), (---+)$ and $(+-++)$. The third group of curves lies between them which contains $(+---), (-+++), (+-+-), (-+-+), (++-+), (--+-), (+++-)$ and $(----)$. All of the details are shown in the appendix.

Based on the study of W-learning and Q-learning, one can move on to study their generalization to double learning and n-step learning. The plot for double Q and W- learning is shown is Fig.4a. For more details, one can see the analysis in Appendix E. At the end of ths section, we present the study of various learning methods based on the hierarchical policy gradient theorem in Fig.4b and 4c, including hierarchical REINFORCE, REINFORCE with Baseline and Actor-Critic. Given the plot, one can see that hierarchical learning outperforms its counterpart from two aspects. One is the average convergence speed, and the other is the average convergence limit. The increase for the learning speed results from the introduction of the strategies and the asymptotic reward value reflects the robustness of the training. Here we set the maximum training step for each episode as 100. The agent equipped with a strategy in general takes fewer steps to reach the goal. Therefore,

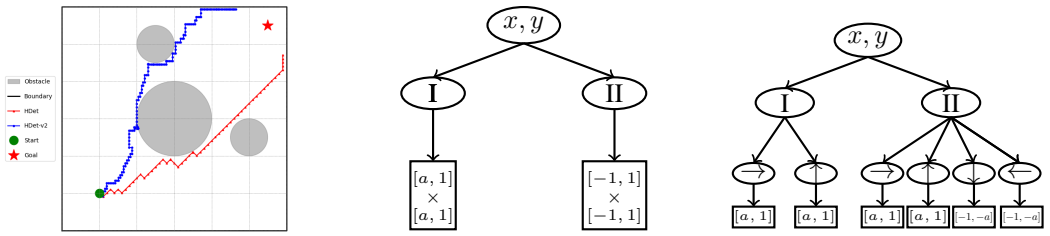


(a) Double Q and W-learning (b) REINFORCE learning (c) Actor-critic learning

Figure 4: Learning results for double Q and W-learning are shown in (a). Hierarchical REINFORCE and REINFORCE with Baseline learning results for learning rate 0.005 are depicted in (b). Hierarchical actor-critic learning results are shown in (c) with learning rate 0.001. For HActor-Critic, parameters are set to $G_1 = G_2 = r$ while different update strengths for two levels, $G_1 = g, G_2 = r$, are introduced during the training denoted as HActor-Critic(g). Learning curves are plotted by taking the average of 100 runs.

fewer failed episodes occur during the training, and the reward curve on average converges to the value closer to the optimal one.

3.2 Comparison to Other Algorithms in the Continuous Domain



(a) Continuous navigation (b) One-level strategy(HDPG) (c) Two-level strategy(HDPGv2)

Figure 5: Illustration of the navigation in the continuous domain and the corresponding strategies. Continuous variables in the square node are learned by DPG.

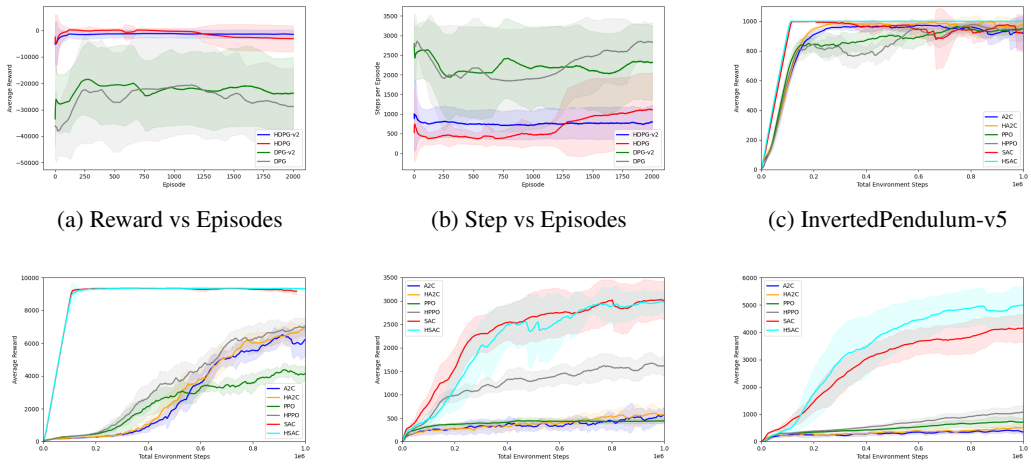
For the continuous navigation task, we put the agent, denoted as the green circle, in the world described by the continuous coordinates space $(x, y) \in [0, 6] \times [0, 6]$, and the initial position is $(1, 1)$. The center of the goal is located at the position $(5.5, 5.5)$, and we say that the agent reaches the goal successfully when the distance between the center and the agent d is less than 1, i.e., $d \leq 1$. Moreover, when the agent hits the obstacles shown as the gray circles in Fig. 5a, it will get a reward of $r = -2$. We denote such a case by o. Otherwise, the reward is $r = -2d$, which we denote by $\bar{o}$. Given the setup of the environment, we choose the update parameters to be

$$g_1(\text{ I}, o) = -10, \qquad g_1(\text{ I}, \bar{o}) = 0.5, \qquad g_1(\text{ II}, o) = 0.5, \qquad g_1(\text{ II}, \bar{o}) = -10, \tag{20}$$

where I and II represent the first level strategies introduced earlier and the corresponding policy can be learned using the actor-critic method. The action also takes a continuous value and we choose to learn it using the deterministic policy gradient (DPG) method.

Here we present two hierarchical structures for continuous navigation. One is similar to the hierarchy shown in Fig. 2b, which employs a one-level strategy. Instead of discrete actions, we use DPG to select an action in $[0, 1] \times [0, 1]$ for strategy I and an action in $[-1, 1] \times [-1, 1]$ for strategy II. The update parameters are set to $g_2 = r$ and the corresponding algorithm is named HDPG. The other structure has a two-level strategy shown in Fig. 5c, where the first level has the options of going to the goal and avoiding obstacles while the second level has the options of different directions. Once the direction is chosen, we use DPG to determine the specific value of the action within either $[0, 1]$ or $[-1, 0]$ and the update parameters are set as $g_2 = g_3 = r$. Such a method is denoted as HDPGv2.

In practice, the action is clipped to the interval determined by a small random number a close to 0 and rescaled by a factor of 0.1.



(a) Reward vs Episodes (b) Step vs Episodes (c) InvertedPendulum-v5

(d) InvertedDoublePendulum-v5 (e) Hopper-v5 (f) Walker2d-v5

Figure 6: Learning results on the continuous navigation and MuJoCo control tasks. The result is plotted by taking the average of 10 trials and curves are smoothed by averaging the reward over 100 episodes.

We set the maximum number of steps to 3000 for each episode, and the learning results are shown in Fig. 6a and Fig. 6b, where the average reward and the number of steps are plotted as functions of the episode number, respectively. From the training results, we find it is hard for the agent without a strategy to reach the goal within the maximum number of steps because the agent tends to get stuck near obstacles or to continue exploring regions far from the goal. However, the agent is able to bypass the obstacles and reach the goal in a more efficient way when using the proposed strategy.

In the end, we introduce the hierarchical structure into the A2C, PPO and SAC (Schulman et al., 2017; Haarnoja et al., 2018) learning algorithms, denoted as HA2C, HPPO, HSAC, and test them in the MoJuCo continuous control benchmark. There is as yet no universal structure for all tasks, instead, the basic strategy we used here is to introduce a one-level strategy to control the system variables separately. Taking the InvertedPendulum task as an example, two of the key quantities for the system are the distance from the center x and the angular velocity $\dot{\theta}$. Therefore, we set two options for the strategy, I for controlling the distance and II for controlling the angular velocity. The update parameters corresponding to this strategy are chosen as $g_1 = r$ and

$$g_2(\text{ I }) = \frac{g_1}{0.9 + 5|x|}, \qquad g_2(\text{ II }) = \frac{g_1}{0.9 + 0.4|\dot{\theta}|}, \tag{21}$$

which means that strategy I tends to make $|x|$ smaller and strategy II tends to keep $|\dot{\theta}|$ at a low level. A similar strategy is used for the InvertedDoublePendulum while we use the strategy to guide the system to move further along the x-axis with smaller action in the Hopper and the Walker2d task. It turns out that a proper set of parameters improves the agent's performance but different algorithms require different parameters and sometimes the fine-tuning will be sophisticated. The learning results are shown in Figs. 6c, 6d, 6e and 6f.

