Instance Weighted Incremental Evolution Strategies for Reinforcement Learning in Dynamic Environments

Zhi Wang, *Member, IEEE*, Chunlin Chen, *Senior Member, IEEE*, and Daoyi Dong, *Senior Member, IEEE*

***Abstract*— Evolution strategies (ESs), as a family of black-box optimization algorithms, recently emerge as a scalable alternative to reinforcement learning (RL) approaches such as Q-learning or policy gradient and are much faster when many central processing units (CPUs) are available due to better parallelization. In this article, we propose a systematic incremental learning method for ES in dynamic environments. The goal is to adjust previously learned policy to a new one incrementally whenever the environment changes. We incorporate an instance weighting mechanism with ES to facilitate its learning adaptation while retaining scalability of ES. During parameter updating, higher weights are assigned to instances that contain more new knowledge, thus encouraging the search distribution to move toward new promising areas of parameter space. We propose two easy-to-implement metrics to calculate the weights: instance novelty and instance quality. Instance novelty measures an instance's difference from the previous optimum in the original environment, while instance quality corresponds to how well an instance performs in the new environment. The resulting algorithm, instance weighted incremental evolution strategies (IW-IESs), is verified to achieve significantly improved performance on challenging RL tasks ranging from robot navigation to locomotion. This article thus introduces a family of scalable ES algorithms for RL domains that enables rapid learning adaptation to dynamic environments.**

***Index Terms*— Dynamic environments, evolution strategies (ESs), incremental learning, instance weighting, reinforcement learning (RL).**

I. Introduction

IN REINFORCEMENT learning (RL) [1], an agent learns to perform a sequence of actions in an environment that maximizes cumulative reward based on the Markov decision process (MDP) formalism [2]–[6]. A primary driving force behind the explosion of RL is its integration with powerful nonlinear function approximators such as deep neural networks (DNNs), aiming to develop agents that can accomplish challenging tasks in complex and uncertain environments. This partnership with deep learning, i.e., deep reinforcement learning (DRL), has enabled RL to successfully extend to tasks with high-dimensional state and action spaces, ranging from arcade games [7] and board games [8] to robotic control tasks [9].

An alternative approach to solving RL problems is using black-box optimization, known as direct policy search [10] or neuroevolution [11] when applied to neural networks. Evolution strategies (ESs) [12] are a particular family of these optimization algorithms that are heuristic search procedures inspired by natural evolution. Recent research has reported that ES can be competitive to popular backpropagation-based algorithms such as policy gradient and Q-learning on challenging RL problems, with much faster training speed when many central processing units (CPUs) are available due to better parallelization [13]. ES can reliably train neural network policies, in a fashion well suited to scale up to modern distributed computer systems without requirements for temporal discounting, backpropagating gradients, and value function approximation [14]–[16]. The promising properties of applying ES for solving RL problems include the following.

1) Since ES only needs to communicate scalar returns of complete episodes, it is highly parallelizable and enables near-linear speedups in runtime as a function of CPUs.
2) ES uses a fitness metric that consolidates returns across an entire episode, making it invariant to sparse or deceptive rewards with arbitrarily long time horizons.
3) The population-based evolutionary search provides diverse exploration, particularly when combined with explicit diversity maintenance techniques. Moreover, the redundancy inherent in a population also facilitates robustness and stable convergence properties, especially when incorporated with elitism.

Traditional research on ES algorithms for RL tasks mainly focuses on stationary optimization problems, which are precisely given in advance and remain fixed during the entire evolutionary process. Instead, the environments of real-world RL applications are often dynamic, where the state space, available actions, state transition functions, or reward functions may change over time instead of being static, such as for multiagent cases [17], robot navigation problems [18], or online learning

Manuscript received March 19, 2020; revised June 17, 2021, October 4, 2021, and January 1, 2022; accepted March 12, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62006111 and Grant 62073160; in part by the Australian Research Council's Discovery Projects Funding Scheme under Project DP190101566; in part by the Natural Science Foundation of Jiangsu Province of China under Grant BK20200330; and in part by the Alexander von Humboldt Foundation, Germany. *(Corresponding authors: Chunlin Chen; Daoyi Dong.)*

Zhi Wang is with the Department of Control and Systems Engineering, Nanjing University, Nanjing 210093, China, and also with the School of Engineering and Information Technology, University of New South Wales, Canberra, ACT 2600, Australia (e-mail: zhiwang@nju.edu.cn).

Chunlin Chen is with the Department of Control and Systems Engineering, Nanjing University, Nanjing 210093, China (e-mail: clchen@nju.edu.cn).

Daoyi Dong is with the School of Engineering and Information Technology, University of New South Wales, Canberra, ACT 2600, Australia (e-mail: daoyidong@gmail.com).

Color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2022.3160173.

Digital Object Identifier 10.1109/TNNLS.2022.3160173

settings [19]. This challenge leads to the dynamic optimization problems [20], [21] for the corresponding ES algorithms where the fitness function, design variables, or environmental conditions change over time.

In this article, we tackle the dynamic environment as a sequence of stationary tasks on a certain timescale where each task corresponds to a stationary environment during the associated time period. Learning in such dynamic environments is characterized not only by the capability of acquiring complex skills but also the ability to adapt rapidly under a nonstationary task distribution. Humans and animals can learn complex models that precisely and reliably reason about real-world phenomena, and they can rapidly adjust such models in the face of unexpected changes. Although (deep) neural network models can represent very complex functions, they lack the capability of rapidly adapting to dynamic environments. To circumvent the necessity for repeatedly reevolving, recent research exploits transfer learning techniques [22] as a tool to reuse information available from a set of source tasks to help the evolutionary performance in a related but different target task [20]. Generally, it requires repeatedly accessing and processing a potentially large distribution of source tasks to provide a good knowledge base for target environments that are supposed to be consistent with the source distribution.

An increasing number of real-world scenarios require RL algorithms to be capable of adapting their behaviors in an incremental manner to environments that may drift or change from their nominal situations, continuously utilizing previous knowledge to benefit the future decision-making process. Hence, incremental learning [23]–[25] emerges by incrementally adjusting the previously learned policy to a new one whenever the environment changes,[1] which offers an appealing alternative that is amenable for rapid learning adaptation to dynamic environments. Such an incremental adaptation is crucial for intelligent systems operating in the real world, where changing factors and unexpected perturbations are the norm. Incremental learning has been widely investigated to cope with learning tasks with an ever-changing environment [28], in areas such as supervised learning [29], RL [23], [24], machine vision [30], human–robot interaction [31], and system modeling [32]. However, an equivalent notion of incremental learning in ES for RL domains has largely eluded researchers, with few related work available in the literature. Here, we aim to develop a new incremental learning framework for the derivative-free ES algorithms, which is orthogonal and complementary to the previous one in [24] that is investigated for the derivative-based RL approaches.

