Incremental Reinforcement Learning in Continuous Spaces via Policy Relaxation and Importance Weighting

Zhi Wang, *Student Member, IEEE*, Han-Xiong Li, *Fellow, IEEE,* and Chunlin Chen, *Member, IEEE*

***Abstract*— In this paper, a systematic incremental learning method is presented for reinforcement learning in continuous spaces where the learning environment is dynamic. The goal is to adjust the previously learned policy in the original environment to a new one incrementally whenever the environment changes. To improve the adaptability to the ever-changing environment, we propose a two-step solution incorporated with the incremental learning procedure: policy relaxation and importance weighting. First, the behavior policy is relaxed to a random one in the initial learning episodes to encourage a proper exploration in the new environment. It alleviates the conflict between the new information and the existing knowledge for a better adaptation in the long term. Second, it is observed that episodes receiving higher returns are more in line with the new environment, and hence contain more new information. During parameter updating, we assign higher importance weights to the learning episodes that contain more new information, thus encouraging the previous optimal policy to be faster adapted to a new one that fits in the new environment. Empirical studies on continuous controlling tasks with varying configurations verify that the proposed method achieves a significantly faster adaptation to various dynamic environments than the baselines.**

***Index Terms*— Continuous spaces, dynamic environments, importance weighting, incremental reinforcement learning (RL), policy relaxation.**

I. Introduction

REINFORCEMENT learning (RL) [1] addresses the problem of how an autonomous active agent can learn to approximate an optimal behavioral policy that maps states to actions to maximize the long-term cumulative reward while interacting with its environment in a trial-and-error manner. Traditional RL algorithms, such as dynamic programming [2], Monte Carlo methods [3], and temporal difference learning [4], have been widely used in intelligent control and industrial applications [5]–[8]. To overcome the "curse of dimensionality," function approximation techniques, such as least-squares policy iteration [9] and fitted Q-iteration [10], [11], are usually employed for Markov decision processes (MDPs) with continuous spaces. The recent development of combining advances in deep learning for learning feature representations makes RL algorithms practical for extremely high-dimensional applications, such as Atari games [12], the game of Go [13], and robot locomotion [14].

In the conventional RL setting, a stationary task is considered, where the environment remains unchanged during the entire learning process. However, in real-world applications, the environments are often dynamic, where the reward functions, state transition functions, or state-action spaces may change over time, such as for robot navigation [15] and multi-agent RL (MARL) problems [16]. Transfer RL [17], [18] can circumvent the necessity for repeatedly retraining the learning system by transferring the experience gained in a set of source tasks to help the learning performance in a related but different task. However, it requires repeatedly accessing and processing a potentially very large set of source tasks to provide a good knowledge base for the downstream target task.

Throughout this paper, we consider the dynamic environment as a sequence of stationary tasks on a certain timescale where each task that we target to solve corresponds to the specific environment that is stationary during the associated time period. As intelligent agents are becoming more ubiquitous nowadays, an increasing number of real-world scenarios requires new learning mechanisms that are amenable for a fast adaptation to environments that may drift or change from their nominal situations. Many of today's data-intensive computing applications require the autonomous RL agent to be capable of adapting its behavior in an incremental manner as the environment changes around it, continuously utilizing previous knowledge to benefit the future decision-making process. Thus, the concept of "incremental learning" emerges by incrementally adjusting the previously learned policy to a new one that fits in the new environment whenever the environment changes, which offers an appealing alternative for a fast adaptation to dynamic environments. Such an incremental adaptation is crucial for the intelligent systems operating in the real world, where the changing factors and unexpected perturbations are the norm.

Manuscript received January 2, 2019; revised May 4, 2019; accepted July 3, 2019. Date of publication August 2, 2019; date of current version June 2, 2020. This work was supported in part by the General Research Fund Project from the Research Grant Council of Hong Kong SAR under Grant CityU 11210719, in part by the Project from the City University of Hong Kong under Grant 7005092, in part by the National Key Research and Development Program of China under Grant 2016YFD0702100, and in part by the Fundamental Research Funds for the Central Universities under Grant 011814380035. *(Corresponding author: Han-Xiong Li.)*

Z. Wang and C. Chen are with the Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University, Nanjing 210093, China (e-mail: njuwangzhi@gmail.com; clchen@nju.edu.cn).

H.-X. Li is with the Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong, and also with the State Key Laboratory of High Performance Complex Manufacturing, Central South University, Changsha 410083, China (e-mail: mehxli@cityu.edu.hk).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2927320

Although incremental learning has been widely investigated for decades in the machine learning and data mining community [19]–[24], a few results are reported until recently regarding incremental learning for RL problems. In the RL community, Wang *et al.* [25] first proposed an incremental learning algorithm for RL in dynamic environments where the reward functions might change over time. Nevertheless, the incremental algorithm in [25] only worked in RL problems with a discrete state-action space, since it involved a tabular form of comparing reward functions and the prioritized sweeping process. For RL in continuous spaces, we mainly focus on methods that explicitly update a representation/approximation of a value function, a policy, or both [26]. The design of a feasible incremental learning method should be capable of incorporating with the function approximation framework.

In the incremental learning setting [25], when the environment changes, one can directly initialize the parameters of a function approximation from the previous optima that have learned some of feature representations (e.g., nodes in a neural network) of the state-action space in the original environment.[1] However, the previous optimum of parameters may be a local one that has been overfitted to the original environment, especially when using a nonlinear function approximator such as the neural network. When updating parameters by interacting with the new environment, the learning agent tends to generate policies that perform well in the original environment and may not discover alternative policies with potentially larger payoffs, i.e., it can get stuck in local optima. Therefore, directly learning based on the existing knowledge may hinder the RL agent to properly explore and further adapt to the new environment. For example, in a navigation task, a robot has learned how to reach a goal in the south direction. When the goal changes to the north, the robot still tends to head south before it can slowly adapt to the new environment.

This paper investigates the incremental RL problem in continuous spaces, which attempts to achieve a fast adaptation to dynamic environments. After initializing parameters from the original environment, we propose to improve the adaptability to the new environment by using a two-step solution: policy relaxation and importance weighting. In the first step, the learning agent is forced to follow a relaxed random policy for a small number of episodes to properly explore the new environment. Intuitively, policy relaxation alleviates the conflict between the new information and the existing knowledge. In the second step, we observe that episodes receiving higher returns are supposed to be more in line with the new environment, and hence contain more new information. During parameter updating, we assign higher importance weights to the episodes containing more new information, thus encouraging the previous optimal policy to be faster adjusted to a new one that fits in the new environment. A good property of the formulated incremental learning procedure is that only the learned function approximation is needed for the new environment, circumventing the necessity for repeatedly accessing, or processing a potentially large set of source tasks.

We use the policy search algorithm with a nonlinear function approximation [26], [29] to handle challenging RL tasks with continuous state-action spaces. The policy is represented by a (deep) neural network, and its parameters are optimized using the gradient descent method. Experiments are conducted under various dynamic environments on continuous control tasks ranging from classical 2-D navigation to complex MuJoCo robot locomotion. The results show that the proposed method achieves a significantly faster adaptation to various dynamic environments than the baselines. In summary, the contribution of this paper lies in the following aspects.

1) We introduce a systematic incremental learning method for RL in continuous spaces where the learning environment is dynamic.
2) We propose a policy relaxation mechanism to encourage the learning agent to properly explore the new environment for a better adaptation in the long term.
3) We incorporate an importance weighting mechanism with the policy iteration process to encourage a faster adaptation to dynamic environments.

The rest of this paper is organized as follows. Section II presents the research background including RL in continuous spaces and the related work. In Section III, the framework of the proposed method is presented, followed by specific mechanisms and implementation details. Experiments on several classical 2-D navigation tasks and complex Mujoco locomotion tasks are conducted in Section IV. Section V presents concluding remarks.

II. Background

A. Reinforcement Learning in Continuous Spaces

1) Markov Decision Process: RL is commonly studied based on the MDP framework. An MDP is a tuple $\langle S, A, T, R, \gamma \rangle$, where S is the set of states, A is the set of actions, $T : S \times A \times S \to [0, 1]$ is the state transition probability, $R : S \times A \to \mathbb{R}$ is the reward function, and γ is the discounting factor. A policy is defined as a function $\pi : S \times A \to [0, 1]$, a probability distribution that maps actions to states, and $\sum_{a \in A} \pi(a|s) = 1, \forall s \in S$. The goal of RL is to find an optimal policy π^* that maximizes the expected long-term return $J(\pi)$

$$J(\pi) = \mathbb{E}_{\tau \sim \pi(\tau)}[r(\tau)] = \mathbb{E}_{\tau \sim \pi(\tau)} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \tag{1}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is the learning episode, $\pi(\tau) = p(s_0) \Pi_{t=0}^{\infty} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t)$, r_t is the instant reward received when executing the action a_t in the state s_t.

2) Policy Gradient: Following the state-of-the-art benchmark [14], we employ the policy gradient approach to handle challenging tasks with continuous state and action spaces. Policy gradient is a branch of RL algorithms that approximate and learn the policy directly by maximizing the return. The policy is often represented as a parameterized approximation $\pi_{\boldsymbol{\theta}}$ using a function $f(\cdot|\boldsymbol{\theta})$. In deep RL (DRL), f is a

[1] Similar to the case in the computer vision community [27], where features of neural networks learned on the common ImageNet data set can provide an empirical initialization for helping the general downstream tasks; or the case in the natural language processing community [28], where the pretrained word embeddings from a large source corpora have an outsized impact on practice and are used in most state-of-the-art models.

deep neural network (DNN) and $\boldsymbol{\theta}$ denotes the weights of the network. The Gibbs distribution is commonly used for a discrete action space

$$\pi_{\boldsymbol{\theta}}(i|s) = \frac{\exp(f_i(s|\boldsymbol{\theta}))}{\sum_{j\in A(s)} \exp(f_j(s|\boldsymbol{\theta}))} \tag{2}$$

and the Gaussian distribution is usually used for a continuous action space

$$\pi_{\boldsymbol{\theta}}(a|s) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{\sigma^2}(f(s|\boldsymbol{\theta}) - a)^2\right). \tag{3}$$

To measure the quality of the policy π, the direct objective function can be equivalently rewritten as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}(\tau)}[r(\tau)] = \int_{\tau} \pi_{\boldsymbol{\theta}}(\tau) r(\tau)\, \mathrm{d}\tau \tag{4}$$

where $r(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ is the return of episode τ.