4 Related Works

Option Framework We can compare the option framework (Sutton et al., 1999b) with W-learning contains two layers, where the first level learns the subgoals while the higher level learns the policy over options, corresponding to the option and intra-option Q-learning for a semi-MDP. One of the main differences is that there is no need to define the termination function β in the context of W-learning. In fact, W-learning procedure is more like setting $\beta = 1$ for all states i.e., we choose a

new option at every step. The other difference is the details of the update rules for two levels. For W-learning, we need to generalize the reward to a series of parameters g at different levels while the update rule for the option framework depends on the termination function. In both cases, these quantities must be fixed by hand before training.

MAXQ In the context of MAXQ (Dietterich, 2000), the Q function is divided into the local and global pieces according to the termination function. The parent task is divided into subtasks represented in terms of the sequence of the agent's actions and states. Here the strategy in W-learning reflects the agent's reasoning between a single state and action, therefore all the W_k functions are global, taking all the following steps into consideration. Moreover, the optimal policy for MAXQ is determined recursively, which requires all the subtask policies to be optimal. Therefore, the recursively optimal policy is weaker than the globally optimal policy given by Q-learning. For W-learning, the optimal policy is determined in the same way as Q-learning by figuring out the unique fixed point of the Bellman optimality operator. It enables us to find stronger policy compared to the globally optimal policy by constructing more complicated clip structure.

Option-Critic The option-critic method (Bacon et al., 2017) is the further development of semi-MDP based on the study of gradient of intra-option policy and termination function. Compared with the option framework, rather than input the termination function before the training, it enables the agent to learn β during the options improvement part of the training procedure. A learnable termination function requires fewer input data during the training but there is no evidence that better performance is guaranteed. In the context of W-learning, we are able to directly learn intra-option policy and policy over options at the same time with the help of hierarchical gradient theorem while a hand-crafted architecture for the policy is often required if the agent wants to outperform the standard one.

Strategy Discovery From the study of projective simulation, we see that there are fruitful structures to investigate, for example, the dynamic evolution of the clips. In the context of hierarchical reinforcement learning, this is connected to the evolution of the strategy, that is, how to train the agent to learn strategies by itself rather than fixing them manually before training. The discovery of strategies is studied in the reinforcement learning literature (McGovern & Barto, 2001; Menache et al., 2002; Şimşek & Barto, 2004; Şimşek et al., 2005; Gregor et al., 2016; Machado et al., 2017).

We are using hand-crafted architecture in this paper so the agent has better performance than the standard one. It is possible to let the agent generate new architectures by itself. For example, in the navigation task, one can start from a fully connected two-level clip network where the two high-level strategies are connected to the four action nodes. Then we put the agent in the empty box without obstacles and do the pretraining. During the pretraining, we can set up a threshold with a small parameter ϵ and remove the low probability legs for $\pi(a \mid \omega) < \epsilon$. After the pretraining, we fix the architecture and apply it to the more complicated tasks with obstacles. In our case, we start from two identical strategies and they are both connected to all four actions. During the pretraining procedure, we shape the architecture of one strategy to $F(x, y)$ by cutting the lower probability legs out of the network. At the same time, we keep the other strategy network fully connected so that the agent could be flexible to deal with more dynamic tasks involving obstacles. Ultimately, we will obtain the architecture presented before and we can see it makes the agent perform better than the standard agent without the hierarchical structure.

5 Conclusions

In this article, based on projective simulation, we introduce a new method for hierarchical reinforcement learning, which generalizes Q-learning to hierarchical W-learning. W-learning enables us to incorporate various strategies into the agent, together with the corresponding hierarchical Actor-Critic and REINFORCE algorithms, we illustrate these learning methods in the context of a gridworld navigation task. Furthermore, we introduce the hierarchical structure into modern reinforcement learning algorithms, such as hierarchical DPG, A2C, PPO, and SAC. We evaluate these algorithms on continuous navigation and MuJoCo control tasks. Experimental results show that hierarchical learning performs better than the standard learning provided that a hand-crafted strategy and an appropriate set of update parameters are specified.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Dimitri Bertsekas and John Tsitsiklis. *Parallel and distributed computation: numerical methods*. Athena Scientific, 2015.

Dimitri Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

Matthew M Botvinick. Hierarchical models of behavior and prefrontal function. *Trends in cognitive sciences*, 12(5):201–208, 2008.

Walter L Boyajian, Jens Clausen, Lea M Trenkwalder, Vedran Dunjko, and Hans J Briegel. On the convergence of projective-simulation–based reinforcement learning in markov decision processes. *Quantum machine intelligence*, 2(2):13, 2020.

Hans J Briegel. On creative machines and the physical origins of freedom. *Scientific reports*, 2(1): 522, 2012.

Hans J Briegel and Gemma De las Cuevas. Projective simulation for artificial intelligence. *Scientific reports*, 2(1):400, 2012.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

Fulvio Flamini, Marius Krumm, Lukas J Fiderer, Thomas Müller, and Hans J Briegel. Towards interpretable quantum machine learning via single-photon quantum walks. *Quantum Science and Technology*, 9(4):045011, 2024.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. Pmlr, 2018.

Bernhard Hengst et al. Discovering hierarchy in reinforcement learning with hexq. In *Icml*, volume 19, pp. 243–250. Citeseer, 2002.

Matthias Hutsebaut-Buysse, Kevin Mets, and Steven Latré. Hierarchical reinforcement learning: A survey and open research challenges. *Machine Learning and Knowledge Extraction*, 4(1): 172–221, 2022.

Tommi Jaakkola, Michael Jordan, and Satinder Singh. Convergence of stochastic iterative dynamic programming algorithms. *Advances in neural information processing systems*, 6, 1993.

David G Luenberger. *Optimization by vector space methods*. John Wiley & Sons, 1997.

Marlos C Machado, Marc G Bellemare, and Michael Bowling. A laplacian framework for option discovery in reinforcement learning. In *International Conference on Machine Learning*, pp. 2295–2304. PMLR, 2017.

Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

Alexey A Melnikov, Adi Makmal, Vedran Dunjko, and Hans J Briegel. Projective simulation with generalization. *Scientific reports*, 7(1):14430, 2017.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *European conference on machine learning*, pp. 295–306. Springer, 2002.

Giuseppe Davide Paparo, Vedran Dunjko, Adi Makmal, Miguel Angel Martin-Delgado, and Hans J Briegel. Quantum speedup for active learning agents. *Physical Review X*, 4(3):031002, 2014.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–387–I–395. JMLR.org, 2014.

Özgür Şimşek and Andrew G Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, pp. 95, 2004.

Özgür Şimşek, Alicia P Wolfe, and Andrew G Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd international conference on Machine learning*, pp. 816–823, 2005.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999b.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Christopher John Cornish Hellaby Watkins et al. Learning from delayed rewards. 1989.

Xiaomao Zhou, Tao Bai, Yanbin Gao, and Yuntao Han. Vision-based robot navigation through combining unsupervised learning and hierarchical reinforcement learning. *Sensors*, 19(7):1576, 2019.

A Proof of Convergence

In this section, we will show the proof of the convergence for the update rules equation 9 and equation 13. To prove that the W function will converge to the optimal one W^* following the update rule. We first introduce the transition probability involving the strategies $p(s', \omega'_1, \dots \omega'_{i-1} \mid s, \omega_1, \dots, \omega_i)$ as

$$p(s', \omega'_1, \dots, \omega'_{i-1} \mid s, \omega_1, \dots, \omega_i) = \sum_{s, a, \omega_{i+1}, \dots, \omega_{l-1}} \pi(\omega'_{i-1} \mid \omega_{i-2}) \cdots \pi(\omega'_1 \mid s') \times p_{s's}(a) \pi(a \mid \omega_{l-1}) \cdots \pi(\omega_{i+1} \mid \omega_i) \cdots \pi(\omega_1 \mid s) \tag{22}$$

where $p_{s's}(a)$ is the probability that the agent transforms to the state s' given the previous state s and the action a. Then we define the proper policy

Definition 1. A hierarchical policy $\pi(\omega_{i+1}|\omega_i)$ is proper when there exists a function $\xi(s, \omega_1 \dots \omega_{i-1})$ such that the condition

$$\sum_{s'\omega'} p(s', \omega'_1, \dots, \omega'_{i-1} \mid s, \omega_1, \dots, \omega_i) \xi(s', \omega'_1, \dots \omega'_{i-1}) \leq \beta \xi(s, \omega_1, \dots, \omega_{i-1}) \tag{23}$$

is satisfied for all $s, \omega_1, \dots, \omega_i$ and $\beta \in [0, 1)$.

and the norm of the W function

Definition 2. The weighted maximum norm of the W function is written as

$$||W(s,\omega_1,\ldots,\omega_i)||_\xi = \max_{s,\omega_1,\ldots,\omega_i} \frac{|W(s,\omega_1,\ldots,\omega_i)|}{\xi(s,\omega_1,\ldots,\omega_{i-1})}. \tag{24}$$

Based on these two definitions, the convergence for updating the W function is guaranteed by the following theorem.