In this article, we formulate an incremental learning procedure that uses natural evolution strategies (NESs) to update the parameters of a policy network for RL in dynamic environments. To increase the capability of rapid learning adaptation, we incorporate an instance weighting mechanism with ES to improve the learning adaptation while not sacrificing the speed/scalability benefits of ES. During parameter updating, we assign higher weights to instances that contain more knowledge on the new environment, thus encouraging the search distribution to move toward new promising areas in the parameter space. We propose two easy-to-implement metrics for calculating the weights: instance novelty and instance quality. First, instance novelty intends to indicate the instance's difference from the previous optimum in the original environment, with the help of a domain-dependent behavior characterization that describes the behavior of the associated policy. Second, instance quality corresponds to how well the instance performs in the new environment, where its performance is evaluated by the received return of the associated policy. Together, instances with high weights are supposed either to differ more from the original environment (high novelty) or to be more in line with the new environment (high quality). The resulting algorithm, instance weighted incremental evolution strategies (IW-IESs), "reinforces" the evolutionary process of searching for well-behaving policies that fit in the new environment, thus facilitating more rapid learning adaptation to dynamic environments.

We test whether IW-IES improves the performance of ES on challenging RL tasks ranging from robot navigation to locomotion in dynamic environments. Experimental results confirm that IW-IES is capable of handling various dynamic environments and achieves significantly rapid learning adaptation to these tasks. In summary, the main contributions are listed as follows.

1) We introduce IW-IESs, a family of scalable ES algorithms that addresses challenging RL problems in dynamic environments from an incremental learning perspective.
2) We incorporate an instance weighting mechanism with ES to facilitate learning adaptation to dynamic environments while retaining scalability benefits and enabling a near-linear speedup in runtime as more CPUs are used.
3) We propose two easy-to-implement metrics for calculating the weights: instance novelty and instance quality, which effectively enhance the evolutionary performance almost without extra computational complexity.
4) We perform extensive experiments to verify that IW-IES can consistently improve learning adaptation to dynamic environments over various state-of-the-art baselines.

The rest of this article sequentially presents the background on ES algorithms for RL domains in Section II, the proposed algorithm with designed weighting metrics in Section III, the experiments in Section IV, and the conclusions in Section V.

II. Background

A. ESs for RL

RL is commonly studied based on the MDP formalism. 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

[1] In this setting, the policy parameters of the new environment are initialized from the previously learned optima of the original environment. The reason is that the previous optimal policy empirically performs better than a randomly initialized one since it has learned some of the features (e.g., nodes in the neural network) of the state–action space. This procedure is akin to the pretraining in the deep learning community, where layers in a neural network extract hierarchical levels of feature representation. Model parameters pretrained on common datasets, such as ImageNet [26], can be used as a helpful initialization for general downstream tasks [27].

transition probability, $R : S \times A \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1]$ 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 $J(\pi)$

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

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

Inspired by natural evolution, ES is designed to cope with high-dimensional continuous-valued domains and has remained an active field of research for more than four decades [12]. ES algorithms address the following search problem: maximize a nonlinear fitness function that is a mapping from search space $\mathcal{S} \subseteq \mathbb{R}^d$ to $\mathbb{R}$. At each iteration (generation), a population of parameter vectors (gnomes) is perturbed (mutated) and optionally recombined (merged) via crossover. The mutation is usually carried out by adding a realization of a normally distributed random vector. Each resultant offspring is evaluated by a fitness function, and the highest scoring parameter vectors are then recombined to form the population for the next generation. Recent research highlights the scalability of ES algorithms on many high-dimensional RL tasks while offering unique benefits over traditional gradient-based RL methods [14]. Most notably, ES is highly parallelizable and well suited to modern distributed computer systems with a near-linear speedup in wall-clock runtime. Salimans *et al.* [13] reported that, with hundreds of parallel CPUs, ES is able to achieve roughly the same performance on Atari games with the same DNN architecture in 1 h as A3C [33] did in 24 h.

Algorithms in the ES class differ in their representations of population and methods of recombination. The version of ES used in this article belongs to the class of NESs [34], which constitutes a well-principled approach with a clean derivation from first principles. The core idea is to iteratively update parameters of the search distribution using the sampled gradient of expected fitness. The search distribution can be taken to be a multinormal distribution but could in principle be any distribution of which the log density is differentiable. Let $\boldsymbol{\theta}$ denote parameters of the search distribution's density $p(z|\boldsymbol{\theta})$ and $f(z)$ denote the fitness function (e.g., received return) for instance z. At each generation, a population of search instances is produced by the parameterized search distribution, and the fitness function is evaluated at each instance. The expected fitness under the search distribution is written as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}[f(z)] = \int f(z)p(z|\boldsymbol{\theta})\,\mathrm{d}z. \tag{2}$$

In a fashion similar to REINFORCE [35], NES takes gradient steps on $\boldsymbol{\theta}$ with the following estimator:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \int f(z)p(z|\boldsymbol{\theta})\,\mathrm{d}z \\ &= \mathbb{E}_{\boldsymbol{\theta}}[f(z)\nabla_{\boldsymbol{\theta}} \log p(z|\boldsymbol{\theta})].\end{aligned} \tag{3}$$

We can obtain the Monte Carlo estimate of the search gradient for instances in a population $(z_1, \ldots, z_m)$ as

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{m}\sum_{i=1}^{m} f(z_i)\nabla_{\boldsymbol{\theta}} \log p(z_i|\boldsymbol{\theta}) \tag{4}$$

where m is the population size. For each generation, NES estimates a search gradient on the parameters toward higher expected fitness in promising regions. Instead of using the plain stochastic gradient for updates, NES follows the natural gradient, which helps mitigate the slow convergence of plain gradient ascent in optimization landscapes with ridges and plateaus. The direction of the natural gradient is associated with the Fisher information matrix of the given parametric family of the search distribution

$$\begin{aligned}\boldsymbol{F} &= \int p(z|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log p(z|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log p(z|\boldsymbol{\theta})^T\,\mathrm{d}z \\ &= \mathbb{E}[\nabla_{\boldsymbol{\theta}} \log p(z|\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}} \log p(z|\boldsymbol{\theta})^T].\end{aligned} \tag{5}$$

If $\boldsymbol{F}$ is invertible, the natural gradient amounts to

$$\widetilde{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \boldsymbol{F}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}). \tag{6}$$

The local structure of the fitness function is adaptively captured by the search distribution's parameters, e.g., the mean and covariance matrix in a Gaussian distribution. The evolutionary process reiterates until a stopping criterion is met.

In RL domains, NES directly searches in the parameter space of a neural network to find an effective policy. For scalability to high-dimensional problems, the population $\{z_i\}_{i=1}^m$ is typically instantiated as a multivariate Gaussian with diagonal covariance matrix centered at $\boldsymbol{\theta}$, i.e., $z_i = \boldsymbol{\theta} + \sigma\boldsymbol{\epsilon}_i$, where σ is the noise standard deviation. The following gradient estimator:

$$\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})}[f(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon})] = \frac{1}{\sigma}\mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})}[f(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon})\boldsymbol{\epsilon}] \tag{7}$$

can be estimated with samples

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \approx \frac{1}{m\sigma}\sum_{i=1}^{m} f(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_i \tag{8}$$

and then, parameters $\boldsymbol{\theta}$ are updated iteratively by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ until convergence, where α is the learning rate. In this way, the gradient estimation reduces to sampling unit Gaussian perturbation vectors $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, evaluating the performance (fitness) of the perturbed policies and aggregating the results over a population of search instances.