Policy gradient searches for a local maximum by ascending the parameters following the gradient of the policy with respect to the expected return. By the policy gradient theorem [1], [30], the basic policy gradient method employs the direct gradient of the objective

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}(\tau)}\left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) r(\tau)\right] \\ &= \int_{\tau} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) r(\tau) \pi_{\boldsymbol{\theta}}(\tau)\, \mathrm{d}\tau \end{aligned} \tag{5}$$

and then the parameters $\boldsymbol{\theta}$ are updated iteratively by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{6}$$

until it converges.

B. Related Work

Traditional RL algorithms focus on a fixed task with a stationary environment. However, we often face the problem of learning in a dynamic environment in practice [25]. For example, the geographic feature in a navigation task may change slightly over time, or the mission to be accomplished by a humanoid robot can vary for different applications. In these cases, the learning agent needs to adjust the previous behavior policy incrementally to adapt to the ever-changing environment.

Regarding dynamic environments, a particularly related setting is the transfer RL [17] as shown in Fig. 1(a). As a combination of transfer learning [18] and RL, it addresses the cross-task generalization problem, which reuses the knowledge from a set of related source domains to help the learning task in the target domain. Different types of knowledge can be transferred, such as sample instances [31], policies [32], and value functions [33]. With the advance in DRL, the researchers focus more on how to transfer the knowledge with the use of DNNs [34]–[36]. Other related settings include the multi-task RL [37], [38] and the meta RL [29], [39]. As shown in Fig. 1(b), multi-task RL aims to solve a fixed set of tasks simultaneously based on the assumption that the tasks share some similarities in components such as the reward structure, the transition dynamics, or the value function. Meta RL is a very new issue that has been studied recently. It trains a model on a variety of learning tasks to provide a good initialization, as called "meta," for general unseen tasks.

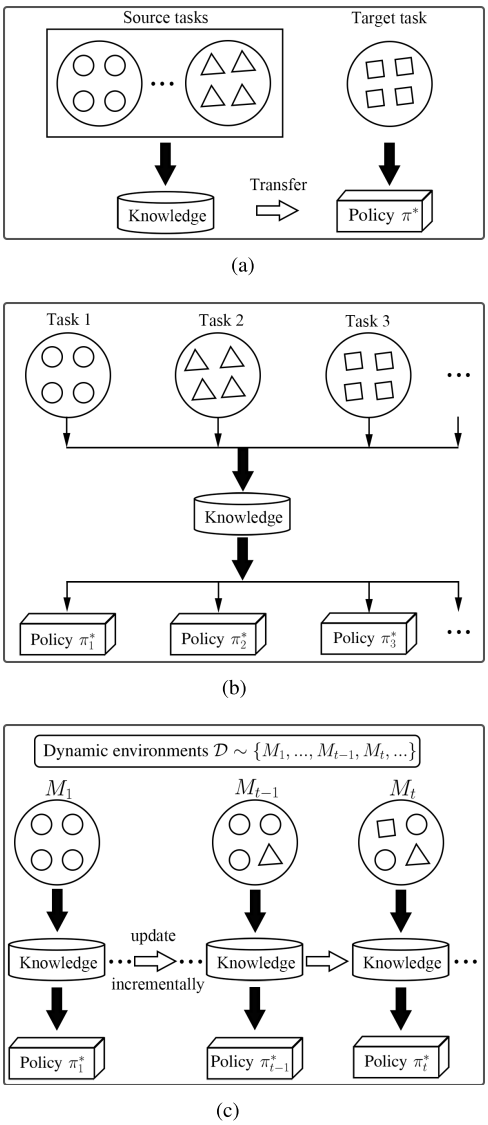


Fig. 1. Intuitive illustration of (a) transfer RL, (b) multi-task RL, and (c) incremental RL.

We note that the above-related work requires repeatedly accessing and processing a potentially very large set of source tasks to provide a good knowledge base for the downstream target task. Hence, we argue that incremental learning is a more feasible alternative for handling the learning adaptation in dynamic environments. As shown in Fig. 1(c), incremental RL continually adjusts the previously learned policy to a new one whenever the environment changes. Incremental learning has been widely addressed in machine learning and intelligent control communities to cope with learning tasks where the training samples become available over time or the learning environment is ever-changing [24], including unsupervised learning [19], supervised learning [40], machine vision [21], evolutionary algorithms [22], system modeling [41], and human–robot interaction [23]. In the RL community, Wang *et al.* [25] first proposed an incremental learning algorithm for dynamic environments where the reward functions might change over time. However, it involved a

tabular form of comparing the reward functions and the prioritized sweeping process, and hence could only be applied for RL problems with a discrete state-action space. In this paper, we aim at designing a feasible incremental learning method that can incorporate the function approximation framework in continuous spaces.

III. Incremental Reinforcement Learning in Continuous Spaces

In this section, we first formulate the incremental RL problem in continuous spaces under a dynamic environment. Then, we introduce the proposed two mechanisms of policy relaxation and importance weighting in turn. Finally, we give the integrated algorithm based on the above-mentioned implementations.

A. Problem Formulation

We consider the dynamic environment as a sequence of stationary tasks on a certain timescale where each task corresponds to the specific type of environment characteristics during the associated time period. Assume there is a space of MDPs, $\mathcal{M}$, and an infinite underlying distribution, $\mathcal{D}$, over time in $\mathcal{M}$. An RL agent interacts with the dynamic environment $\mathcal{D} = \{M_1, \ldots, M_{t-1}, M_t, \ldots\}$, where each $M_t \in \mathcal{M}$ denotes the specific MDP that is stationary during the tth time period. We assume, in this paper, that the environment changes only in the reward and state transition functions, but keeps the same state and action spaces.

Suppose that during the $(t-1)$th time period, the agent learned the optimal parameters $\boldsymbol{\theta}^*_{t-1}$ with respect to the policy approximation

$$\boldsymbol{\theta}^*_{t-1} = \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^d} J_{M_{t-1}}(\pi_{\boldsymbol{\theta}}). \tag{7}$$

When the environment changes to M_t at the tth time period, the goal of incremental learning is to adjust the existing knowledge of $\boldsymbol{\theta}^*_{t-1}$ to a new one $\boldsymbol{\theta}^*_t$ that can achieve a maximum return in the new environment

$$\boldsymbol{\theta}^*_t = \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^d} J_{M_t}(\pi_{\boldsymbol{\theta}}) \tag{8}$$

with initialization of $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}^*_{t-1}$. Continually, the agent incrementally adjusts the existing knowledge to a new one, $(\boldsymbol{\theta}^*_{t+1}, \boldsymbol{\theta}^*_{t+2}, \ldots)$, whenever the environment changes.

Initializing policy parameters from the previous optimum empirically benefits the learning process when starting to interact with the new environment. However, the previous optimum may be a local one that has been overfitted to the original environment, particularly when using a nonlinear function approximator such as the (deep) neural network. This potential drawback in the incremental initialization may degrade the performance of the new learning process in the long term. Unlike in supervised learning with DNNs, wherein local optima are not thought to be a problem [42], the training data in RL are determined by the actions an agent takes. In particular, in the on-policy case, the learning agent can only use samples consistent with the current policy that is being executed [43]. When the agent updates parameters from the previous optimum, the training data for the algorithm will be limited to those generated by policies that perform well in the original environment. Thus, it may not properly explore the new environment to discover alternate policies with potentially larger payoffs, i.e., it can get stuck in local optima.

From the above insight, we design a two-step solution incorporated with the incremental learning procedure as: 1) using policy relaxation to first encourage a proper exploration for a better adaptation in the long term and 2) using importance weighting to assign higher weights to episodes that contain more new information, thus further encouraging the previous optimal policy to be faster adapted to a new one that fits in the new environment. The specific implementations are introduced in Sections III-B–III-D.

B. Policy Relaxation

In the new environment M_t, the agent tends to visit a small part of the whole state-action space when executing the previously learned policy, thus probably leading to a local optimum due to insufficient exploration. Hence, we propose a policy relaxation mechanism to encourage a proper exploration. Specifically, in the k burn-in learning episodes, the agent is forced to execute a relaxed policy where actions are randomly selected from the available set. For better readability, let $\boldsymbol{\theta}$ denote the current parameters in M_t, and $\pi_{\boldsymbol{\theta}}$ be the policy derived from $\boldsymbol{\theta}$. Regarding the number of learning episodes η, the agent's behavior policy π_r is relaxed as

$$\pi_r(a|s) = \begin{cases} \text{Uniform}(A(s)), & \eta \le k \\ \pi_{\boldsymbol{\theta}}(a|s), & \eta > k \end{cases} \tag{9}$$

where $A(s)$ is the set of discrete/continuous available actions in the state s, Uniform$(\cdot)$ is the sampling function from a uniform distribution, and k is a predetermined number of burn-in episodes.

However, in the k burn-in episodes, we would encounter the special difficulty due to a mismatch of distributions. We would like samples drawn from the distribution of the estimated policy $\pi_{\boldsymbol{\theta}}$ but we have only samples drawn from the distribution of another behavior policy π_r. This will lead to an unfavorable bias compared to the conventional RL setting [44]. To make the policy relaxation mechanism feasible, we adopt the classical Monte Carlo technique [45], importance sampling [46], to handle this kind of mismatch.