Theorem 3. *Suppose that the learning rate $\alpha_i(t)$ are nonnegative and satisfy*

$$\sum_{t=0}^{\infty} \alpha_i(t) = \infty, \qquad \sum_{t}^{\infty} \alpha_i^2(t) < \infty \qquad \forall\, i. \tag{25}$$

Then $W(s,\omega_1,\ldots,\omega_i)$, for all i, converge with probability 1 to $W^*(s,\omega_1,\ldots,\omega_i)$ provided that all policies are proper.

Proof. For the proof, we first introduce the mapping H from one W function to the other as

$$HW(s,\omega_1,\ldots,\omega_i) = g + \sum_{s'\omega'} p(s',\omega_1',\ldots,\omega_{i-1}' \mid s,\omega_1,\ldots\omega_i) \max_{\omega'} W(s',\omega_1',\ldots,\omega_{i-1}',\omega').$$

Given the mapping H, the update rule for the W function can be written as

$$W_{t+1} = (1-\alpha_t)W_t + \alpha_t(HW_t + \eta_t). \tag{26}$$

where η_t is the noise term given by

$$\eta_t(s,\omega) = g + \max_{\omega_i} W_t(s,\omega_1,\ldots,\omega_{i-1},\omega_i) - HW_t(s,\omega). \tag{27}$$

Moreover, given the update history $\mathcal{F}_t^i = \{s_0,\ldots,s_t,\omega_0,\ldots,\omega_t^i,a_0,\ldots,a_{t-1}\}$, one can deduce that

$$\mathbb{E}[\eta_t \mid \mathcal{F}_t^i] = 0 \qquad \mathbb{E}[\eta_t^2 \mid \mathcal{F}_t^i] \le C, \tag{28}$$

where C is a constant number. In our case, $s = s_t$ and $s' = s_{t+1}$. Now, we consider the difference between two functions W and $\bar{W}$. Acting the transformation H on it, we have

$$\begin{aligned}
|(HW - H\bar{W})(s,\omega)| &\le \sum_{s'\omega'} p(s',\omega_1',\ldots,\omega_i' \mid s,\omega) |\max_{\omega_i'} W(s',\omega_1',\ldots,\omega_{i-1}',\omega_i') \\
&\qquad - \max_{\omega_i'} \bar{W}(s',\omega_1',\ldots,\omega_{i-1}',\omega_i')| \qquad (29)\\
&\le \sum_{s'\omega'} p(s',\omega_1',\ldots,\omega_{i-1}' \mid s,\omega) |\max_{\omega_i'} |W(s',\omega_1',\ldots,\omega_{i-1}',\omega_i') \\
&\qquad - \bar{W}(s',\omega_1',\ldots,\omega_{i-1}',\omega_i')| \qquad (30)\\
&\le \sum_{s'\omega'} p(s',\omega_i',\ldots,\omega_{i-1}' \mid s,\omega) \xi(s',\omega_i',\ldots,\omega_{i-1}') ||W - \bar{W}||_\xi.
\end{aligned}$$

where we are using $\omega = (\omega_1,\ldots,\omega_i)$ and $\omega' = (\omega_1',\ldots,\omega_{i-1}')$ for simplicity. Given that the policy is proper, we obtain the relation

$$||(HW - H\bar{W})(s,\omega_1,\ldots,\omega_i)||_\xi \le \beta ||(W - \bar{W})(s,\omega_1,\ldots,\omega_i)||_\xi, \tag{31}$$

for $0 \le i \le l$, therefore we can see that H is a contraction mapping and it converges to a unique value W^* called fixed point of the contraction, i.e. $HW^* = W^*$ Luenberger (1997); Bertsekas & Tsitsiklis (2015).

Together with the condition equation 28, we then prove the convergence of the W function according to the convergence property for the random iterative process shown in Jaakkola et al. (1993); Bertsekas & Tsitsiklis (1996). The main idea of the proof for the convergence is to show that $||W_t - W^*|| \le D_t$ for some value D_t at each step and then to verify $D_t \to 0$ when t goes to infinity. This is easy to see when $\eta_t = 0$ since we have $|HW_t - W^*| \le \beta |W_t - W^*|$ while more sophisticated construction of D_t is required when $\eta_t \neq 0$. For details, see the section 4.3.6 of Bertsekas & Tsitsiklis (1996). □

The proof of the convergence for the gradient policy method in equation 13 is the direct application of the Proposition 3.5 shown in Bertsekas & Tsitsiklis (1996) to each i, therefore, we have the following theorem.

Theorem 4. *Suppose that*

$$\max_{\theta_i,j,k}\left|\frac{\partial^2\pi(\omega_{i+1}\mid\omega_i)}{\partial\theta_i^j\partial\theta_i^k}\right|<L<\infty\qquad\forall\, i, \tag{32}$$

where L is a constant upper bound and the nonnegative learning rates satisfy the relation

$$\alpha_i(t)\to 0,\qquad\qquad\sum_t^\infty\alpha_i(t)=\infty\qquad\forall\, i. \tag{33}$$

Then $\pi(\omega_{i+1}\mid\omega_i)$, for all i, converge with probability 1 to π^* i.e., $\frac{\partial V}{\partial\theta_i}\to 0$ when $t\to\infty$.

B Policy Gradient

B.1 Proof of Theorem 1

Proof. The proof is based on the generalization of policy gradient method for the policy without hierarchical structures (Sutton et al., 1999a). From the recursion relation equation 6, we have the derivative for the value function

$$\begin{aligned}\frac{\partial V(s)}{\partial\theta_i} &= \frac{\partial}{\partial\theta_i}\sum_{\omega_1}W(s,\omega_1)\pi_{\theta_1}(\omega_1\mid s) &(34)\\ &= \sum_{\omega_1}\left(\frac{\partial W(s,\omega_1)}{\partial\theta_i}\pi_{\theta_1}(\omega_1\mid s)+W(s,\omega_1)\frac{\partial\pi_{\theta_1}(\omega_1\mid s)}{\partial\theta_i}\right) &(35)\\ &= \sum_{\omega_1}\Bigg(W(s,\omega_1)\frac{\pi_{\theta_1}(\omega_1\mid s)}{\partial\theta_i}+\pi_{\theta_1}(\omega_1\mid s)\sum_{\omega_2}\Big(\pi_{\theta_2}(\omega_2\mid\omega_1)\frac{W(s,\omega_1,\omega_2)}{\partial\theta_i} &(36)\\ &\qquad +W(s,\omega_1,\omega_2)\frac{\partial\pi_{\theta_2}(\omega_2\mid\omega_1)}{\partial\theta_i}\Big)\Bigg). &(37)\end{aligned}$$

where we have used the relation two times, one from $V(s)$ to $W(s,\omega_1)$ the other from $W(s,\omega_1)$ to $W(s,\omega_1,\omega_2)$. Following such a procedure, one can write $W(s,\omega_1,\omega_2)$ in terms of $W(s,\omega_1,\omega_2,\omega_3)$ and then finally reach $V(s')$. At the same time, taking the relation

$$\frac{\partial\pi_{\theta_i}(\omega_{i+1}\mid\omega_i)}{\partial\theta_j}=0 \tag{38}$$

for $j\neq i$ into consideration, we obtain

$$\begin{aligned}\frac{\partial V(s)}{\partial\theta_i} &= \sum_{\omega_1,\ldots,\omega_i}\frac{\partial\pi_{\theta_i}(\omega_1,\ldots\omega_i\mid s)}{\partial\theta_i}W(s,\omega_1,\ldots,\omega_i) &(39)\\ &\quad+\sum_{a,s'\omega}\pi_\theta(\omega,a\mid s)p(s'\mid s,\omega,a)\frac{\partial V(s')}{\partial\theta_i}.\end{aligned}$$

where $\pi_{\theta_i}(\omega_1,\ldots,\omega_i\mid s)=\pi(\omega_i\mid\omega_{i-1})\ldots\pi(\omega_1\mid s)$. Summing over all the components, then we have

$$\frac{\partial V(s)}{\partial\theta_i}=\sum_{s',\omega_1,\ldots,\omega_i}\sum_{k=0}^\infty\gamma^k\Pr(s\to s',k,\pi)W(s',\omega_1,\ldots,\omega_i)\frac{\partial\pi_{\theta_i}(\omega_1,\ldots,\omega_i\mid s')}{\partial\theta_i}. \tag{40}$$