If the random seeds between workers are synchronized before optimization, each worker can know the perturbations used by other workers. In this way, only a single scalar (fitness) needs to be communicated among the workers to agree on a parameter update, thus resulting in highly parallelizable implementations. More details about ES can be found in [13] and [34].

B. Related Work

While RL algorithms have demonstrated the ability to learn control policies for complex and high-dimensional problems, it is still challenging to apply them to tasks in dynamic environments. A related class of methods in the context of

dynamic environments is transfer RL [22], which reuses the knowledge from a set of related source domains to help the learning task in the target domain. One feasible approach is to use *domain randomization* to learn policies that can work under a large variety of environments. Training a robust policy with domain randomization has been shown to improve the transfer from simulation to reality, also known as "sim-to-real." Tobin *et al.* [36] first trained an object detector with randomized appearances in simulation and transferred it to perform the real robotic grasping task. Muratore *et al.* [37] randomized the parameters of the physics simulations to train robust policies that can be applied directly to real second-order nonlinear systems with an approximate probabilistic guarantee on the suboptimality. Sheckells *et al.* [38] showed that, with the aid of fitting a stochastic dynamics model, the learned robust policy can be transferred back to the real vehicle with little loss in predicted performance. These methods require task-specific knowledge to design parameters and the range of the randomized domain. The policy trained over an enormous range of domains may learn a conservative strategy or fail to learn the target task, while a small range may be insufficient of providing sufficient variation for the policy to transfer to uncertain environments.

Instead of learning invariance to environmental dynamics, an alternative solution is to formulate an environment-conditioned policy as a function of the current state and task feature. Chen *et al.* [39] proposed an explicit representation of the hardware variations and used it as additional input to the policy function for each discrete instance of the environment. Yu *et al.* [40] incorporated an online system identification module with history observations to explicitly predict the dynamics parameters, which are provided as the input to a policy to compute appropriate controls. Subsequently, Yu *et al.* [41] leveraged domain randomization for learning a family of policies conditioned on explicit environmental dynamics. When tested in unknown environments, it directly searched for the best policy in the family based on the task performance via covariance matrix adaptation evolution strategies (CMA-ESs). Constructing these environment-conditioned policies necessarily requires structural assumptions about the system's dynamics, which may not hold in the real world. In addition, it may be difficult for more complex systems to identify the dynamics parameters at runtime.

A similar idea is to train an adaptive policy that is able to identify the environmental dynamics and apply actions appropriate for different system dynamics. In the absence of direct knowledge of parameters of interest, the dynamics can be inferred from a history of past states and actions. Peng *et al.* [42] implicitly embedded the system identification module into the policy by using a recurrent model, where the internal memory acts as the summary of past states and actions, thereby providing a mechanism for inferring the system's dynamics from the policy itself. Andrychowicz *et al.* [43] formulated memory-augmented recurrent polices for in-hand manipulation tasks, which admits the possibility to learn an adaptive behavior and implicit system identification on the fly. These adaptive policies can be trained in the assumed source tasks and deployed in the unknown dynamic environment without fine-tuning. However, policies trained over the source distribution may not generalize well when the discrepancy between the target environment and the source is too large.

Another line of research that tries to address dynamic environments is meta-learning, also called learning-to-learn [44]. A recent trend in meta-learning is to find good initial weights from which adaptation can be quickly performed to tasks sampled from a distribution. One such approach is the gradient-based model-agnostic meta-learning (MAML) algorithm [45]. Gajewski *et al.* [46] derived a novel objective that maximizes the diversity of exhibited behaviors and explicitly optimizes the *evolvability* of ES algorithms, i.e., the ability to further adapt to changing circumstances. Houthooft *et al.* [47] evolved a differentiable loss function that is meta-trained via temporal convolutions over the agent's experiences, resulting in faster test time learning on novel tasks sampled from the same distribution. Song *et al.* [48] employed ES algorithms to solve the MAML problem and to train the meta-policy without estimating any second derivatives. In general, existing methods require repeatedly accessing and processing a potentially large distribution of source tasks to provide a reliable knowledge base for target environments that are supposed to be consistent with the source distribution. In contrast, our incremental learning mechanism concentrates on the ability to rapidly learn and adapt in a sequential manner, without any structural assumptions or prior knowledge on the dynamics of the ever-changing environment.

III. Instance Weighted Incremental Evolution Strategies

In this section, we first formulate the incremental learning procedure to address ES algorithms in a dynamic environment. Then, we present the framework of incremental ESs incorporated with the instance weighting mechanism. Next, we introduce two easy-to-implement metrics of instance novelty and instance quality as well as their mixing variant to calculate the weights. Finally, we give the integrated IW-IES algorithm based on the above implementations.

A. Problem Formulation

Throughout this article, we tackle the dynamic environment as a sequence of stationary tasks on a certain timescale. Each task corresponds to the specific environment characteristics during the associated time period. The dynamic environment involves an infinite task distribution $\mathcal{D}$ over time

$$\mathcal{D} = [M_1, \ldots, M_{t-1}, M_t, \ldots] \tag{9}$$

where each $M_t \in \mathcal{M}$ denotes the specific MDP that is stationary during the tth time period and $\mathcal{M}$ denotes the space of MDPs. We assume that the environment changes in terms of the reward and transition functions only while keeping the same state–action space. Suppose that in the $(t-1)$th time period, the optimal parameters $\boldsymbol{\theta}^*_{t-1}$ are obtained by evolving the search distribution as

$$\boldsymbol{\theta}^*_{t-1} = \arg\max_{\boldsymbol{\theta} \in \mathbb{R}^d} J_{M_{t-1}}(\boldsymbol{\theta}). \tag{10}$$

When the environment changes to M_t, the goal of incremental learning is to adjust the previous optimum of policy parameters $\boldsymbol{\theta}^*_{t-1}$ to new $\boldsymbol{\theta}^*_t$ that fit in the new environment

$$\boldsymbol{\theta}^*_t = \arg\max_{\boldsymbol{\theta}\in\mathbb{R}^d} J_{M_t}(\boldsymbol{\theta}) \tag{11}$$

with initialization of $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}^*_{t-1}$. Continually, the optimum of policy parameters is incrementally adjusted to a new one, $(\boldsymbol{\theta}^*_{t+1}, \boldsymbol{\theta}^*_{t+2}, \ldots)$, whenever the environment changes.

Remark 1: As a matter of fact, automatic detection and identification of changes is also an important component of learning in dynamic environments. In this article, we merely concentrate on how ES algorithms enable rapid learning adaptation to the new environment once the change has been detected and identified. It is analogous to investigations in fault-tolerant control or dynamic multiobjective optimization [20], which solely focus on how controllers/algorithms can quickly accommodate dynamic changes while leaving the detection and identification to be approached individually.

B. Framework

In the incremental learning setting, initializing policy parameters from the original environment empirically benefits the evolutionary process when starting to interact with the new environment since the previous optimum has learned some of the feature representations of the state–action space. However, the previous optimum of policy parameters may be a local one that has been overfitted to the original environment, especially when using a nonlinear function approximator such as the (deep) neural networks. This potential drawback in the incremental initialization may degrade the performance of the new evolutionary process in the long term.