Recall the goal of RL in (1), and $r(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$ is the received return of episode τ. In the k burn-in episodes with the relaxed policy, we need to estimate the expectation of $r(\tau)$ under the distribution of $\tau \sim \pi_{\boldsymbol{\theta}}(\tau)$, while we are given samples from a different behavior policy $\tau \sim \pi_r(\tau)$. The observation of importance sampling is

$$\begin{aligned} \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)}[r(\tau)] &= \int_{\tau} \pi_{\boldsymbol{\theta}}(\tau) r(\tau)\, \mathrm{d}\tau \\ &= \int_{\tau} \frac{\pi_{\boldsymbol{\theta}}(\tau)}{\pi_r(\tau)} r(\tau) \pi_r(\tau)\, \mathrm{d}\tau \\ &= \mathbb{E}_{\tau \sim \pi_r(\tau)} \left[\frac{\pi_{\boldsymbol{\theta}}(\tau)}{\pi_r(\tau)} r(\tau) \right]. \end{aligned} \tag{10}$$

Given an episode τ generated by π_r, the return regarding the current parameters $\boldsymbol{\theta}$ should be multiplied with a ratio,

Algorithm 1 Policy Relaxation With Importance Sampling

Input: Number of burn-in episodes k;
learning rate α; batch size m
Output: Optimal policy parameters $\boldsymbol{\theta}^*$

Initialize the number of learning episodes: $\eta \leftarrow 0$
while *not converged* **do**
  if $\eta \leq k$ **then**
    $\pi_r(a|s) = \text{Uniform}(A(s)), \forall s$
    Sample m episodes from π_r: $\tau^i \sim \pi_r$
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \frac{\pi_{\boldsymbol{\theta}}(\tau^i)}{\pi_r(\tau^i)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau^i) r(\tau^i)$
  else
    Sample m episodes from $\pi_{\boldsymbol{\theta}}$: $\tau^i \sim \pi_{\boldsymbol{\theta}}$
    $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau^i) r(\tau^i)$
  end
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  $\eta \leftarrow \eta + m$
end

$\pi_{\boldsymbol{\theta}}(\tau)/\pi_r(\tau)$, to ensure that the estimated policy is unbiased compared to the conventional RL setting. In the policy gradient step, the gradient ascent rule in (5) is rewritten as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau\sim\pi_r(\tau)}\left[\frac{\pi_{\boldsymbol{\theta}}(\tau)}{\pi_r(\tau)}\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) r(\tau)\right] \\ &= \mathbb{E}_{\tau\sim\pi_r(\tau)}\left[\sum_{t=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t)\left(\prod_{t'=0}^{t} \frac{\pi_{\boldsymbol{\theta}}(a_{t'}|s_{t'})}{\pi_r(a_{t'}|s_{t'})}\right)\right. \\ &\quad \left.\times\left(\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}\right)\right]. \end{aligned} \tag{11}$$

Together, the implementation of policy relaxation with importance sampling is presented in Algorithm 1.

Remark 1: Policy relaxation is in a similar spirit to the classical ϵ-greedy exploration strategy in the way that the behavior policy is allowed to generate random actions to collect samples, which, in turn, are utilized to update the learning system (e.g., a value function, a policy function, or both). The ϵ-greedy technique is usually employed in value function-based RL algorithms such as the classical Q-learning [4] and the state-of-the-art deep Q-network [12]. In this paper, we adopt the spirit of ϵ-greedy to encourage a proper exploration for the implemented policy gradient approach. Moreover, we employ the importance sampling technique to make such an exploration scheme feasible under the policy-based RL architecture.

C. Importance Weighting

The policy initialized from the original environment empirically achieves a moderate performance in the new environment, especially after a very small number of update steps, because the previous optimum of policy parameters has learned some of feature representations (e.g., nodes in a neural network) of the state-action space. Fig. 2 shows a simple example of the 2-D navigation task in a dynamic environment where the goal changes. In Fig. 2(a), the episode τ_{t-1}, receiving the highest return, is generated by the learned optimal policy $\pi_{\boldsymbol{\theta}^*_{t-1}}$ in the original environment M_{t-1}. Then, during the tth period shown in Fig. 2(b), the environment changes to M_t where the goal moves to a new position. Suppose that τ_r is one learning episode generated by a randomly initialized policy π_{ran}, and τ_t^1 and τ_t^2 are two episodes generated by the current policy $\pi_{\boldsymbol{\theta}_t}$ that is initialized from $\pi_{\boldsymbol{\theta}^*_{t-1}}$. Consistent with the above-mentioned observation, episodes generated by $\pi_{\boldsymbol{\theta}_t}$ tend to receive higher returns than those generated by π_{ran}.

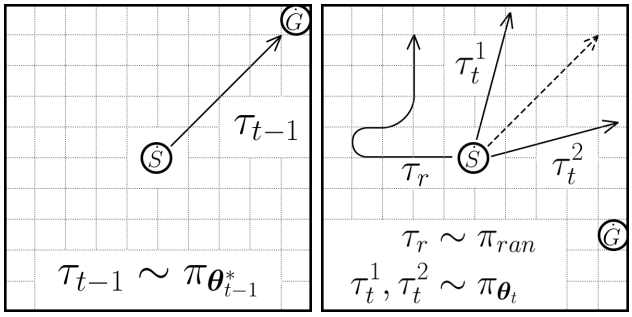


Fig. 2. Simple example of the 2-D navigation task in a dynamic environment where the goal changes. S is the start point and G is the goal point. τ is one learning episode generated by the associated policy π. (a) Original environment M_{t-1}. (b) New environment M_t.

We can see that initializing parameters from the original environment empirically benefit the learning process when starting to interact with the new environment. However, a potential drawback in this initialization may degrade the agent's performance in the long term. As discussed above, in the new environment, the initial set of policy parameters tends to be a local optimum that has been overfitted to the original environment. For highly nonlinear function approximators such as DNNs, the parameter iterate may move from one local basin to another, since the optimization can suffer from pathological curvature [47]. Hence, adjusting the previous optimum to a new one under a new data distribution could get stuck in bad local basins. Moreover, every critical point that is not a global optimum is a saddle point [42], which can significantly slow down training. In the beginning, since the current policy has not been adapted to the new environment yet, the agent still tends to execute policies that are close to the previous optimal one, and hence, the parameters are updated in the adjacent regions of the previous optimum. Therefore, we need to encourage the policies to move toward regions of parameter space that better fits in the new environment, which may be far away from the previous optimum.

Due to the environment change, the two example episodes in Fig. 2(b), τ_t^1 and τ_t^2, cannot obtain a satisfactory learning performance yet in the new environment. Nevertheless, compared to τ_t^1, the episode τ_t^2 is closer to the new optimal path and receives a higher return in the new environment. Empirically, it indicates that episodes receiving higher returns are more in line with the new environment, i.e., containing more new information. Based on this insight, we propose to use an importance weighting mechanism: during parameter updating, we assign higher importance weights to episodes that contain more new information, thus encouraging the previous optimum of parameters to be faster adjusted to a new one

Algorithm 2 Importance Weighted Policy Gradient

Input: Batch size m; learning rate α;
increment of smoothness constant Δu
Output: Optimal policy parameters $\boldsymbol{\theta}^*$

1 Initialize the smoothness constant u
2 **while** *not converged* **do**
3 Sample m episodes from the behavior policy: $\tau^i \sim \pi_{\boldsymbol{\theta}}$
4 Calculate the importance weight $w(\tau^i)$ for each episode using (12)
5 $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=1}^{m} w(\tau^i) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau^i) r(\tau^i)$
6 $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
7 $u \leftarrow u + \Delta u$
8 **end**

that fits in the new environment. It may be helpful for the algorithm to escape from those "deceptive" regions adjacent to the parameter space of the previous optimum.

Motivated by the above observation, the importance weight $w(\tau^i)$ assigned to an episode τ^i can be calculated using the received return $r(\tau^i)$ as

$$w(\tau^i) = \frac{1}{\rho}(r(\tau^i) + u), \quad i = 1, \ldots, m \tag{12}$$

where m is the batch size, u is a smoothness constant for preventing the weight of an episode with a small return being zero, and ρ is a normalization constant making the average of weights from a minibatch equal to 1. When u gets larger, all w's will be close to 1, and this weighting metric reduces to uniform weighting. In practice, we increase the smoothness constant u by a small increment Δu every iteration when the policy parameters are updated. As the learning proceeds, the effect of importance weighting is gradually weakening, thus the policy gradient process tends to be unbiased eventually compared to the conventional RL setting. Correspondingly, the gradient ascent rule in (5) is rewritten as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[w(\tau) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\tau) r(\tau) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}(\tau)} \left[\sum_{t=0}^{\infty} w(\tau) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t|s_t) \left(\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} \right) \right].\end{aligned} \tag{13}$$

Together, the algorithm of importance weighted policy gradient is presented in Algorithm 2.

D. Integrated Algorithm

With the above-mentioned policy relaxation and importance weighting mechanisms, Algorithm 3 presents the integrated incremental RL method in continuous spaces for dynamic environments. The agent is interacting with a dynamic environment $\mathcal{D} = \{M_1, M_2, \ldots\}$. In the first time period, we learned the first policy network from scratch using the canonical policy gradient method, as shown in Lines 2–7. Then, during a later tth ($t \geq 2$) time period, we initialize the policy parameters using the learned parameters that are optimal during the previous time period in Line 11. Empirically, this step circumvents the necessity for repeatedly retraining the policy network from scratch whenever the environment changes, because the previous policy network has learned some of the features of the state-action space. In Line 14, the policy relaxation mechanism is applied to generate more diversified samples to encourage a proper exploration in the k burn-in episodes. The importance sampling technique utilized to ensure the estimated policy is unbiased compared to the conventional RL setting in Line 17. In Lines 16 and 20, an importance weight is assigned to each learning episode, aiming at escaping from bad local optima and a faster adaptation to the new environment. Next, the gradient of the policy network is computed in Lines 17 and 21, and the parameters are updated in Lines 23–25 till convergence. Finally, the optimal policy $\pi_{\boldsymbol{\theta}_t^*}$ is obtained for the new environment M_t. Correspondingly, the entire process of the integrated algorithm is illustrated by a flow diagram as shown in Fig. 3.