□

Based on the hierarchical policy gradient theorem, now we have the update method for the policy

$$\theta_i^{(t+1)}=\theta_i^{(t)}+\alpha_i(t)\sum_s d^\pi(s)\sum_{\omega_1\cdots\omega_i}W(s,\omega_1,\ldots,\omega_i)\frac{\partial\pi_{\theta_i}(\omega_1,\ldots,\omega_i\mid s)}{\partial\theta_i} \tag{41}$$

where $d^\pi(s)=\sum_{t=0}^\infty\gamma^t\Pr(s_t=s\mid s_0,\pi)$. Moreover, given a function $f_u(s,\omega_1,\cdots,\omega_k)$ with parameters u, we have the function approximation of the gradient, shown in the lemma below.

Lemma 1. *If f_u satisfies the condition*

$$\sum_s d^\pi(s) \sum_{\omega_1,\cdots,\omega_i} \pi(\omega_1,\ldots,\omega_i \mid s)(W(s,\omega_1,\ldots,\omega_k) - f_u(s,\omega_1,\ldots,\omega_k))\frac{\partial f_u}{\partial u} = 0 \tag{42}$$

and

$$\frac{f_u(\omega_1,\ldots,\omega_i,a)}{\partial u} = \frac{\pi(\omega_1,\ldots,\omega_i,\mid s)}{\partial \theta_i}\frac{1}{\pi_{\theta_i}(\omega_1,\ldots,\omega_i \mid s)}, \tag{43}$$

then

$$\frac{\partial V(s)}{\partial \theta_i} = \sum_{s',\omega_1,\ldots,\omega_i} \sum_{k=0}^{\infty} \gamma^k \Pr(s \to s', k, \pi) f_u(s',\omega_1,\ldots,\omega_i)\frac{\partial \pi_{\theta_i}(\omega_1,\ldots,\omega_i \mid s')}{\partial \theta_i}. \tag{44}$$

Therefore, we obtain the update rule for the policy parameters

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \alpha_i(t) \sum_s d^\pi(s) \sum_{\omega_1\cdots\omega_i} f_u(s,\omega_1,\ldots,\omega_i)\frac{\partial \pi_{\theta_i}(\omega_1,\ldots,\omega_i \mid s)}{\partial \theta_i}. \tag{45}$$

In fact, by substituting the policy with the given basis into equation 43, one can check that the function f satisfies the relation $\sum_{\omega_i} \pi(\omega_i \mid \omega_{i-1}) f(\omega_1,\ldots,\omega_i,a) = 0$, therefore, it is more reasonable to interpret the function f as the hierarchical advantage function defined as $A(s,\omega_1,\ldots,\omega_{i+1}) = W(s,\omega_1,\ldots,\omega_i) - W(s,\omega_1,\ldots,\omega_{i-1})$.

B.2 Proof of Theorem 2

Proof. To get the gradient with respect to the parameter θ_i, we use the relation

$$V_d^\mu(s) = W(s,\mu_1(s),\ldots,\mu_l(\omega_{l-1})) = W(s,\mu_1(s),\ldots,\mu_{i\ldots 1}(s)). \tag{46}$$

By directly taking the derivative, we have

$$\begin{aligned}
\nabla_{\theta_i} V^\mu(s) &= \nabla_{\theta_i} W(s,\mu_1(s),\ldots,\mu_{i\ldots 1}(s)) \\
&= \nabla_{\theta_i}\Big(r(s,\omega_1,\ldots,\omega_i) + \gamma \int_{\mathcal{S}} p(s' \mid s,\omega_1,\ldots,\omega_i) V^\mu(s') ds'\Big)\Big|_{\omega_i=\mu_{i\ldots 1}(s)} \\
&= \nabla_{\theta_i}\mu_{\theta_i}(\omega_{i-1})\nabla_{\omega_i}\big(r(s,\omega_1,\ldots,\omega_i) + \gamma \int_{\mathcal{S}} p(s' \mid s,\omega_1,\ldots,\omega_i) V^\mu(s') ds'\big) \\
&\quad + \gamma \int_{\mathcal{S}} p(s' \mid s,\omega_1,\ldots,\omega_i)\nabla_{\theta_i} V^\mu(s') ds'\Big|_{\omega_i=\mu_{i\ldots 1}(s)} \qquad (47) \\
&= \nabla_{\theta_i}\mu_{\theta_i}(\omega_{i-1})\nabla_{\omega_i} W(s,\omega_1,\ldots,\omega_i)\big|_{\omega_i=\mu_{i\ldots 1}(s)} \qquad (48) \\
&\quad + \gamma \int_{\mathcal{S}} \Pr(s \to s', 1, \mu)\nabla_{\theta_i} V^\mu(s') ds'.
\end{aligned}$$

After applying above procedure on the second part recursively, we obtain

$$\begin{aligned}
\nabla_{\theta_i} V^\mu(s) &= \nabla_{\theta_i}\mu_{\theta_i}(\omega_{i-1})\nabla_{\omega_i} W(s,\omega_1,\ldots,\omega_i)\big|_{\omega_i=\mu_{i\ldots 1}(s)} \qquad (49) \\
&\quad + \gamma \int_{\mathcal{S}} ds' \Pr(s \to s', 1, \mu)\nabla_{\theta_i}\mu_{\theta_i}(\omega_{i-1})\nabla_{\omega_i} W(s',\omega_1,\ldots,\omega_i)\Big|_{\omega_i=\mu_{l\ldots 1}(s)} \\
&\quad + \gamma^2 \int_{\mathcal{S}} \Pr(s \to s', 1, \mu) \int_{\mathcal{S}} \Pr(s' \to s'', 1, \mu)\nabla_{\theta_i} V^\mu(s'') ds' ds'' \\
&= \int_{\mathcal{S}} ds' \sum_{t=0}^{\infty} \gamma^t \Pr(s \to s', t, \mu)\nabla_{\theta_i}\mu_{\theta_i}(s')\nabla_{\omega_i} W(s',\omega_1,\ldots,\omega_i)\Big|_{\omega_i=\mu_{i\ldots 1(s)}}
\end{aligned}$$

therefore, we finish the proof. Taking the density $\rho^\pi(s)$ into consideration, we have

$$\begin{aligned}
\nabla_{\theta_i} J(\mu_\theta) &= \int_{\mathcal{S}} ds\ \rho^\mu(s)\nabla_{\theta_i}\mu_{\theta_i}(s)\nabla_{\omega_i} W(s,\omega_1,\ldots,\omega_i)\Big|_{\omega_i=\mu_{i,\ldots,1}(s)} \qquad (50) \\
&= \mathbb{E}_{s\sim\rho^\mu}\Big[\nabla_{\theta_i}\mu_{\theta_i}(s)\nabla_{\omega_i} W(s,\omega_1,\ldots,\omega_i)\Big] \qquad (51)
\end{aligned}$$

□

B.3 POLICY GRADIENT ALGORITHMS

In this section, we present the algorithms for the hierarchical REINFORCE, REINFORCE with Baseline, actor-critic and DPG.