Unlike in supervised learning with DNNs, in which local optima are not thought to be a problem [49], the training data in RL are determined by the executed policies associated with the search distribution. Fig. 1 shows a simple example of the 2-D navigation task, where a three-sided wall blocks the previous optimal path to the goal in the new environment. Due to not having adapted to the new environment yet, the search distribution tends to induce policies that perform well in the original environment [e.g., π_1 in Fig. 1(b)] and move around regions in the parameter space adjacent to the previous optimum. Therefore, the training data for the new evolutionary process are probably limited and it may not discover alternative strategies with potentially larger payoffs (e.g., π_2 or π_3). Thus, it probably gets stuck in bad local optima. Here, we give another example. Suppose that a navigation 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.

It can be inferred that, directly updating policy parameters from the previous optima probably hinders the search distribution to effectively explore the new environment, thus slowing down the learning adaptation. To alleviate this problem, we incorporate an instance weighting mechanism with ES to improve its learning adaptation while preserving the speed/scalability benefits of ES. The idea is straightforward; during parameter updating, we assign higher weights to instances out of a population that contain more knowledge on the new environment, thus encouraging the search distribution to move toward new promising regions in the parameter space.

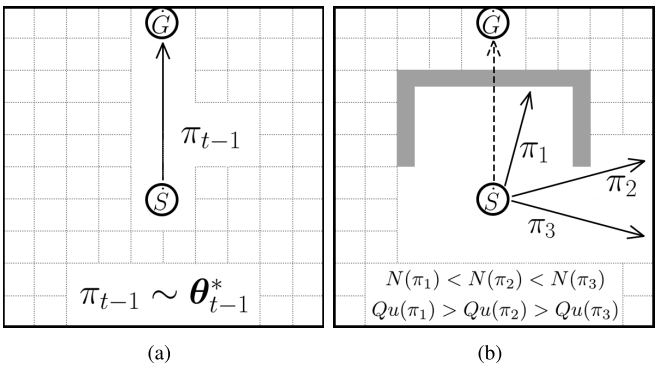


Fig. 1. Simple example of the 2-D navigation task in a dynamic environment. $\dot{S}$ and $\dot{G}$ are the start and goal points, respectively, and the black U-shaped object is a three-sided wall. (a) Original environment M_{t-1}. (b) New environment M_t, where $N(\cdot)$ denotes instance novelty and $Qu(\cdot)$ denotes instance quality.

Recall the gradient estimator in (7) where NES estimates the gradient by taking a sum of sampled parameter perturbations $\boldsymbol{\epsilon}$ weighted by their fitness $f(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon})$, where σ is the noise standard deviation. It rewards instances with high fitness and encourages the search distribution to move toward the direction of those "promising" instances. In a similar spirit, we rearrange the canonical objective function in (2) by multiplying it with a weight $w(z)$ assigned to each instance z as

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}}[w(z)f(z)] = \int w(z)f(z)p(z|\boldsymbol{\theta})\,\mathrm{d}z. \tag{12}$$

Consequently, the gradient estimator in (7) becomes

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) \approx \frac{1}{m\sigma}\sum_{i=1}^{m} w(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}_i)f(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_i. \tag{13}$$

Intuitively, the algorithm follows the approximated gradient in the parameter space toward instances that achieve high fitness of $f(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}_i)$ and exhibit high weights of $w(\boldsymbol{\theta}+\sigma\boldsymbol{\epsilon}_i)$. If the weighting metric can correctly indicate the amount of new knowledge contained by the instance, then the gradient estimator will reward instances with more new knowledge and encourages the search distribution to move toward new promising regions of parameter space that fit in the new environment. The instance weighting mechanism "reinforces" the evolutionary process that searches for well-behaving policies in the new environment, thus improving the learning adaptation to dynamic environments.

Based on the above insight, Algorithm 1 presents the framework of the formalized incremental learning procedure. It is clear that the performance highly depends on how the weight is calculated for each instance. Next, we will introduce the weighting metrics of instance novelty, instance quality, and their mixing variant.

C. Weighting Metrics

1) Instance Novelty: Instance novelty, as our first metric, is designed to indicate the instance's difference from the

Algorithm 1 Incremental Learning Framework

Input: Current time period $t (t \geq 2)$; learning rate α; population size m; noise standard deviation σ
Output: Optimal policy parameters $\boldsymbol{\theta}_t^*$ for M_t

Initialize $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1}^*$, and m CPU workers with known random seeds
while *not converged* **do**
  for *each worker* $i = 1, \ldots, m$ **do**
    Sample $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
    Compute fitness $f_i = f(\boldsymbol{\theta}_t + \sigma \boldsymbol{\epsilon}_i)$
    Calculate and normalize instance weights $w_i = w(\boldsymbol{\theta}_t + \sigma \boldsymbol{\epsilon}_i)$ by metrics in Section III-C
  end
  Send all scalar fitness f_i and weights w_i from each worker to every other worker
  for *each worker* $i = 1, \ldots, m$ **do**
    Reconstruct all perturbations $\boldsymbol{\epsilon}_i$ using known random seeds
    Set $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha \frac{1}{m\sigma} \sum_{i=1}^{m} w_i f_i \boldsymbol{\epsilon}_i$
  end
end

previous optimum in the original environment. Instances with high novelty are supposed to induce different policies from those performing well in the original environment and hence probably contain more knowledge on the new environment. As shown in Fig. 1(b), compared to π_1, the example policy π_2 exhibits the behavior that differs more from the previous optimum and is supposed to reveal higher instance novelty.

To attain feasible computation of such difference, one needs to hand-design or learn an abstract, holistic description of an agent's lifetime of behavior policy. Let π_z denote the executing policy induced by instance z. The policy is assigned a domain-dependent behavior characterization $b(\pi_z)$ that describes its behavior. For example, in the case of a humanoid locomotion problem, $b(\pi_z)$ may be as simple as a 2-D vector containing the humanoid's final coordinate or a concatenation of coordinates that records the humanoid's movement trajectory. Throughout training in the original environment M_{t-1}, the final behavior characterization $b(\pi_{\boldsymbol{\theta}_{t-1}^*})$ can be obtained corresponding to the optimal instance $\boldsymbol{\theta}_{t-1}^*$. Next, in the new environment M_t, a particular instance's novelty $N(\pi_z)$ is calculated by computing the distance between behavior characterizations of this instance and the previous optimum

$$\begin{aligned} N(\pi_z) &= \text{dist}(b(\pi_z), b(\pi_{\boldsymbol{\theta}_{t-1}^*})) \\ &= \left\| b(\pi_z) - b\left(\pi_{\boldsymbol{\theta}_{t-1}^*}\right) \right\|_2. \end{aligned} \tag{14}$$

Here, the Euclidean distance ($L2$-norm) is used for behavior characterizations. However, any distance function can be employed in principle.

Now, the calculated novelty is used as the first metric to assign the instance weight out of a population as

$$w(z_i) = m \cdot \frac{e^{N(\pi_{z_i})/\rho}}{\sum_{j=1}^{m} e^{N(\pi_{z_j})/\rho}}, \quad i = 1, \ldots, m \tag{15}$$

where $\rho \in \mathbb{R}, \rho > 0$ is the temperature hyperparameter for controlling the weight distribution. When ρ becomes larger, all w's will be close to 1, and this weighting metric reduces to uniform weighting. In practice, we increase the temperature ρ by a small increment $\Delta\rho$ at each parameter updating iteration. As the evolution proceeds, the effect of instance weighting gradually becomes weak, and the exploration of novel behaviors decreases. This procedure is akin to the classical "exploration–exploitation tradeoff" in the RL or evolutionary computation community, where exploration is progressively replaced by exploitation as the learning/evolution proceeds.