Algorithm 3 Incremental RL in Continuous Spaces for Dynamic Environments

Input: Dynamic environment $\mathcal{D} = \{M_1, M_2, \ldots\}$;
current time period $t (t \geq 1)$;
learning rate α; batch size m;
number of burn-in episodes k;
increment of smoothness constant Δu
Output: Optimal policy parameters $\boldsymbol{\theta}_t^*$ for M_t

1 **if** *t equals to* 1 **then**
2 Randomly initialize $\boldsymbol{\theta}_t$
3 **while** *not converged* **do**
4 Sample m episodes: $\tau^i \sim \pi_{\boldsymbol{\theta}_t}, i = 1, \ldots, m$
5 $\nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) = \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\tau^i) r(\tau^i)$
6 $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t)$
7 **end**
8 **else**
9 Initialize the learning episode: $\eta \leftarrow 0$
10 Initialize the smoothness constant u
11 Initialize: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1}^*$
12 **while** *not converged* **do**
13 **if** $\eta \leq k$ **then**
14 $\pi_r(a|s) = \text{Uniform}(A(s)), \forall s$
15 Sample m episodes from π_r: $\tau^i \sim \pi_r$
16 Calculate the importance weights $w(\tau^i)$ for each episode using (12)
17 $\nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) = \sum_{i=1}^{m} w(\tau^i) \frac{\pi_{\boldsymbol{\theta}_t}(\tau^i)}{\pi_r(\tau^i)} \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\tau^i) r(\tau^i)$
18 **else**
19 Sample m episodes from $\pi_{\boldsymbol{\theta}_t}$: $\tau^i \sim \pi_{\boldsymbol{\theta}_t}$
20 Calculate the importance weights $w(\tau^i)$ for each episode using (12)
21 $\nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) = \sum_{i=1}^{m} w(\tau^i) \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\tau^i) r(\tau^i)$
22 **end**
23 $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t)$
24 $\eta \leftarrow \eta + m$
25 $u \leftarrow u + \Delta u$
26 **end**
27 **end**

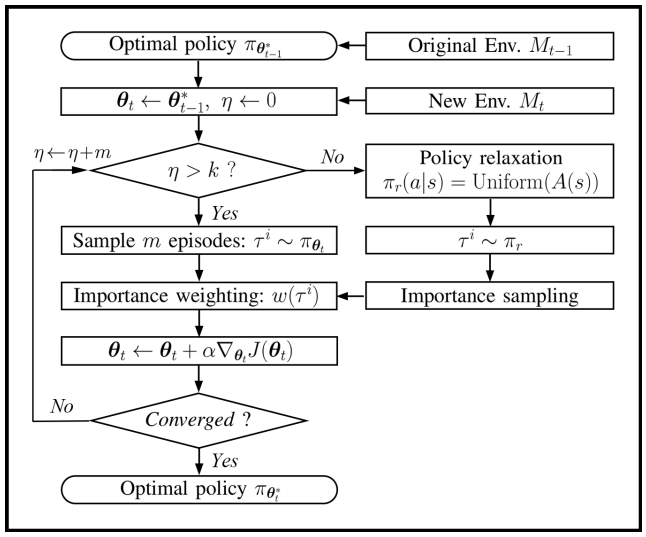


Fig. 3. Flow diagram of the integrated incremental RL algorithm in continuous spaces. Env. stands for environment.

Here, we give a convergence analysis on the integrated algorithm. First, since policy relaxation is ensured to be unbiased by using the importance sampling technique in (10), it enjoys the same convergence guarantees as canonical policy gradient methods [1]. Second, the convergence guarantee of the importance weighted policy gradient in Algorithm 2 is presented in Theorem 1. Therefore, it can be demonstrated that the integrated algorithm exhibits good theoretical convergence properties.

Theorem 1: Let π be any differentiable policy approximator with σ-bounded derivatives that $\max_{\boldsymbol{\theta},s,a,i,j} |(\partial^2 \pi(a|s))/(\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j)| < \sigma < \infty$. Let $\{\alpha_n\}_{n=0}^{\infty}$ be any step-size sequence such that $\lim_{n\to\infty} \alpha_n = 0$ and $\sum_n \alpha_n = \infty$. Let $\{w_n\}_{n=0}^{\infty}$ be any importance weight sequence that is clipped to be bounded by $[\underline{w}, \overline{w}]$, i.e., $0 < \underline{w} \leq w_n \leq \overline{w} < \infty$. Let $d^{\pi}(s) = \lim_{t\to\infty} Pr(s_t = s|s_0, \pi)$, $r(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t | s_0, \pi]$, and $Q^{\pi}(s,a) = \mathbb{E}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'} | s_t = s, a_t = a, \pi]$ denote the stationary distribution of states, the discounted return, and the action value under π, respectively. Then, for any MDP with bounded rewards, the sequence $\{r(\pi_n)\}_{n=0}^{\infty}$, defined by any $\boldsymbol{\theta}_0$, $\pi_n = \pi(\boldsymbol{\theta}_n)$, and

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \alpha_n w_n \sum_s d^{\pi_n}(s) \sum_a \frac{\partial \pi_n(a|s)}{\partial \boldsymbol{\theta}} Q^{\pi_n}(s,a) \quad (14)$$

converges such that $\lim_{n\to\infty} (\partial r(\pi_n))/(\partial \boldsymbol{\theta}) = 0$.

Proof: The bounds on $(\partial^2 \pi(a|s))/(\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j)$ and on the MDP's rewards together assure us that $(\partial^2 r(\pi))/(\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j)$ is also bounded. Let $\widetilde{\alpha}_n = \alpha_n w_n$, then the gradient ascent rule in (14) becomes

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n + \widetilde{\alpha}_n \sum_s d^{\pi_n}(s) \sum_a \frac{\partial \pi_n(a|s)}{\partial \boldsymbol{\theta}} Q^{\pi_n}(s,a) \quad (15)$$

where $\widetilde{\alpha}_n$ can be considered as a new step-size. According to the boundedness of the importance weight w_n, we have

$$\lim_{n\to\infty} \widetilde{\alpha}_n \leq \lim_{n\to\infty} \alpha_n \overline{w} = \overline{w} \cdot \lim_{n\to\infty} \alpha_n = 0$$
$$\lim_{n\to\infty} \widetilde{\alpha}_n \geq \lim_{n\to\infty} \alpha_n \underline{w} = \underline{w} \cdot \lim_{n\to\infty} \alpha_n = 0$$

which gives the result that $\lim_{n\to\infty} \widetilde{\alpha}_n = 0$. Furthermore, we have

$$\sum_n \widetilde{\alpha}_n \leq \sum_n \alpha_n \overline{w} = \overline{w} \cdot \sum_n \alpha_n = \infty$$
$$\sum_n \widetilde{\alpha}_n \geq \sum_n \alpha_n \underline{w} = \underline{w} \cdot \sum_n \alpha_n = \infty$$

so that $\sum_n \widetilde{\alpha}_n = \infty$. These two requirements on the new step-size $\widetilde{\alpha}_n$, together with the bound on $(\partial^2 r(\pi))/(\partial \boldsymbol{\theta}_i \partial \boldsymbol{\theta}_j)$, assure that the sequence $\{r(\pi_n)\}_{n=0}^{\infty}$ following the ascent rule in (15) converges to a local optimum [30], [48], i.e., $\lim_{n\to\infty} (\partial r(\pi_n))/(\partial \boldsymbol{\theta}) = 0$. □

Remark 2: Indeed, how to detect and identify changes is a crucial part of learning in dynamic environments. In this paper, we solely focus on how RL algorithms can rapidly adapt to the new environment once the change has been detected and identified. This is in a similar spirit to those studies in fault-tolerant control or dynamic multi-objective optimization [49], which exclusively concentrate on enabling controllers/algorithms to quickly accommodate dynamic changes while leaving the detection and identification to be addressed separately.

Remark 3: We also give a rough complexity analysis on the integrated algorithm. In the policy relaxation procedure, we need to calculate the importance sampling ratio $(\pi_{\boldsymbol{\theta}}(\tau))/(\pi_r(\tau))$ in (11). However, it would not consume any computation for extra variables, since $\pi_r(\tau)$ is uniformly distributed and $\pi_{\boldsymbol{\theta}}$ has been calculated in the primitive policy gradient step. Similarly, when using importance weighting in (13), no additional variables need to be addressed since the weights are calculated from the episodes' returns. Calculating the weights in (12) would incur effectively zero extra overhead, because the scalars generally take up much less memory than the large parameter vector $\boldsymbol{\theta}$ updated in each iteration. Together, it can be observed that the integrated algorithm occupies roughly the same computational complexity as the canonical policy gradient methods.

IV. Experiments

To evaluate the proposed method on RL problems in continuous spaces, we construct several sets of tasks based on the simulated continuous control environments in the rllab benchmark suite [14], particularly addressing the following questions.

- Q1: What is the degree of dynamic changes in the environment that can be handled by the proposed method?
- Q2: Does the proposed method achieve a faster adaptation to these dynamic environments?
- Q3: How do the policy relaxation and the importance weighting mechanisms affect the incremental learning performance, respectively?

A. Experimental Settings

1) Baselines: Since we aim at improving the adaptability of RL algorithms to dynamic environments, our focus is to compare with two direct baselines: Random and Pretrained. In addition, a particularly related transfer learning method, policy reuse policy gradient (PRPG), is compared in the experiments. The baselines are agiven as follows.

1) *Random:* Training a policy from randomly initialized parameters in the new environment, i.e., learning from scratch.
2) *Pretrained:* Initializing the policy parameters from the original environment and directly training the policy in the new environment.
3) *PRPG:* As a more challenging reference point, we report results for a policy-based transfer method called probabilistic policy reuse [32], [50]. We adopt a version of the algorithm that builds on policy gradient and reuses the policy from the original environment. The resulting method, PRPG, is thus directly comparable to our proposed method. The details of PRPG, including its pseudocode, are given in the Appendix.

2) Training Configurations: Empirically, RL algorithms with nonlinear function approximation are employed more for solving complex tasks, since nonlinear architectures have better approximation and generalization capabilities in regression of high-dimensional policy/value functions than the linear forms. Following the state-of-the-art benchmarks for continuous RL problems [14], [51], we adopt a similar model architecture for all of the investigated domains. The trained policy of all tested methods is approximated by a neural network with two hidden layers of size 100, with rectified linear unit (ReLU) nonlinearities. The discount factor is set as $\gamma = 0.99$.

For each report unit (a particular algorithm running on a particular task), we define two performance metrics. One is the average return over a batch of learning episodes in each policy iteration, which is defined as $(1/m)\sum_{i=1}^{m} r_i(\pi_{\boldsymbol{\theta}})$, where m is the batch size, and $r_i(\pi_{\boldsymbol{\theta}})$ is the received return for executing the associated policy. The other is the average return over all policy iterations, which is defined as $(1/mJ)\sum_{j}^{J}\sum_{i=1}^{m} r_i^j(\pi_{\boldsymbol{\theta}})$, where J is the number of training iterations. The former will be plotted in figures and the latter will be presented in tables.