Algorithm 2: Hierarchical REINFORCE

Input: a set of differentiable policy parameterization $\pi(a \mid \omega_l), \pi(\omega_{i+1} \mid \omega_i)$ and $\pi(\omega_1 \mid s)$
Algorithm parameters: step size $\alpha_i > 0$
Initialize policy parameters θ_i

Loop forever (for each episode):
  Generate an episode $S_0, \Omega_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi_{\theta_i}(\cdot|\cdot)$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\theta_i \leftarrow \theta_i + \alpha \gamma^t G \nabla \ln \pi_{\theta_i}(\Omega_{i+1} \mid \Omega_i)$

Algorithm 3: Hierarchical REINFORCE with Baseline

Input: a set of differentiable policy parameterization $\pi(a \mid \omega_l), \pi(\omega_{i+1} \mid \omega_i)$ and $\pi(\omega_1 \mid s)$
Input: a set of strategy-value function parameterization $\hat{u}_{w_i}(s, \omega_i)$
Algorithm parameters: step size $\alpha_i^{\theta} > 0$, $\alpha_i^{w} > 0$
Initialize policy parameters θ_i, w_i

Loop forever (for each episode):
  Generate an episode $S_0, \Omega_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi_{\theta_i}(\cdot|\cdot)$
  Loop for each step of the episode $t = 0, 1, \ldots, T-1$ and for all i
    $G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k$
    $\delta_i \leftarrow G - \hat{u}_{w_i}(S_t, \Omega_t)$
    $w_i \leftarrow w_i + \alpha_i^{w} \delta_i \nabla \hat{u}_{\omega_i}(S_t, \Omega_t)$
    $\theta_i \leftarrow \theta_i + \alpha_i^{\theta} \gamma^t \delta_i \nabla \ln \pi_{\theta_i}(\Omega_{i+1} \mid \Omega_i)$

Algorithm 4: Hierarchical Actor-Critic

Input: a set of differentiable policy parameterization $\pi(a \mid \omega_l), \pi(\omega_{i+1} \mid \omega_i)$ and $\pi(\omega_1 \mid s)$
Input: a set of strategy-value function parameterization $\hat{u}_{w_i}(s, \omega_i)$
Algorithm parameters: step size $\alpha_i^{\theta} > 0$, $\alpha_i^{w} > 0$, update strength G_i
Initialize policy parameters θ_i, w_i

Loop forever (for each episode):
  Initialize S
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $\Omega_1 \sim \pi(\cdot \mid S), \ldots, \Omega_i \sim \pi(\cdot \mid |\Omega_{i-1}), \ldots, A \sim \pi(\cdot \mid \Omega_{l-1})$
    Take action A, observe S', G_i, R
    $\delta_i = G_i + \gamma \hat{u}_{w_i}(S', \Omega') - \hat{u}_{w_i}(S, \Omega)$
    $w_i \leftarrow w_i + \alpha_i^{w} \delta_i \nabla \hat{u}_{w_i}(S, \Omega)$
    $\theta_i \leftarrow \theta_i + \alpha_i^{\theta} I \ \delta_i \nabla \ln \pi_{\theta_i}(\Omega_{i+1} \mid \Omega_i)$
$I \leftarrow \gamma I$
$S \leftarrow S'$

Algorithm 5: Hierarchical Deterministic Policy Gradient

Input: a set of differentiable policy parameterization $\mu_{\theta_{l-1}}(\omega_{l-1}), \mu_{\theta_i}(\omega_i)$ and $\mu_{\theta_0}(s)$
Input: a set of strategy-value function parameterization $\hat{u}_{w_i}(s, \omega_i)$
Algorithm parameters: step size $\alpha_i^\theta > 0$, $\alpha_i^w > 0$, update strength G_i
Initialize policy parameters θ_i, w_i

Loop forever (for each episode):
  Initialize S
  $I \leftarrow 1$
  Loop while S is not terminal (for each time step):
    $\Omega_1 \leftarrow \mu_0(S), \ldots, \Omega_i \leftarrow \mu_{i-1}(\Omega_{i-1}), \ldots, A \leftarrow \mu_{l-1}(\Omega_{l-1})$
    Take action A, observe S', G_i, R
    $\delta_i = G_i + \gamma \hat{u}_{w_i}(S') - \hat{u}_{w_i}(S)$
    $w_i \leftarrow w_i + \alpha_i^w \delta_i \nabla \hat{u}_{w_i}(S)$
    $\theta_i \leftarrow \theta_i + \alpha_i^\theta I \, \nabla_{\theta_i} \mu_{\theta_i}(S) \nabla_{\omega_i} \hat{u}_{w_i}(S, \Omega_i)$
$I \leftarrow \gamma I$
$S \leftarrow S'$

C More on Parameters

In this appendix, we show the performance for various signs of the parameter g_2 when the absolute value is fixed $|g_2| = 5$. For simplicity, the value

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \mp 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \pm 5 \tag{52}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \mp 5 \tag{53}$$

are denoted as $(-++-)$ and $(+--+)$ therefore, one should note that $(-++-)$ represents the value chosen in the Fig.3c in the main part of this article and we use it together will the Q-learning results as the standard to compare with in the following experiments. Their performance is shown in Fig.7.

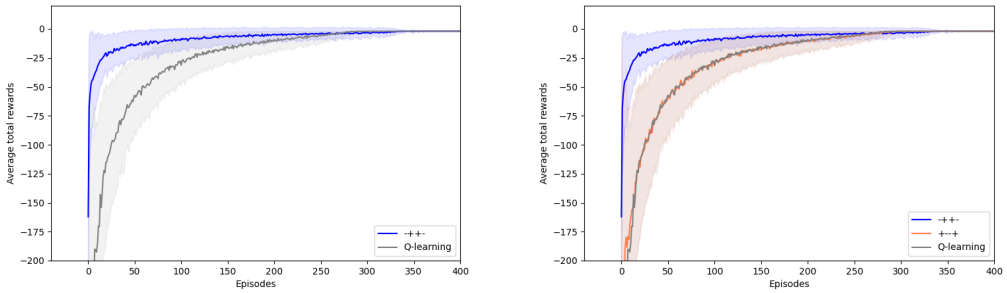


Figure 7: The performance of the training with g_1 taking the value of $(-++-)$ on the left and $(+--+)$ on the right. The reward is plotted by taking the average of 200 runs.

We denote the case where

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \pm 5 \tag{54}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \mp 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \mp 5 \tag{55}$$

as $(++--)$, $(--++)$ and the plot is shown in Fig.8.

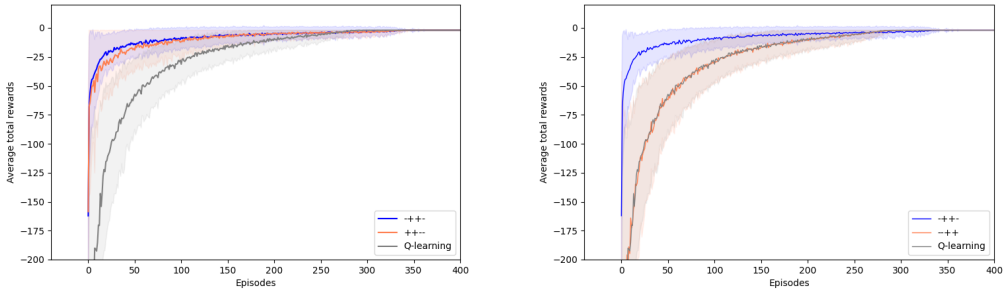


Figure 8: The performance of the training with g_1 taking the value of $(++--)$ on the left and $(--++)$ on the right. The reward is plotted by taking the average of 200 runs.

The value for

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \mp 5 \tag{56}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \mp 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \mp 5 \tag{57}$$

are denoted as $(+---)$ and $(-+++)$ with the result shown in Fig. 9.

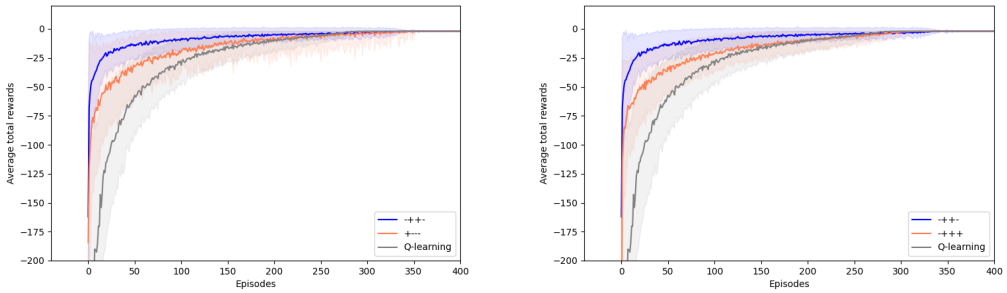


Figure 9: The performance of the training with g_1 taking the value of $(+---)$ on the left and $(-+++)$ on the right. The reward is plotted by taking the average of 200 runs.