Remark 2: This metric is related to the concept of curiosity and seeking novelty in RL research and developmental robotics [50], which pushes a learning robot toward novel or curious situations. The notion of novelty is also analogous to that of novelty search algorithms in the evolutionary computation community [14], [51], which is inspired by nature's drive toward diversity and stimulates policies to explore different behaviors from those previously performed.

2) Instance Quality: As it literally means, instance quality corresponds to how well the instance performs in the new environment. Naturally, the performance is evaluated by the received return of the induced learning policy. Since adjusting the previous optimum to a new one under a new data distribution could get stuck in bad local basins, assigning greater importance to high-quality instances can encourage the policies to move toward regions of parameter space that better fit in the new environment, which may be far away from the previous optimum. As shown in Fig. 1(b), due to environmental change, the two example policies, π_2 and π_3, cannot obtain a satisfactory learning performance yet in the new environment. On the other hand, compared with π_3, the policy π_2 is more prone to inducing the new optimal path and receives a higher return in the new environment. It empirically implies that policies receiving higher returns are supposed to be more in line with the new environment and hence contain more new knowledge.

Based on the above observation, a particular instance's quality $Qu(\pi_z)$ can be directly approximated by the received return of its induced policy as

$$Qu(\pi_z) = r(\pi_z). \tag{16}$$

Also, the second metric for assigning the instance weight is calculated as

$$w(z_i) = m \cdot \frac{e^{Qu(\pi_{z_i})/\rho}}{\sum_{j=1}^{m} e^{Qu(\pi_{z_j})/\rho}}, \quad i = 1, \ldots, m. \tag{17}$$

Similarly, we also increase the temperature ρ by a small increment $\Delta\rho$ at each iteration, gradually approaching the form of uniform weighting as the evolution proceeds. During parameter updating, higher importance weights are assigned to episodes that contain more new information, thus encouraging the previous optimum of parameters to be faster adjusted to a new one 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.

3) Mixing Variant: We observe the fact that weighting by instance novelty encourages executing policies to exhibit

different behaviors from those well-performed in the original environment, while weighting by instance quality strengthens the searching for well-performing policies in the new environment. Recalling the example in Fig. 1(b), we have $N(\pi_1) < N(\pi_2) < N(\pi_3)$ and $Qu(\pi_1) > Qu(\pi_2) > Qu(\pi_3)$. Using novelty only as the weighting metric may overvalue instances with bad performance (e.g., π_3), while using quality only may lead to policies getting stuck in a deceptive trap (e.g., π_1). Therefore, to make the most use of the two metrics, we explore a mixing variant to calculate the weight as

$$w(\boldsymbol{z}_i) = m \cdot \frac{e^{N(\pi_{z_i})\cdot Qu(\pi_{z_i})/\rho}}{\sum_{j=1}^{m} e^{N(\pi_{z_j})\cdot Qu(\pi_{z_j})/\rho}}, \quad i = 1, \ldots, m. \tag{18}$$

The following experimental results show that incremental ESs with the mixing weighting metric generally perform the best among all compared implementations.

D. Integrated Algorithm

Based on the problem formulation and the instance weighting mechanism with designed metrics, Algorithm 2 presents the integrated IW-IES algorithm for RL in dynamic environments. In the incremental learning setting, an RL agent is interacting with a dynamic environment $\mathcal{D} = [M_1, M_2, \ldots]$. In the first time period, the policy parameters are randomly initialized in Line 3, followed by the evolutionary process from scratch using the canonical NES algorithm in Lines 4–14. In a subsequent tth ($t \geq 2$) time period (new environment), we first obtain the previous optimal policy parameters $\boldsymbol{\theta}^*_{t-1}$ and the corresponding behavior characterization $b(\pi_{\boldsymbol{\theta}^*_{t-1}})$ from the last time period (original environment) in Line 16. Then, the policy parameters are initialized from the previous optimum in Line 17. In Line 22, we assign a weight to each instance according to the designed metrics, aiming at facilitating the learning adaptation to the new environment. Finally, the policy parameters iteratively evolve in Line 27 until the new optimum $\boldsymbol{\theta}^*_t$ is obtained for M_t.

The use of specific metrics for the instance weighting mechanism yields three variants of implementations: 1) IW-IES-N: using instance novelty to calculate weights in (15); 2) IW-IES-Qu: using instance quality as the weighting metric in (17); and 3) IW-IES-Mix: using the mixing weighting metric in (18). Instance novelty is designed to measure the instance's difference from the previous optimum of the original environment, while instance quality corresponds to how well the instance performs in the new environment. Together, an instance with high novelty or high quality is supposed to contain more knowledge on the new environment. With this mechanism, IW-IES prefers behaviors that either differ more from the original environment or better fit in the new environment, thus encouraging the search distribution to move toward new promising regions of parameter space.

Remark 3 (Scalability): As shown in [13], ES scales well with the amount of computation available, enabling a near-linear speedup in runtime as more CPUs are used. The proposed algorithm, IW-IES, enjoys the same parallelization benefits as ES because it uses an almost identical optimization process. In IW-IES, broadcasting both scalars of fitness $f(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$ and instance weight $w(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$ would incur almost zero extra network overhead because the scalars usually take up much less memory than the large parameter vector $\boldsymbol{\theta}_t$ that must be broadcast at the beginning of each iteration. Moreover, the addition of the behavior characterization of previous optimal policy does not hurt scalability because it is kept fixed during the calculation of instance novelty and the coordinator needs to broadcast it only once at the beginning of each iteration.

Remark 4 (Complexity): Here, we also give a rough complexity analysis on the designed metrics. First, for calculating instance novelty $N(\pi_z)$, we need to compute each instance's behavior characterization $b(\pi_z)$, which can be simultaneously acquired when primitively executing the associated behavior

Algorithm 2 IW-IES

Input: Dynamic environment $\mathcal{D} = \{M_1, M_2, \ldots\}$; current time period $t\,(t \geq 1)$; learning rate α; population size m; noise standard deviation σ; increment of temperature $\Delta\rho$