We utilize a statistical analysis method to address the issue of stochastic dynamic environments. In each task, the learning agent first learned an optimal policy given a randomly chosen environment. Then, the environment randomly changes to a new one, and we record the performance of all tested methods when adapting to the new environment. We repeat the process for 20 times and report the statistical results to demonstrate the performance for learning in dynamic environments. Moreover, due to the randomness of neural networks, we repeat five runs over each policy training and report the mean regarding the performance metrics. All the algorithms are implemented with Python 3.6 running on Ubuntu 16 with 20 Intel Core i7-6950X 3.00-GHz CPU processors, 128-GB RAM, and a Titan Xp GPU of 12-GB memory. The code is available online.[2]

B. 2-D Navigation Tasks

In our first RL experiment in a dynamic environment, we study the task where a point agent must move to a goal position in 2-D. The observation is the current 2-D position, and the actions correspond to 2-D velocity commands clipped to be in the range of $[-0.1, 0.1]$. Episodes terminate when the agent is within 0.01 of the goal or at the horizon of $H = 100$. The gradient updates are computed using vanilla policy gradient (REINFORCE) [14], [52]. The hyperparameters are set as learning rate $\alpha = 0.01$, batch size $m = 20$, and burn-in episodes $k = 100$.

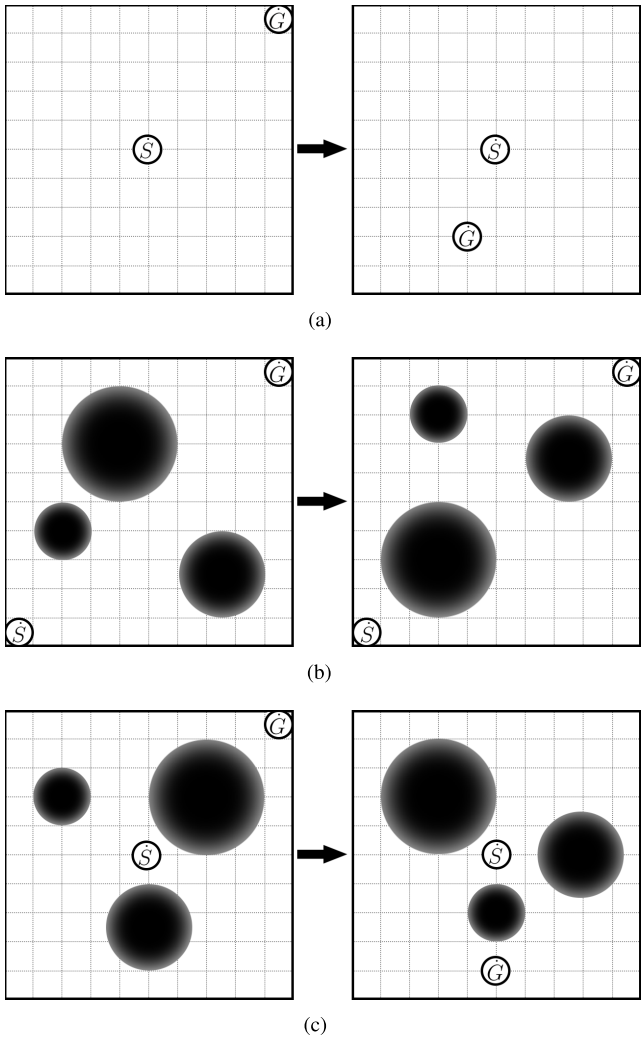


Fig. 4. Examples of three types of dynamic environments in the 2-D navigation tasks. $\dot{S}$ is the start point and $\dot{G}$ is the goal point. Puddles are shown in black. (a) Type I: the goal changes. (b) Type II: the puddles change. (c) Type III: both the goal and the puddles change.

1) Q1: To address Q1, we simulate three types of dynamic environments in Fig. 4 as follows.

1) *Type I:* As shown in Fig. 4(a), the dynamic environment is created by changing the goal position within the unit square randomly. The reward is the negative squared distance to the goal minus a control cost that is positively related to the scale of actions. Corresponding to the statement in Section III-A, the environment changes in the reward functions in this case.
2) *Type II:* This experimental domain is also a modified version of the benchmark puddle world environment presented in [31] and [53]. As shown in Fig. 4(b), the agent should drive to the goal while avoiding the three circular puddles with different sizes. When hitting on the puddles, the agent will receive an additional negative reward and

[2]https://github.com/HeyuanMingong/irl_cs

TABLE I
NUMERICAL RESULTS IN TERMS OF AVERAGE RETURN OVER ALL ITERATIONS OF ALL TESTED METHODS IMPLEMENTED IN THE 2-D NAVIGATION TASKS. HERE AND IN SIMILAR TABLES BELOW, THE MEAN ACROSS 20 RUNS IS PRESENTED, AND THE CONFIDENCE INTERVALS ARE THE CORRESPONDING STANDARD ERRORS. THE BEST PERFORMANCE IS MARKED IN BOLDFACE

Task	Type I	Type II	Type III
Random	-61.89 ± 0.26	-52.18 ± 0.54	-66.89 ± 0.75
Pretrained	-24.47 ± 0.78	-27.06 ± 1.00	-43.49 ± 3.54
PRPG	-63.78 ± 1.01	-48.62 ± 0.95	-68.05 ± 0.85
Proposed method	$\mathbf{-14.32 \pm 0.43}$	$\mathbf{-21.92 \pm 0.87}$	$\mathbf{-22.35 \pm 0.66}$

bounce to its previous position. The dynamic environment is created by moving the puddles within the unit square randomly, i.e., the environment changes in the reward and state transition functions.

3) *Type III:* As a combination of the above two types shown in Fig. 4(c), this kind of dynamic environment is created by changing both the goal and the puddles within the unit square randomly. It is considered to be more complex than the other two types.

2) Q2: To address Q2, the main experimental results of the three baselines and the proposed method regarding the three types of dynamic environments are first presented. The average return per iteration is shown in Fig. 5. Furthermore, Table I reports the numerical results in terms of average return over all iterations, which are acquired from 100 iterations for types I and III, and 500 iterations for type II, respectively. It can be observed that, in all three types of dynamic environments, the proposed method achieves a larger average return as well as a faster learning adaptation compared to the three baselines. The performance gap in terms of average return per iteration is more pronounced for smaller amounts of computation (Fig. 5), which is supposed to benefit from the distinct acceleration of adaptation to these dynamic environments. The Pretrained baseline performs the best among the three baselines due to the benefit from the initialization of policy parameters from the original environment.

Fig. 6 shows the detailed running time regarding the received average return per iteration. Compared to the baselines, the required time of the proposed method increases much slower as the received average return increases. Furthermore, Table II shows the final running time required for convergence[3] of all test methods. Roughly, for a comparable convergence performance, the total time required of the proposed method is 2–9 times smaller than that of the baselines. In summary, consistent with the statement in Section III-A, it is verified that the proposed method is capable of handling various types of dynamic environments where the reward and state transition functions may change and provides a significantly faster learning adaptation to them.

Remark 4 (Jumpstart): It can be observed that the Pretrained baseline and the proposed method achieve a significant jumpstart [17] performance compared to the other baselines in types I and II settings. It is consistent with the analysis in Section III-C that, compared to a set of randomly initialized policy parameters, the previous one has learned some of the features of the state-action space in the original environment. Recall the environments in Fig. 4, compared to type I, the optimal path in type II is likely to change less, indicating that this type of dynamic environment is less complex. Thus, the jumpstart is obviously more appealing in type II setting. In contrast, the type III dynamic environment is so complex (i.e., changing too much) that the initial jumpstart improvement is much smaller than that of the other two types.

Remark 5 (PRPG): The performance of PRPG is close to that of Random. In all three settings, PRPG receives a slightly higher return than Random in the first few iterations, because the reused policy is a little better than a randomly initialized policy in the new environment. However, since only one source task is available in the incremental learning setting, the provided knowledge base is not likely to benefit the target task a lot. Due to the change of the environment, the reused policy from the original environment quickly becomes less useful in the new environment. Comparing PRPG with Pretrained, it can be inferred that transferring the parameters of the policy network (Pretrained) generally performs much better than transferring only the behavior policies (PRPG).

Due to involving importance weighting of learning episodes, the proposed method shares some similarities with methods that reweight reward functions using function approximation. Therefore, we further compare the proposed method with several recent reward-weighted regression (RWR) methods [14]. RWR uses a function $\lambda : \mathbb{R} \to \mathbb{R}_{\geq 0}$ that transforms raw returns to nonnegative values. We report results for the following three kinds of RWR methods.

1) *RWR-1:* $\lambda(r) = \beta e^{-\beta r}$ [54], where β is an adaptive internal parameter.
2) *RWR-2:* Policy learning by weighting exploration with the returns [55], $\lambda(r) = \varepsilon r$, where ε is associated with the exploration degree when sampling actions from the Gaussian distribution in (3).
3) *RWR-3:* $\lambda(r) = r - r_{\min}$ [14], [56], where $r_{\min}$ is the minimum return among all episodes collected in the current iteration. For a fair comparison in the incremental learning setting, we also initialize the policy parameters from the original environment for these RWR methods.

Fig. 7 shows the average return per iteration, and Table III shows the numerical results in terms of average return over all iterations. It can be observed that the proposed method significantly outperforms these state-of-the-art RWR methods.

3) Q3: To address Q3, i.e., identify the respective effects of the policy relaxation and importance weighting mechanisms, a control variables approach is adopted to separate the two processes apart for observation. In each task, after initializing the policy parameters from the original environment, the following four variants of the proposed method are applied for the learning adaptation in the new environment.

1) *None:* No policy relaxation or importance weighting mechanism is used, i.e., degenerating to the Pretrained baseline.

[3]Empirically, all variants of the policy gradient algorithms in this paper are judged to have converged when the increased return over ten iterations is less than 10^{-2}. More details about the convergence of the policy gradient theorem can be found in [1].