The value for

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \mp 5 \tag{58}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \mp 5 \tag{59}$$

are denoted as $(+-+-)$, $(-+-+)$ and the results are presented in Fig.10.

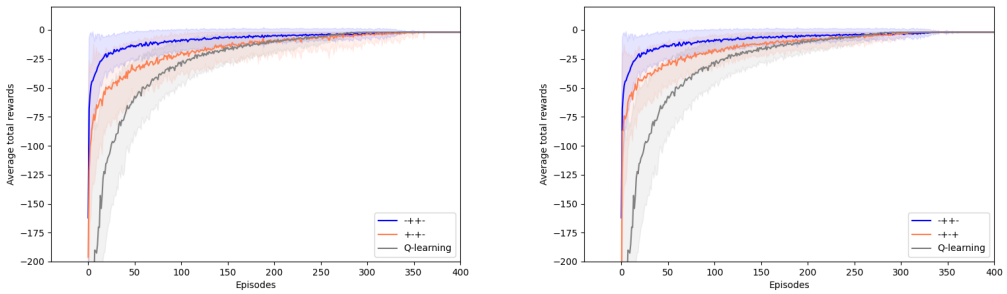


Figure 10: The performance of the training with g_1 taking the value of $(+-+-)$ on the left and $(-+-+)$ on the right. The reward is plotted by taking the average of 200 runs.

The value for

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \pm 5 \tag{60}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \mp 5 \tag{61}$$

are denoted as $(++\!+\!-)$ and $(---\!+)$ with the performance illustrated in Fig.11.

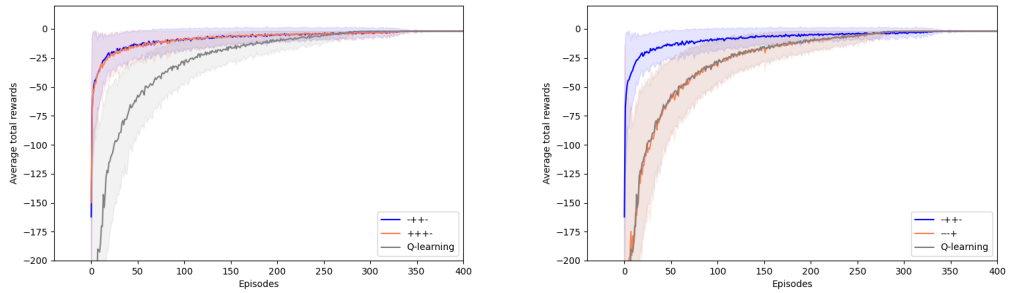


Figure 11: The performance of the training with g_1 taking the value of $(++\!+\!-)$ on the left and $(---\!+)$ on the right. The reward is plotted by taking the average of 200 runs.

The value for

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \mp 5 \tag{62}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \pm 5 \tag{63}$$

are denoted as $(+-\!++)$, $(-+\!--)$ and the performance is presented in Fig.12.

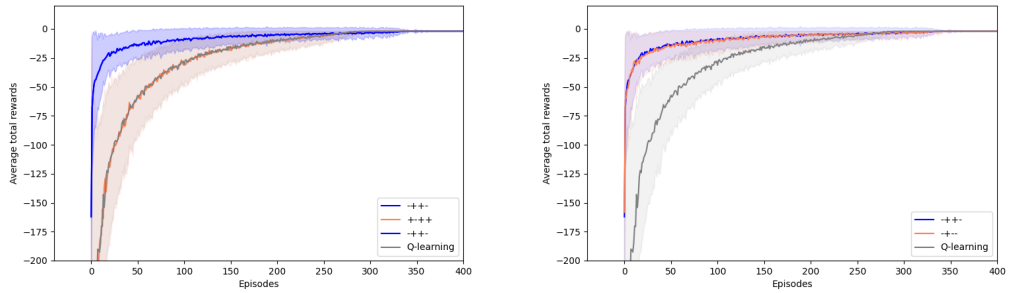


Figure 12: The performance of the training with g_1 taking the value of $(+-\!++)$ on the left and $(-+\!--)$ on the right. The reward is plotted by taking the average of 200 runs.

The value for

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \pm 5 \tag{64}$$
$$g_1(\omega_1 = \mathrm{II}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \pm 5 \tag{65}$$

are denoted as $(++\!++)$, $(---\!-)$ and the performance is shown in Fig.13.

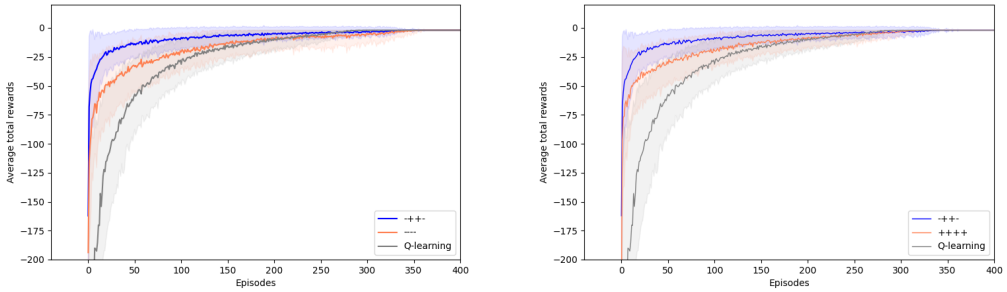


Figure 13: The performance of the training with g_1 taking the value of $(--\,--)$ on the left and $(+\,+\,+\,+)$ on the right. The reward is plotted by taking the average of 200 runs.

The case where

$$g_1(\omega_1 = \mathrm{I}, r = -10) = \pm 5 \qquad g_1(\omega_1 = \mathrm{I}, r = -2) = \pm 5 \tag{66}$$

$$g_1(\omega_1 = \mathrm{II}, r = -10) = \mp 5 \qquad g_1(\omega_1 = \mathrm{II}, r = -2) = \pm 5 \tag{67}$$

are denoted as $(+\,+\,-\,+)$ and $(-\,-\,+\,-)$ with the results shown in Fig.14.

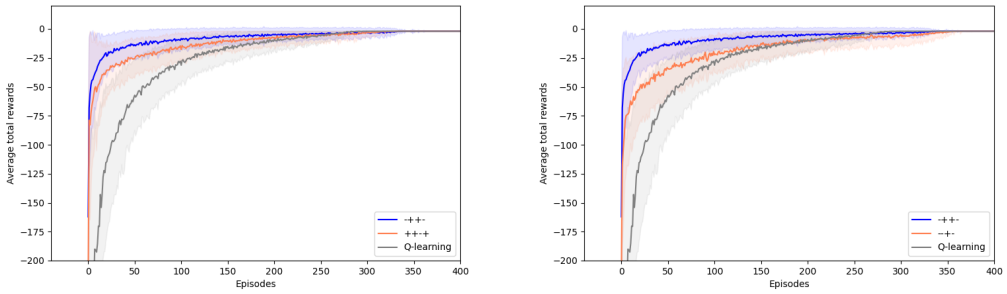


Figure 14: The performance of the training with g_1 taking the value of $(+\,+\,-\,+)$ on the left and $(-\,-\,+\,-)$ on the right. The reward is plotted by taking the average of 200 runs.

D State Dependent Strategy

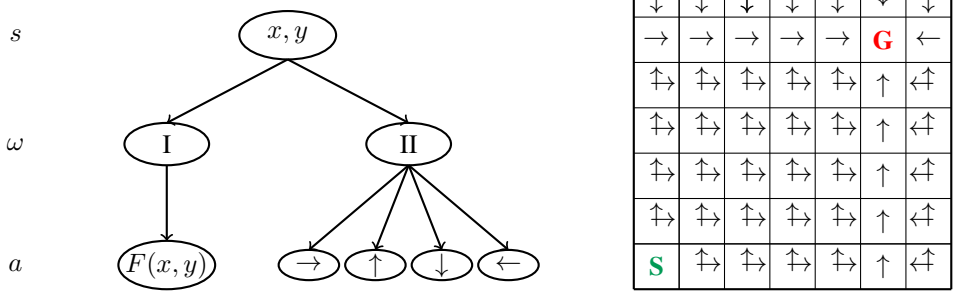


Figure 15: The hierarchical structure is shown on the left figure where the state dependent strategy $F(x, y)$ is illustrated by the figure on the right.