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

1 Initialize: m CPU workers with known random seeds
2 **if** *t equals to 1* **then**
3 Randomly initialize $\boldsymbol{\theta}_t$
4 **while** *not converged* **do**
5 **for** *each worker* $i = 1, \ldots, m$ **do**
6 Sample $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
7 Compute fitness $f_i = f(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$
8 **end**
9 Send all scalar fitness f_i from each worker to every other worker
10 **for** *each worker* $i = 1, \ldots, m$ **do**
11 Reconstruct all perturbations $\boldsymbol{\epsilon}_i$ using known random seeds
12 Set $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha\frac{1}{m\sigma}\sum_{i=1}^{m} f_i\boldsymbol{\epsilon}_i$
13 **end**
14 **end**
15 **else**
16 Obtain $\boldsymbol{\theta}^*_{t-1}$ and behavior characterization $b(\pi_{\boldsymbol{\theta}^*_{t-1}})$
17 Initialize: $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}^*_{t-1}$, and the temperature ρ
18 **while** *not converged* **do**
19 **for** *each worker* $i = 1, \ldots, m$ **do**
20 Sample $\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$
21 Compute fitness $f_i = f(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$
22 Calculate instance weights $w_i = w(\boldsymbol{\theta}_t + \sigma\boldsymbol{\epsilon}_i)$ by metrics in (15), (17), or (18)
23 **end**
24 Send all scalar fitness f_i and weights w_i from each worker to every other worker
25 **for** *each worker* $i = 1, \ldots, m$ **do**
26 Reconstruct all perturbations $\boldsymbol{\epsilon}_i$ using known random seeds
27 Set $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_t + \alpha\frac{1}{m\sigma}\sum_{i=1}^{m} w_i f_i\boldsymbol{\epsilon}_i$
28 **end**
29 $\rho \leftarrow \rho + \Delta\rho$
30 **end**
31 **end**

policy to compute its fitness. For example, in the humanoid locomotion case, the observed reward and state can be used for calculating the fitness and the behavior characterization, respectively. Besides, the behavior characterization usually consumes much less memory than the parameter vector of instance z. Second, calculating instance quality $Qu(\pi_z)$ would not consume any computation for extra variables since the quality indicator, i.e., the received return $r(\pi_z)$, has been already acquired in the primitive step for computing the fitness. Together, it can be inferred that the designed weighting metrics hardly increase the computational complexity of ES. Hence, we claim that the two metrics are "easy-to-implement."

IV. Experiments

To test IW-IES, we conduct experiments on challenging RL tasks ranging from classical navigation tasks to benchmark MuJoCo robot locomotion tasks [52]. Using agents in these tasks, we design challenging dynamic environments that involve (multiple) changes in the underlying environment distribution, where incremental learning is critical. Through these experiments, we aim to build problem settings that are representative types of disturbances and shifts that a real RL agent may encounter in practical applications. The questions that we aim to study from our experiments include the following.

- *Q1:* Can IW-IES handle various dynamic environments where the reward or state transition function changes over time?
- *Q2:* Can IW-IES successfully facilitate rapid learning adaptation to these dynamic environments?
- *Q3:* How do the two weighting metrics, instance novelty and instance quality, affect the performance of IW-IES?

A. Experimental Settings

We compare IW-IES to four baseline methods.

1) *Robust [38]:* It takes the most recent observation (i.e., $\pi_{\text{robust}} : s \mapsto a$) as input and uses domain randomization to train a robust policy that is supposed to work for all training environments, while current environmental dynamics cannot be identified from its input.
2) *SO-CMA [41]:* It uses the environment feature μ as additional input (i.e., $\pi_{\text{so}} : [s, \mu] \mapsto a$) and trains an environment-conditioned policy with domain randomization. Given particular μ, the instantiated policy is called a strategy. In the target environment, it performs strategy optimization using CMA-ES, which only optimizes the environment feature input to the policy.
3) *Hist [42]:* The adaptive policy is represented as a long short-term memory (LSTM) network that takes a history of observations as input, i.e., $\pi_{\text{adapt}} : [s_{t-h}, \ldots, s_t] \mapsto a$. This allows the policy to implicitly identify the environment being tested and to adaptively choose actions based on the identified environment.
4) *ES-MAML [48]:* It trains a meta-policy on a variety of tasks based on the NES algorithm such that it can solve new learning tasks using only a small number of training samples. More details about MAML can be found in [45].

In all domains, the policy model evolved by IW-IES is instantiated as a feedforward neural network with two 128-unit hidden layers separated by ReLU nonlinearities, similar to the benchmark network architectures used in [13] and [14]. For a fair comparison to our method, the network architecture of Robust and ES-MAML is set the same as that of IW-IES. SO-CMA uses the same network architecture except that it takes the environment feature as additional input. For Hist, we feed a history of five observations to a recurrent policy network that consists of a 64-unit embedding layer and a 64-unit LSTM layer separated by ReLU nonlinearities. In this way, the recurrent policy network has the same order of magnitude number of parameters as the policy model of IW-IES. While existing methods mostly use policy gradient algorithms to train neural network policies, we implement these baseline methods by the NES algorithm as a more challenging reference point. The number of CPU workers is set as $m = 16$ for parallelizing IW-IES and all baseline methods. The universal polices of baselines are trained over a variety of environments that are randomly sampled from a known distribution. For SO-CMA, the environment feature is assumed to essentially capture the MDP drawn from the distribution, which, for instance, can be represented by the position of goals or obstacles in a navigation task. Furthermore, we continue to train the universal polices after transferring to the new task whenever the environment changes, using the same number of samples IW-IES consuming in each environment. We refer to this additional training step as "fine-tuning." In contrast, IW-IES focuses on directly adapting to dynamic environments on the fly, avoiding access to a large distribution of training environments, and releasing the dependency on structural assumptions of environmental dynamics.

For each report unit (a particular algorithm running on a particular task), we define two performance metrics. One is the received return for executing the policy induced by the unperturbed instance in each evolution generation, defined as $r(\pi_\theta)$. The other is the average return received over all generations, defined as $(1/I)\sum_{i=1}^{I} r_i(\pi_\theta)$, where I is the number of total evolving generations. The former will be plotted in figures and the latter will be presented in tables. Due to the randomness of training neural networks, we run three trials with different seeds and adopt the mean as the performance for each report unit. We utilize a statistical analysis method to address the issue of "stochastic" dynamic environments. The learning agent first learns 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 ten times and report the mean and standard error to demonstrate the performance for learning in stochastic dynamic environments. The code is available online.[2]

B. Navigation Tasks

We first test our IW-IES algorithms on a set of navigation tasks where a point agent must move to a goal position in 2-D

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

within a unit square. The state is the current observation of the 2-D position, and the action corresponds to the 2-D velocity commands that are clipped to be in the range of $[-0.1, 0.1]$. The reward is the negative squared distance to the goal minus a small control cost that is proportional to the action's scale. Each learning episode (generated by the instance $\boldsymbol{z}$) always starts from a given point and terminates when the agent is within 0.01 of the goal or at the horizon of $H = 100$. The hyperparameters are set as: population size $m = 16$ and noise standard deviation $\sigma = 0.05$. The learning rate is set as the same for all tested methods in each task.

As described in Section III-C, the first weighting metric requires a domain-specific behavior characterization for calculating the novelty of each instance $\boldsymbol{z}$. For the navigation problems, the behavior characterization is the trace of the agent's (x, y) locations through all time steps

$$b(\pi_z) = \left\{\left(x_z^1, y_z^1\right), \ldots, \left(x_z^H, y_z^H\right)\right\}. \tag{19}$$

Computing instance novelty also requires a distance function between behavior characterizations of the instance $\boldsymbol{z}$ and the previous optimum $\boldsymbol{\theta}_{t-1}^*$. Following [14] and [51], we use the average Euclidean distance of these 2-D coordinates as the distance function:

$$\text{dist}\left(b(\pi_z), b\left(\pi_{\boldsymbol{\theta}_{t-1}^*}\right)\right) = \frac{1}{H}\sum_{i=1}^{H}\sqrt{\left(x_z^i - x_{t-1}^i\right)^2 + \left(y_z^i - y_{t-1}^i\right)^2} \tag{20}$$

where $\{(x_{t-1}^i, y_{t-1}^i)\}_{i=1}^H$ are the agent's locations when executing the previous optimal policy $\pi_{\boldsymbol{\theta}_{t-1}^*}$.