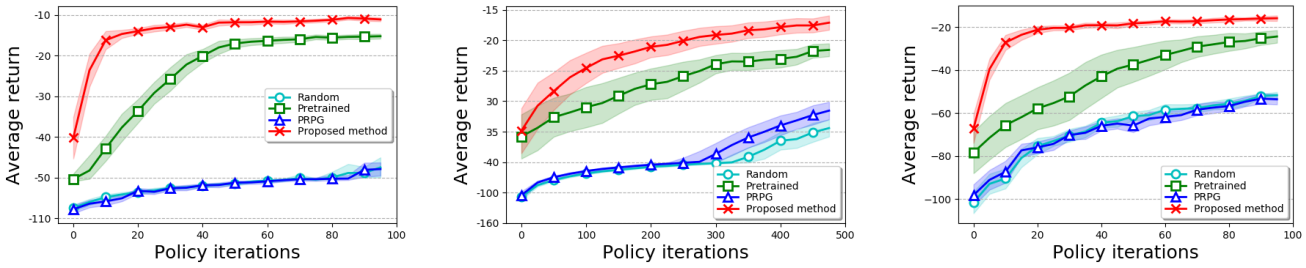


Fig. 5. Average return per iteration of all tested methods in the 2-D navigation tasks. Here and in similar figures below, the mean of average return per iteration across 20 runs is plotted as the bold line with 95% bootstrapped confidence intervals of the mean (shaded). (a) Type I. (b) Type II. (c) Type III.

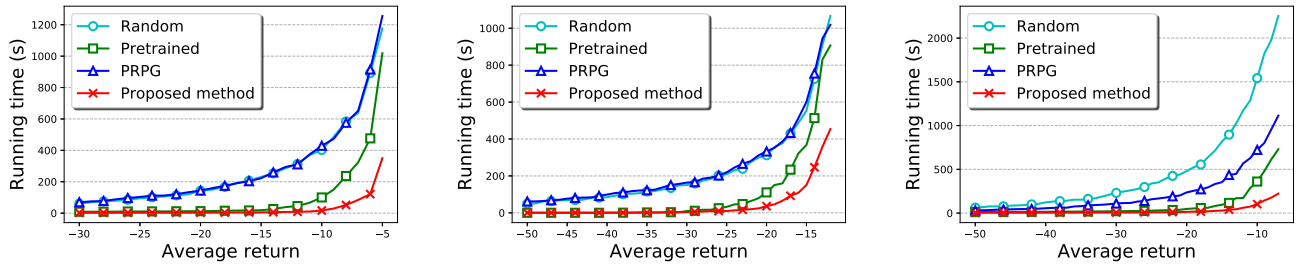


Fig. 6. Running time regarding the averaged return per iteration of all tested methods in the 2-D navigation tasks. (a) Type I. (b) Type II. (c) Type III.

TABLE II

RUNNING TIME (SECONDS) FOR CONVERGENCE OF ALL TESTED METHODS IN THE THREE TYPES OF 2-D NAVIGATION TASKS. THE BEST PERFORMANCE IS MARKED IN BOLDFACE

Task	Random	Pretrained	PRPG	Proposed method
Type I	1049	683	1136	**195**
Type II	1135	1168	1318	**571**
Type III	3023	1011	1248	**353**

TABLE III

NUMERICAL RESULTS IN TERMS OF AVERAGE RETURN OVER ALL ITERATIONS OF THE PROPOSED METHOD AND THE RWR METHODS IN THE 2-D NAVIGATION TASKS

Task	Type I	Type II	Type III
RWR-1	-24.45 ± 0.90	-26.56 ± 0.72	-37.47 ± 2.66
RWR-2	-48.28 ± 1.81	-35.15 ± 1.63	-76.42 ± 4.43
RWR-3	-30.84 ± 1.13	-28.92 ± 1.32	-53.64 ± 4.52
Proposed method	$\mathbf{-14.32 \pm 0.43}$	$\mathbf{-21.92 \pm 0.87}$	$\mathbf{-22.35 \pm 0.66}$

2) *PR:* Only apply the policy relaxation mechanism.
3) *IW:* Only apply importance weighting mechanism.
4) *PR + IW:* Both the policy relaxation and the importance weighting mechanisms are used.

The learning performance in terms of average return per iteration of the four variants is shown in Fig. 8.

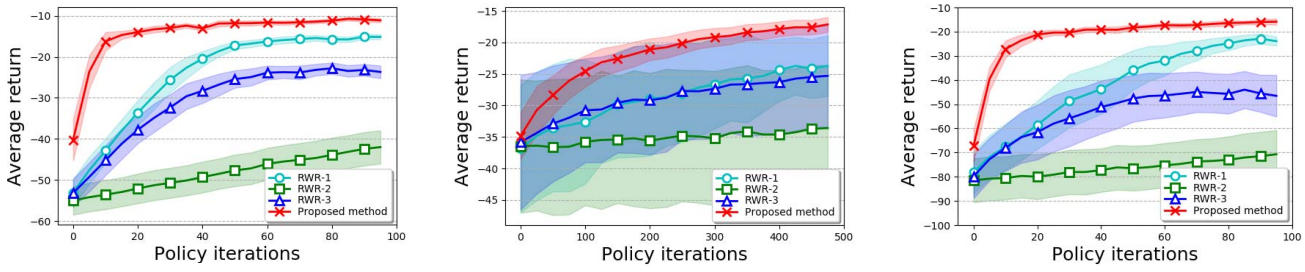
First, the variants of None and PR are compared to identify how the policy relaxation mechanism affects the adaptation in the new environment. In all three types of settings, the policies are relaxed for just a few initial iterations, and the performance in terms of average return is improved a little bit in the subsequent learning iterations. It verifies that encouraging a proper exploration in the beginning leads to a better adaptation in the long term, as stated in Section III-B. Specifically, in type II setting, the policy relaxation degenerates the average return a bit at early policy iterations. Recall the analysis in Section III-C that initializing parameters from the original environment tends to benefit the new learning process. Relaxing the behavior policy to a uniform one probably sacrifices that benefit a bit at early steps. However, in the long term, the policy relaxation will boost the performance regarding the learning speed and average return when adapting to the new environment, i.e., short-term pessimism and long-term optimism.

Next, the IW variant is compared to None for verifying the effectiveness of the importance weighting mechanism. Obviously, when weighing each learning episode according to its importance, the performance in terms of average return per iteration can be improved during the entire learning process. In the incremental learning setting, since the policy parameters are initialized from another environment, the generated episodes with higher returns are considered to be more in line with the new environment, and hence contain more new information. It verifies the assumption in Section III-C that if higher importance weights are assigned to episodes that contain more new information, the previous optimum of parameters is likely to be faster adjusted to a new one that fits in the new environment.

Finally, all the four variants are compared. It can be observed that the importance weighting mechanism better improves the learning performance than the policy relaxation one, and combining the two mechanisms together (PR + IW) leads to the best adaptation to these various dynamic environments.

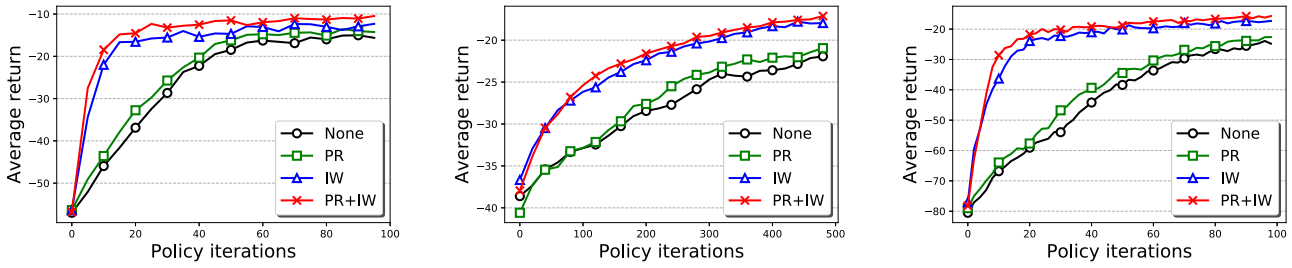
C. Locomotion Tasks

To study how well the proposed method can scale to more complex DRL problems in dynamic environments,



(a) (b) (c)

Fig. 7. Average return per iteration of the proposed method and the RWR methods in the 2-D navigation tasks. (a) Type I. (b) Type II. (c) Type III.



(a) (b) (c)

Fig. 8. Average return per iteration of the four variants of the proposed method in the 2-D navigation tasks. (a) Type I. (b) Type II. (c) Type III.

we also study the learning adaptation on several high-dimensional locomotion tasks. We employ four environments in the MuJoCo simulator [57] of OpenAI Gym to evaluate the algorithms, including Reacher, Swimmer, Hopper, and HalfCheetah domains. The state and action spaces are the original ones provided in the MuJoCo environments. All the tasks require controlling an agent to move, where the agent is all composited by links and joints. The hyperparameters are set as learning rate $\alpha = 0.01$, batch size $m = 50$, burn-in episodes $k = 200$, and time horizon $H = 100$. The detailed setting is described in the following.

1) Reacher: This domain consists of moving a two-joint torque-controlled simulated robotic arm to a specific target location. The reward is the negative squared distance to the target point minus a control cost. The gradient updates are computed using vanilla policy gradient with the learning rate $\alpha = 0.01$. Similar to the 2-D navigation domain, we vary the reward and state transition functions to create three types of stochastic dynamic environments as follows.

1) *Type I*: Varying the target location within the reachable circle randomly. The environment changes in the reward functions in this case.
2) *Type II*: Varying the physical variables of "link0" and "joint0" at random, i.e., the state transition functions change.
3) *Type III*: Varying both the target location and the physical variables randomly. The environment changes in both the reward and state transition functions. The meaning of the physical variables can be found in the MuJoCo simulator.

2) Swimmer/Hopper/HalfCheetah: It requires a 2-D Swimmer/one-legged Hopper/planar Cheetah robot to swim/hop/run forward at a particular velocity. The reward is the negative absolute value between the current velocity of the agent and a goal, minus a control cost. The dynamic environment is created by changing the goal velocity between 0.0 and 2.0 at random. To handle these challenging tasks, the trust region policy optimization (TRPO) [58] is employed as the base policy search algorithm.