In this appendix, we consider the case where the goal is not located in the corner; therefore we need to introduce a state dependent policy. In our case, the goal is set at $(6, 6)$. To construct the strategy, we consider the optimal policy without the obstacles and denote it as $F(x, y)$ then the hierarchical

structure of the strategy is shown in Fig.15 where the figure on the left represents the strategy and the right figure shows the detail of the function $F(x, y)$.

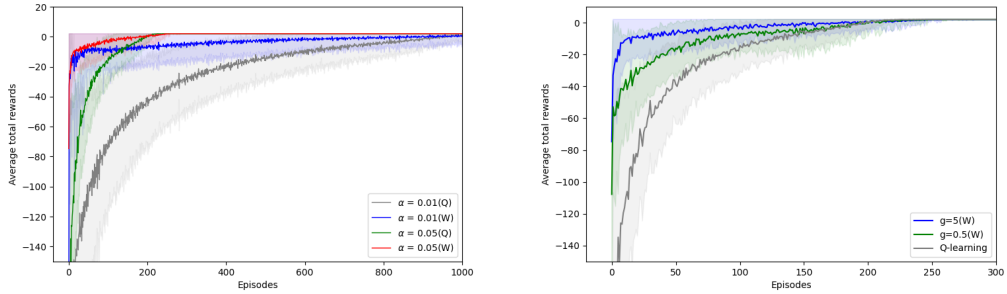


Figure 16: Total rewards for the W and Q learning with learning rate $\alpha = 0.05$ and $\alpha = 0.01$ are shown on the left while the results for W learning is illustrated in the right figure in the case $|g_1| = 5$ and $|g_1| = 0.5$. The reward is plotted by taking the average of 200 runs.

Given the strategy, we first plot the reward for the Q and W learning and put the result together shown on the left hand side of Fig.16. One can see the performance for the W learning is similar to the previous case when the goal is set at $(7, 7)$ and Q-learning shows slight improvement since the distance between the starting point and the goal (now at (6, 6)) becomes shorter. Moreover, we also present the results for different update parameters $|g_1| = 0.5$ in the right figure, showing that the agent tends to find out the optimal policy more quickly if the $|g_1|$ takes bigger value.

E DOUBLE W AND N-STEP W-LEARNING

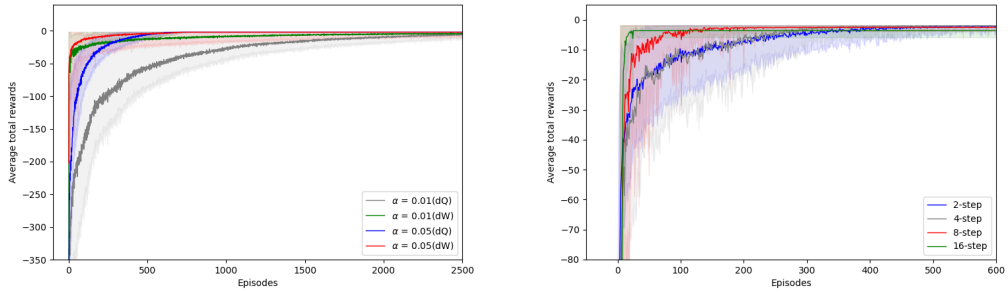


Figure 17: Double Q and W learning results for $\alpha = 0.01$, $\alpha = 0.05$ are shown on the left while the 2, 4, 8, 16-step learning results for W-learning are shown on the right. The reward is plotted by taking the average of 200 runs.

In this section, we present the results of the double learning and n-step learning for our original 7-by-7 gridworld task in Fig.17, from which we can see that double W-learning tends to learn faster than double Q-learning under the same learning rate. The results for 2, 4, 8, 16 - step W-learning are presented on the right with learning rate $\alpha = 0.05$. From the figure we can see that there is no significant improvement when the learning step is small, for example, the 2-step and 4-step learning. When the learning step becomes large, the learning speed will increase while the performance will be less robust.

Furthermore, we present the 2, 4, 8, 16-step Q-learning and W-learning results. For the 2-step case, we can see that there is an increase in learning speed compared with the 1-step Q-learning, and there is no significant improvement for the W-learning. But 2-step W-learning still outperforms 2-step Q-learning. For the 4-step learning case, there is continuous improvement for the 4-step Q-learning and still no improvement for the 4-step W-learning. The learning speed of 4-step Q and W-learning becomes similar.

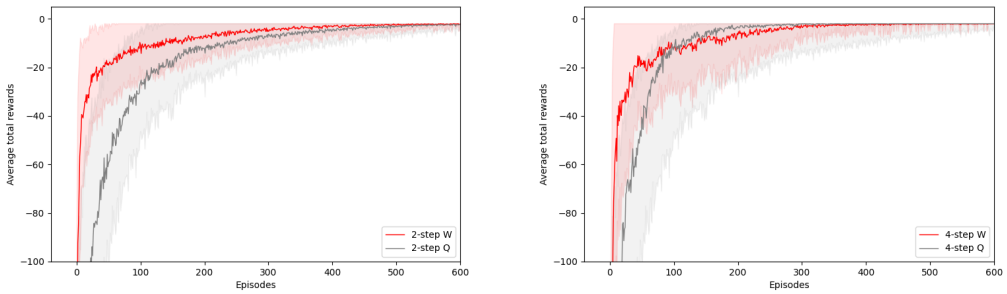


Figure 18: The 2-step and 4-step Q and W-learning results are shown in the figure, in which W-learning is represented by the red curves and Q-learning is represented by the gray ones. The reward is plotted by taking the average of 200 runs.

For the 8-step case, both of the 8-step Q and W-learning experience a speedup and their performance becomes similar. For the 16-step case, there is an increase in learning speed for 16-step Q and W-learning but we can see that Q-learning becomes less stable than the W-learning. In general, when the number of steps is increased during training, more episodes occur in which the agent fails to identify the optimal policy. Overall, W-learning appears more robust than Q-learning.

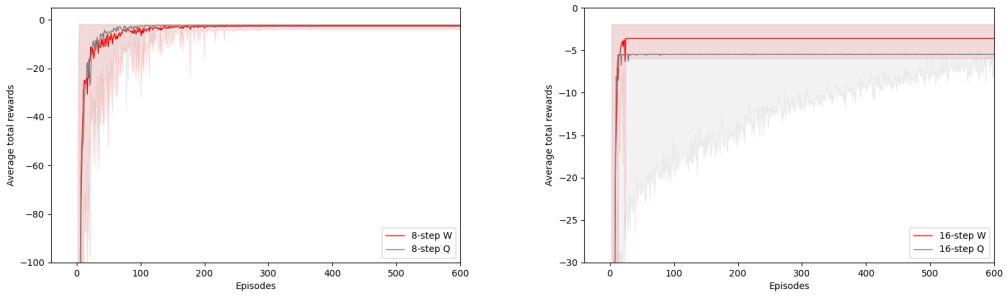
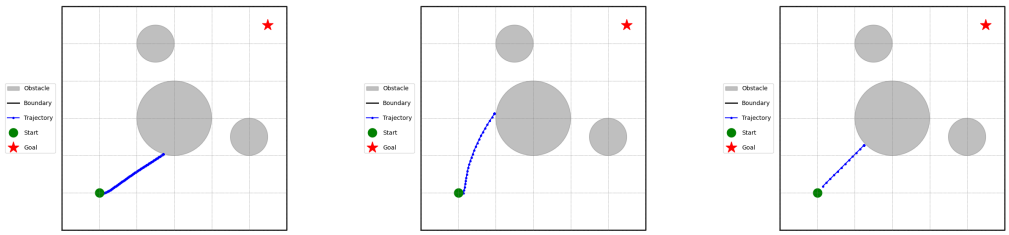


Figure 19: The 8-step and 16-step Q and W-learning results are shown in the figure, in which W-learning is represented by the red curves and Q-learning is represented by the gray ones. The reward is plotted by taking the average of 200 runs.