1) Q1: We start with two illustrative cases of simulated dynamic environments, as shown in Fig. 2.

1) *Case I:* The dynamic environment is created by randomly changing the goal position while keeping the start point fixed at $(0, 0)$. The environment changes in terms of the reward function, and the goal position can be used as the environment feature for SO-CMA.
2) *Case II:* The start and goal points are kept fixed at $(0, -0.5)$ and $(0, 0.5)$, respectively. The dynamic environment is created by moving a 0.6×0.6 square obstacle at random. When hitting on the obstacle, the agent will bounce to its previous position. The environment changes in terms of the state transition function, and the environment feature can be represented by the centering position of the square obstacle.

2) Q2: We present the primary experimental results of baselines and IW-IES implemented on the two cases of dynamic environments. The average return (of the executing policy induced by the unperturbed instance) per generation across ten independent runs is plotted in Fig. 3. Here and in similar figures in the following, the mean of average return per generation across ten runs is plotted as the bold line with 95% bootstrapped confidence intervals of the mean (shaded). Furthermore, Table I reports the numerical results in terms of average received return over 200 training generations for Case I and over 1000 generations for Case II. Here and in similar tables in the following, the mean across ten runs is presented, and the confidence intervals are corresponding standard errors. The best performance is marked in boldface.

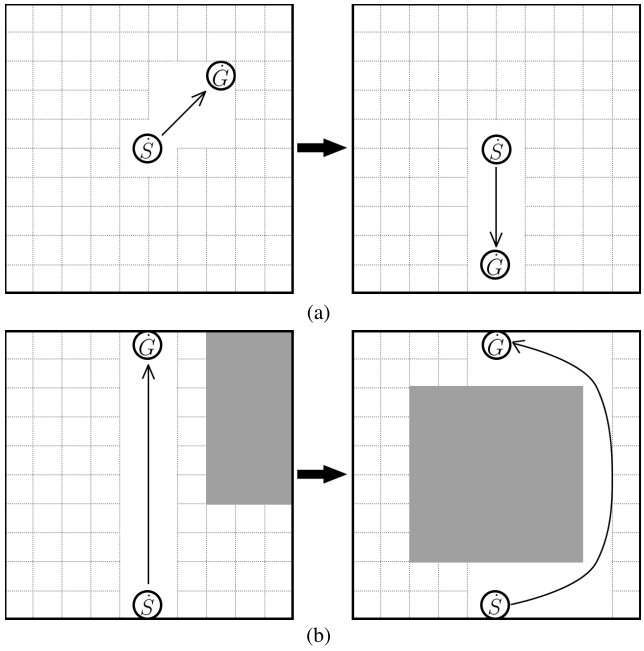


Fig. 2. Two illustrative cases of dynamic environments in the 2-D navigation tasks. $\dot{S}$ is the start point and $\dot{G}$ is the goal point. (a) Case I: goal changes. (b) Case II: landscape changes and the gray is a square obstacle.

TABLE I
AVERAGE RECEIVED RETURN OVER ALL TRAINING GENERATIONS OF BASELINES AND IW-IES IMPLEMENTED ON TWO NAVIGATION TASKS

Methods	Case I	Case II
Robust	-16.99 ± 2.08	-79.59 ± 1.10
SO-CMA	-25.99 ± 4.57	-79.79 ± 1.40
Hist	-17.22 ± 1.85	-79.55 ± 1.19
ES-MAML	-15.69 ± 2.03	-79.90 ± 0.86
IW-IES	$\mathbf{-8.65 \pm 1.04}$	$\mathbf{-31.43 \pm 0.77}$

In Case I, since all baselines pretrain the policy model over a large distribution of randomized source environments, they receive higher jumpstart return than IW-IES when start interacting with the new environment. SO-CMA obtains good jumpstart performance in the beginning. However, in the latter learning process, it receives nonincreasing return that is smaller than other baseline methods. SO-CMA can adapt to dynamic environments with fewer data by optimizing only the environment feature input to the policy [41], whereas its final performance may be inferior to methods that adjust the neural network weights in the fine-tuning phase. By comparison, despite obtaining smaller return initially, IW-IES exhibits significantly faster learning adaptation to dynamic environments than all baselines. For instance, IW-IES receives near-optimal return with only 40 generations, whereas Robust, Hist, and ES-MAML need to take more than 200 generations for achieving comparable performance. In addition, the statistical results show that IW-IES obtains smaller confidence intervals and standard errors than all baselines, indicating that IW-IES can provide more stable learning adaptation to new environments.

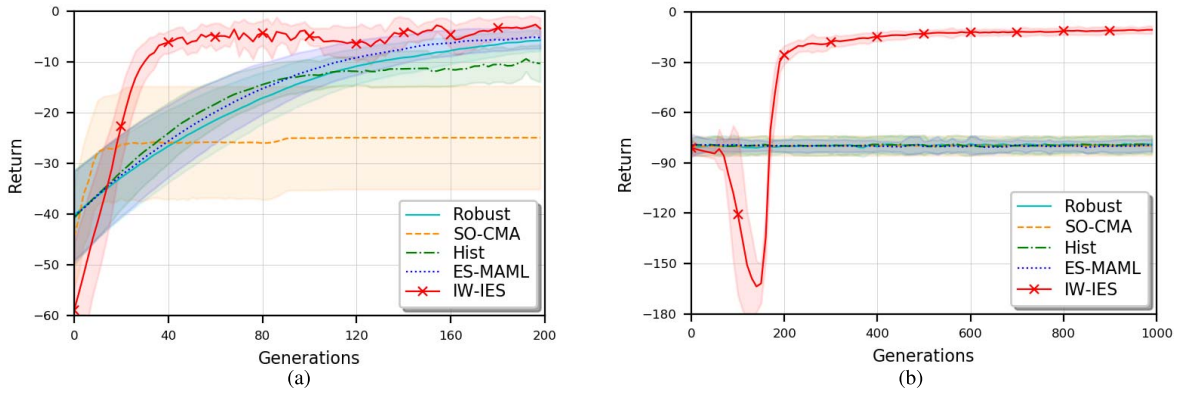


Fig. 3. Received return per generation of baselines and IW-IES implemented on two navigation tasks. (a) Case I. (b) Case II.

Next, the results in Case II reveal some differences. In Case I, IW-IES achieves faster learning adaptation than baselines, while the performance gap is relatively moderate. The reason is that, in this simple case of navigation, all baselines can evolve near-optimal policies that will find a path to the new goal, as shown in Fig. 2(a). In contrast, IW-IES obtains much higher final return in Case II. Since navigating to the goal while bypassing the wall is usually difficult, policies evolved by baselines tend to head directly toward the goal. Thus, it easily terminates in front of the huge obstacle and gets stuck in bad local optima. Instead, the proposed instance weighting mechanism "reinforces" the policy evolved by IW-IES to explore behaviors that are different from previous optima (instance novelty) and to exploit behaviors that are in line with the new environment (instance quality). Therefore, the policy evolved by IW-IES is more capable of bypassing the deceptive wall first and moving to the goal finally. The primary results verify that the instance weighting mechanism effectively encourages the search distribution to move toward promising regions of parameter space that fit in the new environment. IW-IES can obtain superior performance compared to all baselines, demonstrating the effectiveness of our method for addressing incremental learning problems in dynamic environments.