First, all tested methods are applied to the three types of dynamic environments in the Reacher domain. Fig. 9 shows the learning performance in terms of the average return per policy iteration, and Table IV shows the numerical results in terms of the average return over all 500 training iterations. Similar to the 2-D navigation domain, the Pretrained baseline and the proposed method obtain a much better performance than the other two baselines. It can be observed that the proposed method is capable of handling various dynamic environments in the challenging Reacher tasks and achieves a faster learning adaptation to them compared to all baselines.

Furthermore, the proposed method and the baselines are implemented on the complex Swimmer/Hopper/HalfCheetah tasks, and the learning performances in terms of the received return per iteration and the average return over all iterations are presented in Fig. 10 and Table V, respectively. The Pretrained baseline and the proposed method obtain an obvious jumpstart performance in all three tasks. It indicates that the locomotion skills learned in the original environment can help the new learning process a lot in the beginning. However, in Swimmer and Hopper domains, the Pretrained baseline receives a nonincreasing return in the latter policy iterations. We conjecture that the learning algorithm probably gets stuck in a bad local optimum in the new environment since it was overfitted to the original environment. Consistent with the analysis in Section III-C, initializing parameters from the original environment tends to benefit the learning agent when starting to interact with the new environment, while it may degrade the learning performance in the long term. On the contrast, the proposed method can receive significantly

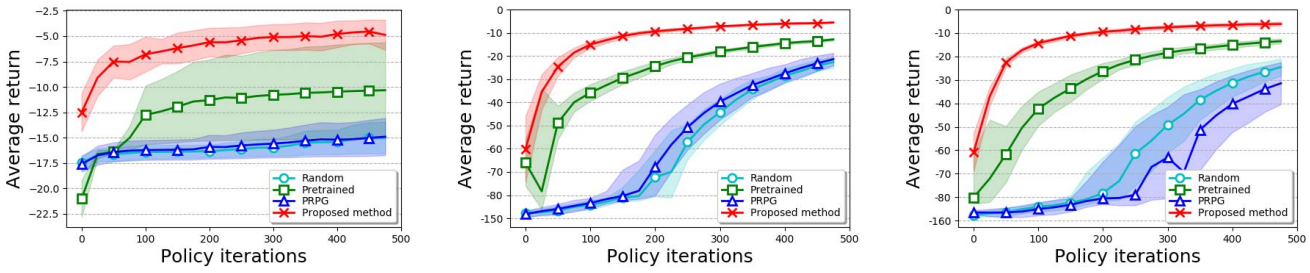


Fig. 9. Average return per iteration of all tested methods in the Reacher domain tasks. (a) Type I. (b) Type II. (c) Type III.

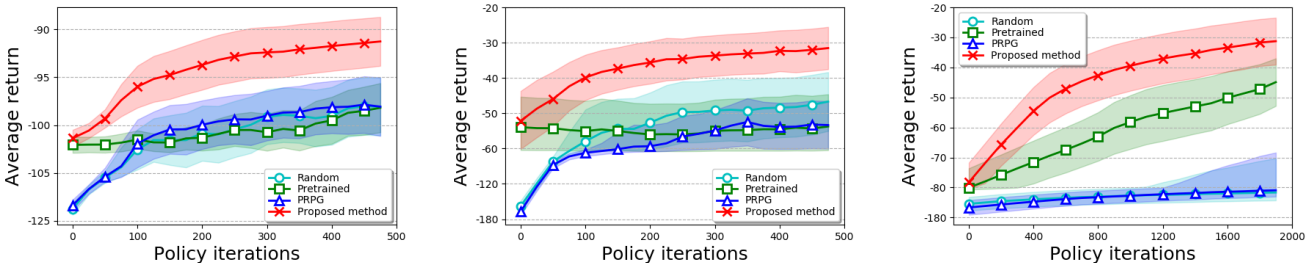


Fig. 10. Average return per iteration of all tested methods in the complex Swimmer/Hopper/HalfCheetah domain tasks. (a) Swimmer. (b) Hopper. (c) HalfCheetah.

TABLE IV

NUMERICAL RESULTS IN TERMS OF AVERAGE RETURN OVER ALL ITERATIONS OF ALL TESTED METHODS IMPLEMENTED IN THE REACHER DOMAIN TASKS

Task	Type I	Type II	Type III
Random	-16.06 ± 0.28	-70.11 ± 1.99	-74.76 ± 3.75
Pretrained	-12.31 ± 1.06	-27.72 ± 0.99	-31.25 ± 1.46
PRPG	-15.86 ± 0.39	-69.30 ± 2.70	-82.30 ± 5.26
Proposed method	$\mathbf{-6.14 \pm 0.44}$	$\mathbf{-14.62 \pm 0.59}$	$\mathbf{-13.76 \pm 0.30}$

TABLE V

NUMERICAL RESULTS IN TERMS OF AVERAGE RETURN OVER ALL ITERATIONS OF ALL TESTED METHODS IMPLEMENTED IN THE SWIMMER/HOPPER/HALFCHEETAH DOMAIN TASKS

Task	Swimmer	Hopper	HalfCheetah
Random	-101.04 ± 2.06	-83.06 ± 1.10	-120.15 ± 12.03
Pretrained	-100.71 ± 1.13	-64.541 ± 4.97	-47.23 ± 5.29
PRPG	-101.51 ± 1.76	-76.04 ± 2.52	-116.41 ± 10.66
Proposed method	$\mathbf{-94.29 \pm 1.60}$	$\mathbf{-44.04 \pm 3.56}$	$\mathbf{-39.46 \pm 4.88}$

higher returns during the entire learning process than all baselines. By mechanisms of encouraging a proper exploration and emphasizing the new information, the proposed method rapidly guides the policy toward regions of parameter space that better fits in the new environment. In summary, the results demonstrate that the proposed method is still capable of acquiring a faster adaptation to the dynamic environments of high-dimensional locomotion tasks.

V. CONCLUSION

This paper addresses the incremental RL problem in continuous spaces for dynamic environments, which may change in the reward and state transition functions over time. The goal is to adjust the previously learned policy in the original environment to a new one whenever the environment changes. To improve adaptability, we introduce a two-step solution incorporated with the incremental learning procedure: policy relaxation and importance weighting. First, the policy relaxation mechanism aims at a proper exploration in the new environment by relaxing the behavior policy to a uniform one for a few learning episodes. It alleviates the conflict between the new information and the existing knowledge for a better adaptation in the long term. Second, an importance weighting mechanism is applied based on the observation that episodes receiving higher returns are more in line with the new environment, and hence contain more new information. During parameter updating, we assign higher weights to episodes that contain more new information, thus encouraging the previous optimal policy to be faster adapted to a new one that fits in the new environment. Experiments were conducted on traditional navigation tasks and complex locomotion tasks with varying configurations. The results verified that the proposed method was capable of handling various types of dynamic environments and providing a significantly faster learning adaptation to them.

Our future work will address more complicated cases, such as detecting changes of environments, learning in more challenging dynamic environments where the state-action space may change, or learning in intensively changing environments (e.g., changing between consecutive episodes).

APPENDIX
BASELINE PRPG

The policy reuse learner [32] improves its exploration by probabilistically exploiting from these past policies. It selects to reuse the past knowledge depending on its contribution to

Algorithm 4 PRPG

Input: Previous optimal policy parameters $\boldsymbol{\theta}^*_{t-1}$; learning rate α; discount factor γ; batch size m; decay factor υ

Output: Optimal policy parameters $\boldsymbol{\theta}^*_t$ for M_t

Randomly initialize $\boldsymbol{\theta}_t$
$\text{score}_{t-1} \leftarrow 0$, the score associated with $\pi_{\boldsymbol{\theta}^*_{t-1}}$;
$\text{score}_t \leftarrow 0$, the score associated with $\pi_{\boldsymbol{\theta}_t}$;
$\text{used}_{t-1} \leftarrow 0$, the number of times $\pi_{\boldsymbol{\theta}^*_{t-1}}$ is used;
$\text{used}_t \leftarrow 0$, the number of times $\pi_{\boldsymbol{\theta}_t}$ is used
Initialize the control probability to reuse old policy: ψ
while *not converged* **do**
  for $i \leftarrow t-1, t$ **do**
    $p_i \leftarrow e^{\upsilon \times \text{score}_i} / \sum_j e^{\upsilon \times \text{score}_j}$
  end
  Select the behavior policy: $c \sim \text{Bernoulli}(p_{t-1}, p_t)$
  if $c \neq t$ **then**
    use_pre_policy $\sim$ Bernoulli(ψ)
  else
    use_pre_policy $\leftarrow$ false
  end
  if use_pre_policy **then**
    Sample m episodes: $\tau^i \sim \pi_{\boldsymbol{\theta}^*_{t-1}}, i = 1, \ldots, m$
    $\nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) = \sum_{i=1}^m \frac{\pi_{\boldsymbol{\theta}_t}(\tau^i)}{\pi_{\boldsymbol{\theta}^*_{t-1}}(\tau^i)} \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\tau^i) r(\tau^i)$
    $\text{score}_{t-1} \leftarrow \frac{\text{score}_{t-1} \times \text{used}_{t-1} + \sum_{i=1}^m r(\tau^i)}{\text{used}_{t-1}+1}$
    $\text{used}_{t-1} \leftarrow \text{used}_{t-1} + 1$
  else
    Sample m episodes: $\tau^i \sim \pi_{\boldsymbol{\theta}_t}, i = 1, \ldots, m$
    $\nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t) = \sum_{i=1}^m \nabla_{\boldsymbol{\theta}_t} \log \pi_{\boldsymbol{\theta}_t}(\tau^i) r(\tau^i)$
    $\text{score}_t \leftarrow \frac{\text{score}_t \times \text{used}_t + \sum_{i=1}^m r(\tau^i)}{\text{used}_t+1}$
    $\text{used}_t \leftarrow \text{used}_t + 1$
  end
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J(\boldsymbol{\theta}_t)$
  $\psi \leftarrow \psi \upsilon$
end

the exploration process, called reuse gain, which is discovered concurrently during the learning process. Barreto *et al.* [50] adopted a version of this algorithm that built on Q-learning, resulting in a method, called policy reuse in Q-learning (PRQL), for a direct comparison. In this paper, we adopt a version of the algorithm that incorporates with policy gradient and reuses the policy from the original environment. The resulting method, called PRPG, is used as a baseline that is directly comparable to the proposed method. The pseudocode is given in Algorithm 4.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. 2nd ed., MIT Press, Cambridge, U.K.:, 2018.