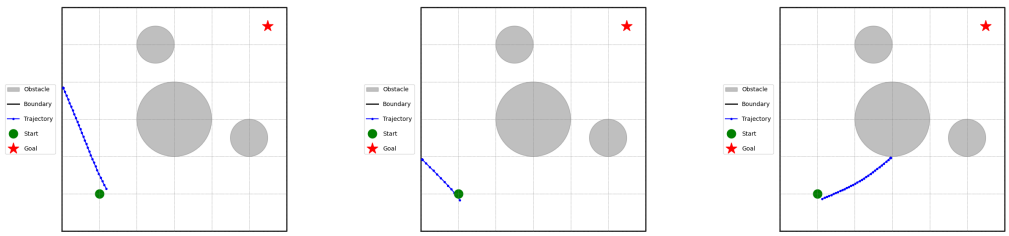
F Experimental Results on Continuous Navigation

In this appendix, we will present more results on the continuous navigation task. For the standard DPG algorithm, during the experiment, we found it is hard for the agent to reach the goal successfully before reaching the maximum number of steps. As we can see from Fig.20, for most of the training episodes, the agent tends to become trapped near the obstacles or the boundary. DPGv2 is an improved version of DPG in which we make the agent learn the direction first, then choose the action from $[0, 1]$ or $[-1, 0]$. For example, if the agent decides to go right, then the x component of the action takes a value in $[0, 1]$ determined by DPG and the y component will be 0. It turns out that there were more successful episodes during training for DPGv2 but it is still hard for the agent to reach the goal efficiently. For most episodes, the agent tends to continue exploring regions far from the goal.

Given the strategy, we have the corresponding method HDPG and HDPGv2, and it turns out that the agent will reach the goal in fewer steps under the guidance of such strategy as shown in Fig.22 and Fig.23. We adopted a state dependent strategy in HDPG like the discrete case shown in Fig. 15 and a state independent strategy for HDPGv2.

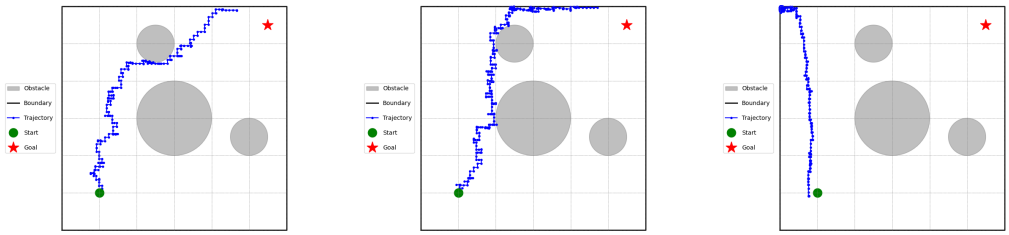


(a) DPG-1 (b) DPG-2 (c) DPG-3

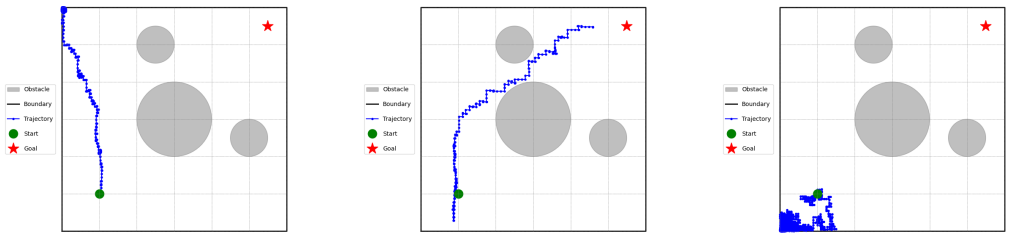


(d) DPG-4 (e) DPG-5 (f) DPG-6

Figure 20: Learning results for DPG

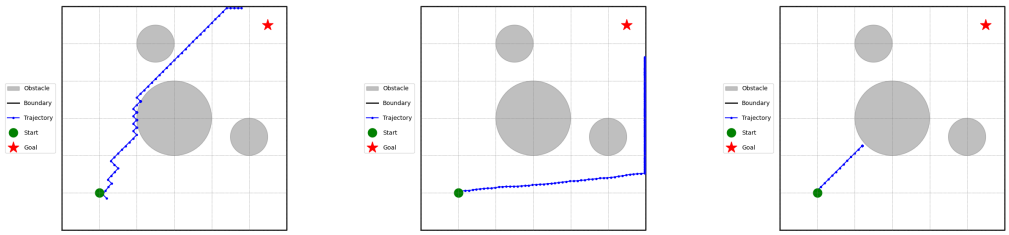


(a) DPGv2-1 (b) DPGv2-2 (c) DPGv2-3

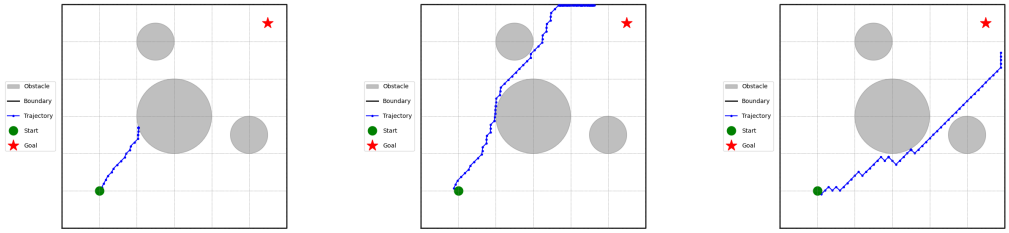


(d) DPGv2-4 (e) DPGv2-5 (f) DPGv2-6

Figure 21: Learning results for DPGv2

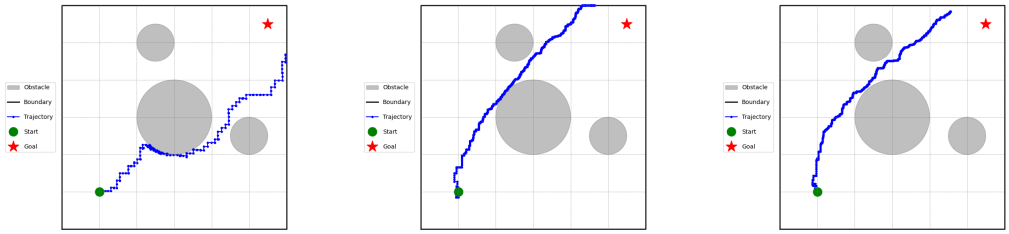


(a) HDPG-1 (b) HDPG-2 (c) HDPG-3

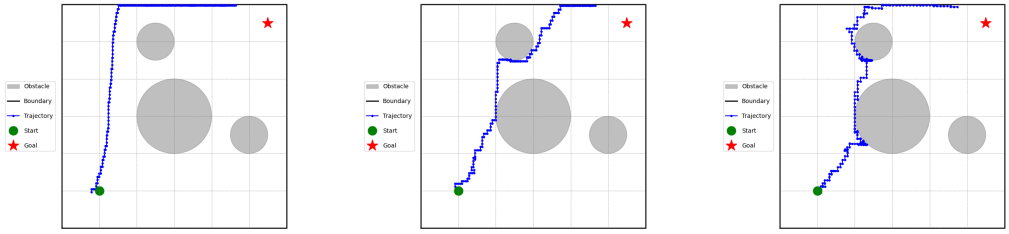


(d) HDPG-4 (e) HDPG-5 (f) HDPG-6

Figure 22: Learning results for HDPG



(a) HDPGv2-1 (b) HDPGv2-2 (c) HDPGv2-3



(d) HDPGv2-4 (e) HDPGv2-5 (f) HDPGv2-6

Figure 23: Learning results for HDPGv2

G HYPERPARAMETERS

Parameter	Value
Environment set up	
size	6×6
agent's initial position	$(1, 1)$
center of the goal with radius 1	$(5.5, 5.5)$
center of the obstacle with radius 1	$(3, 3)$
center of the obstacle with radius 0.5	$(5, 2.5), (2.5, 5)$
distance between the agent and the goal's center	d
clip for the action a	randomly chosen from $[0, 0.2]$
Rewards	
reach the goal	2000
hit the obstacles	-2
otherwise	$-2 \times d$
Training Parameters	
optimizer	Adam
actor learning rate	1×10^{-5}
critic learning rate	1×10^{-6}
discount(γ)	0.99
number of hidden layers (Actor Critic)	2
number of hidden units per layer (Actor Critic)	128
number of hidden layers (DPG)	3
number of hidden units per layer (DPG)	64
nonlinearity	ReLU
critic update frequency	10