[2] R. Bellman, "Dynamic Programming," *Science*, vol. 153, pp. 34–37, Jul. 1966.

[3] G. Tesauro and G. R. Galperin, " On-line policy improvement using Monte-Carlo search," in *Proc. Adv. Neural Inf. Process. Syst.*, May 1997, pp. 1068–1074.

[4] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[5] D. Dong, C. Chen, H. Li, and T.-J. Tarn, "Quantum reinforcement learning," *IEEE Trans. Syst. Man, Cybern. B, Cybern.*, vol. 38, no. 5, pp. 1207–1220, Oct. 2008.

[6] D. Dong, C. Chen, J. Chu, and T.-J. Tarn, "Robust quantum-inspired reinforcement learning for robot navigation," *IEEE/ASME Trans. Mechatronics*, vol. 17, no. 1, pp. 86–97, Feb. 2012.

[7] C. Chen, D. Dong, H.-X. Li, J. Chu, and T.-J. Tarn, "Fidelity-based probabilistic Q-learning for control of quantum systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 5, pp. 920–933, May 2014.

[8] Z. Wang, H.-X. Li, and C. Chen, "Reinforcement learning based optimal sensor placement for spatiotemporal modeling," *IEEE Trans. Cybern.*, to be published.

[9] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107–1149, Dec. 2003.

[10] M. Riedmiller, "Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.*, Oct. 2005, pp. 317–328.

[11] A. Antos, C. Szepesvári, and R. Munos, "Fitted Q-iteration in continuous action-space MDPs," in *Proc. Adv. Neural Inf. Process. Syst.*, Oct. 2008, pp. 9–16.

[12] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Jul. 2015.

[13] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[14] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2016, pp. 1329–1338.

[15] M. A. K. Jaradat, M. Al-Rousan, and L. Quadan, "Reinforcement based mobile robot navigation in dynamic environment," *Robot. Comput. Integr. Manuf.*, vol. 27, no. 1, pp. 135–149, Feb. 2011.

[16] L. Zhou, P. Yang, C. Chen, and Y. Gao, "Multiagent reinforcement learning with sparse interactions by negotiation and knowledge transfer," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1238–1250, May 2017.

[17] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *J. Mach. Learn. Res.*, vol. 10, pp. 1633–1685, Jul. 2009.

[18] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010.

[19] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *Computer*, vol. 21, no. 3, pp. 77–88, Mar. 1988.

[20] R. Polikar, L. Upda, S. S. Upda, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.

[21] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental learning for robust visual tracking," *Int. J. Comput. Vis.*, vol. 77, nos. 1–3, pp. 125–141, 2008.

[22] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 12, no. 5, pp. 542–561, Oct. 2008.

[23] D. Kulic, D. Lee, J. Ishikawa, and Y. Nakamura, "Incremental learning of full body motion primitives and their sequencing through human motion observation," *Int. J. Robot. Res.*, vol. 31, no. 3, pp. 330–345, 2012.

[24] H. He, S. Chen, K. Li, and X. Xu, "Incremental learning from stream data," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1901–1914, Dec. 2011.

[25] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, "Incremental reinforcement learning with prioritized sweeping for dynamic environments," *IEEE/ASME Trans. Mechatronics*, vol. 24, no. 2, pp. 621–632, Apr. 2019.

[26] H. V. Hasselt, "Reinforcement learning in continuous state and action spaces," *Reinforcement Learn.*, vol. 12, pp. 207–251, Feb. 2012.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?," in *Proc. Adv. Neural Inf. Process. Syst.*, Nov. 2014, pp. 3320–3328.

[28] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Oct. 2014, pp. 1532–1543.

[29] Y. Yu, S.-Y. Chen, Q. Da, and Z.-H. Zhou, "Reusable reinforcement learning via shallow trails," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2204–2215, Jun. 2018.

[30] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, Oct. 2000, pp. 1057–1063.
[31] A. Tirinzoni, A. Sessa, M. Pirotta, and M. Restelli, "Importance weighted transfer of samples in reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn.*, Jul. 2018, pp. 4936–4945.
[32] F. Fernández, J. García, and M. Veloso, "Probabilistic Policy Reuse for inter-task transfer learning," *Robot. Auton. Syst.*, vol. 58, no. 7, pp. 866–871, Jul. 2010.
[33] G. Konidaris, I. Scheidwasser, and A. G. Barto, "Transfer in reinforcement learning via shared features," *J. Mach. Learn. Res.*, vol. 13, pp. 1333–1371, May 2012.
[34] J. Pan, X. Wang, Y. Cheng, and Q. Yu, "Multisource transfer double DQN based on actor learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2227–2238, Jun. 2018.
[35] A. Barreto *et al.*, "Transfer in deep reinforcement learning using successor features and generalised policy improvement," in *Proc. 35th Int. Conf. Mach. Learn.*, Jul. 2018, pp. 510–519.
[36] E. Parisotto, J. L. Ba, and R. Salakhutdinov, "Actor-mimic: Deep multitask and transfer reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, Jul. 2015, pp. 1–6.
[37] H. Li, X. Liao, and L. Carin, "Multi-task reinforcement learning in partially observable stochastic environments," *J. Mach. Learn. Res.*, vol. 10, pp. 1131–1186, 2009.
[38] Z. Yang, K. E. Merrick, H. A. Abbass, and L. Jin, "Multi-task deep reinforcement learning for continuous action control," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 3301–3307.
[39] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, Aug. 2017, pp. 1126–1135.
[40] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.
[41] Z. Wang and H.-X. Li, "Incremental spatiotemporal learning for online modeling of distributed parameter systems," *IEEE Trans. Syst., Man, Cybern., Syst.*, to be published.
[42] K. Kawaguchi, "Deep learning without poor local minima," in *Proc. Adv. Neural Inf. Process. Syst.*, Jul. 2016, pp. 586–594.
[43] S. Gu, T. Lillicrap, R. E. Turner, Z. Ghahramani, B. Schölkopf, and S. Levine, "Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Jun. 2017, pp. 3846–3855.
[44] D. Precup, R. S. Sutton, and S. Singh, "Eligibility traces for off-policy policy evaluation," in *Proc. Int. Conf. Mach. Learn.*, Jan. 2000, p. 80.
[45] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA, USA: MIT Press, 2009.
[46] S. Levine and V. Koltun, "Guided policy search," in *Proc. Int. Conf. Mach. Learn.*, Feb. 2013, pp. 1–9.
[47] J. Martens, "Deep learning via Hessian-free optimization," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2010, pp. 735–742.
[48] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[49] M. Jiang, Z. Huang, L. Qiu, W. Huang, and G. G. Yen, "Transfer learning-based dynamic multiobjective optimization algorithms," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 501–514, Aug. 2018.
[50] A. Barreto *et al.*, "Successor features for transfer in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, Jun. 2017, pp. 4055–4065.
[51] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," Sep. 2017, *arXiv:1703.03864*. [Online]. Available: https://arxiv.org/abs/1703.03864
[52] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
[53] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. Adv. Neural Inf. Process. Syst.*, Nov. 1995, pp. 1038–1044.
[54] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2007, pp. 745–750.
[55] J. Kober and J. R. Peters, "Policy search for motor primitives in robotics," in *Proc. Adv. Neural Inf. Process. Syst.*, Jun. 2009, pp. 849–856.
[56] M. P. Deisenroth, G. Neumann, and J. Peters, "A survey on policy search for robotics," *Found. Trends Robot.*, vol. 2, nos. 1–2, pp. 1–142, Aug. 2013.
[57] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Oct. 2012, pp. 5026–5033.
[58] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. Int. Conf. Mach. Learn.*, Jun. 2015, pp. 1889–1897.

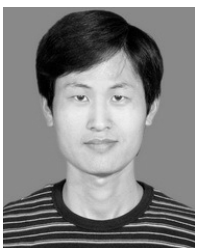
Zhi Wang (S'19) received the B.E. degree in automation from the Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University, Nanjing, China, in 2015, and the Ph.D. degree in machine learning from the Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong, in 2019.

He is currently an Assistant Professor with the Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University. His current research interests include reinforcement learning, machine learning, and robotics.

Han-Xiong Li (S'94–M'97–SM'00–F'11) received the B.E. degree in aerospace engineering from the National University of Defense Technology, Changsha, China, in 1982, the M.E. degree in electrical engineering from the Delft University of Technology, Delft, The Netherlands, in 1991, and the Ph.D. degree in electrical engineering from the University of Auckland, Auckland, New Zealand, in 1997.

He is currently a Professor with the Department of System Engineering and Engineering Management, City University of Hong Kong, Hong Kong. He has a broad experience in both academia and industry. He has authored two books. He has authored or coauthored more than 200 SCI journal papers with an h-index of 42 from the Web of Science. He holds 20 patents. His current research interests include process modeling and control, system intelligence, distributed parameter systems, and battery management system.

Dr. Li was a recipient of the Distinguished Young Scholar (overseas) by the China National Science Foundation in 2004, the Chang Jiang Professorship by the Ministry of Education, China, in 2006, and the National Professorship in China Thousand Talents Program in 2010. He serves as a distinguished expert for Hunan Government and China Federation of Returned Overseas Chinese. He was an Associate Editor of the IEEE TRANSACTIONS ON CYBERNETICS from 2002 to 2016 and the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS from 2009 to 2015. He serves as an Associate Editor for the IEEE TRANSACTIONS ON SMC: SYSTEM.



Chunlin Chen (S'05–M'06) received the B.E. degree in automatic control and the Ph.D. degree in control science and engineering from the University of Science and Technology of China, Hefei, China, in 2001 and 2006, respectively.

From 2012 to 2013, he was with the Department of Chemistry, Princeton University Princeton, NJ, USA. He held visiting positions at the University of New South Wales, Sydney, NSW, Australia, and City University of Hong Kong, Hong Kong. He is currently a Professor and the Chair with the Department of Control and Systems Engineering, School of Management and Engineering, Nanjing University, Nanjing, China. His current research interests include machine learning, intelligent control, and quantum control.

Dr. Chen is the Co-Chair of the Technical Committee on Quantum Cybernetics for the IEEE Systems, Man and Cybernetics Society.