In the above experiments, the original and new environments are sampled from the same distribution. In addition, we employ the Case I navigation task to serve as an illustrative example to test the performance of baselines and IW-IES when the discrepancy between distributions of the original and new environments is large. The universal policies of baselines are trained over a limited range of environments where goal positions are in the first quadrant. Then, they are transferred to and fine-tuned in new environments where goal positions are in the third quadrant. For IW-IES, the original and new environments are sampled from the same distributions as those of baselines. Fig. 4 presents the received return per generation of baselines and IW-IES, and Table II shows the corresponding numerical results. It is observed that the advantage of IW-IES over baselines is more prominent than the case with no discrepancy of environment distributions, as shown in Fig. 3(a). Consistent with the analysis in Section II-B, existing methods rely on transferring the knowledge trained over a large distribution of source tasks to new environments that are in line with the source distribution. The universal policies trained by baselines may not generalize well when the discrepancy between distributions of the original and new environments is too large. In contrast, IW-IES can stably facilitate learning adaptation to dynamic environments regardless of the distribution discrepancy.

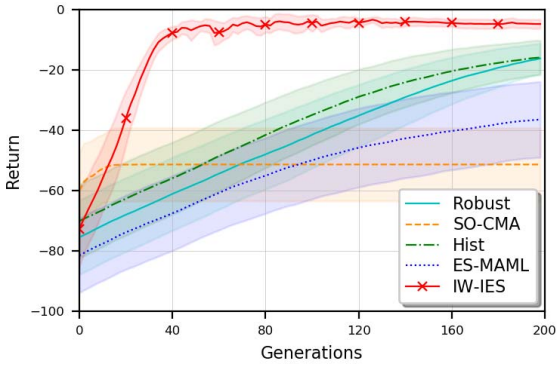


Fig. 4. Received return per generation of baselines and IW-IES in Case I navigation task with a large discrepancy between environment distributions.

TABLE II
AVERAGE RECEIVED RETURN OVER ALL GENERATIONS OF BASELINES AND IW-IES IMPLEMENTED ON CASE I NAVIGATION TASKS

Methods	Without discrepancy	With discrepancy
Robust	-16.99 ± 2.08	-42.38 ± 4.72
SO-CMA	-25.99 ± 4.57	-51.67 ± 5.34
Hist	-17.22 ± 1.85	-38.05 ± 4.26
ES-MAML	-15.69 ± 2.03	-53.49 ± 5.58
IW-IES	$\mathbf{-8.65 \pm 1.04}$	$\mathbf{-11.57 \pm 1.03}$

3) Q3: To identify the respective effects of the two weighting metrics, we adopt a control variate approach to separate them apart for observation. In each task, we first initialize policy parameters from the original environment and then implement four variants of IW-IES according to the employed weighting metric.

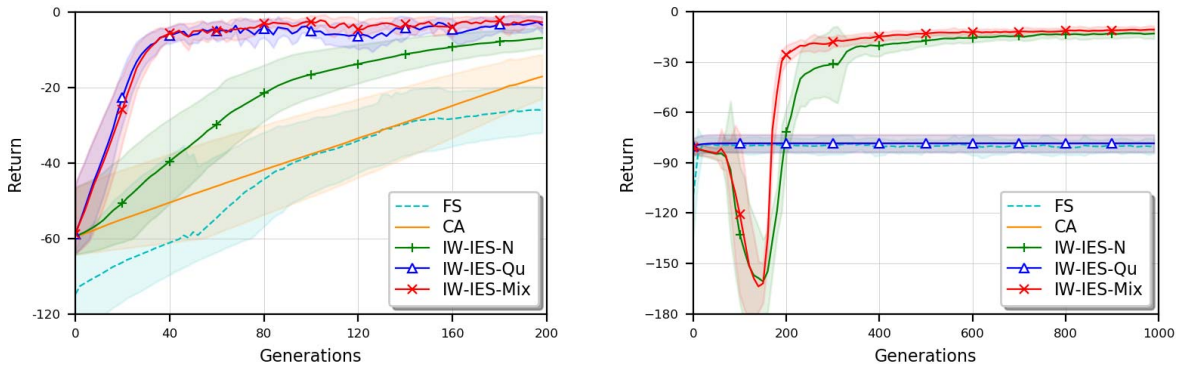


Fig. 5. Received return per generation of the four variants of IW-IES implemented on two navigation tasks. (a) Case I. (b) Case II.

TABLE III

AVERAGE RECEIVED RETURN OVER ALL TRAINING GENERATIONS OF THE FOUR VARIANTS OF IW-IES AND FS ON TWO NAVIGATION TASKS

Methods	Case I	Case II
FS	-49.12 ± 5.42	-79.53 ± 1.06
CA	-37.94 ± 4.70	-78.48 ± 1.26
IW-IES-N	-23.53 ± 2.97	-38.59 ± 1.57
IW-IES-Qu	-9.34 ± 0.93	-78.50 ± 1.26
IW-IES-Mix	$\mathbf{-8.65 \pm 1.04}$	$\mathbf{-31.43 \pm 0.77}$

TABLE IV

RUNTIME (SECONDS) OF THE FOUR VARIANTS OF IW-IES WITH VARYING NUMBERS OF CPU WORKERS ON CASE I NAVIGATION TASK

# CPU workers	1	2	3	4	5
CA	141.84	81.82	59.04	49.88	41.21
IW-IES-N	141.23	83.72	58.72	49.92	40.77
IW-IES-Qu	141.51	82.23	58.65	50.14	40.94
IW-IES-Mix	141.11	82.87	57.81	48.87	41.09

1) *CA:* No instance weighting mechanism is applied, i.e., continuously adapting a single policy model whenever the environment changes. This is representative of commonly used dynamic evaluation methods [19].
2) *IW-IES-N:* Use instance novelty in (15) as the metric.
3) *IW-IES-Qu:* Use instance quality in (17) as the metric.
4) *IW-IES-Mix:* Use the mixing weighting metric in (18).

In addition, we also investigate the performance of the policy trained with randomly initialized parameters whenever the environment changes, i.e., learning from scratch (FS). For the four variants and FS, Fig. 5 presents their learning performance per evolution generation, and Table III reports the average received return over all training generations.

We observe that CA usually has a better performance than FS regarding the adaptation speed and the average received return. Especially, CA can achieve jumpstart performance at the beginning of the new learning process compared to FS. It is consistent with the analysis in Section III-B that, initializing policy parameters from the original environment empirically benefits the evolutionary process since the previous optimum has learned some of the feature representations of the state–action space. This is also a basic impetus for the formalized incremental learning procedure in this article.

In Case I, using instance novelty or instance quality alone as the weighting metric can achieve faster learning adaptation to the dynamic environment. The performance gap in terms of average return is more pronounced for smaller amount of computation, which is supposed to benefit from the instance weighting mechanism that allows for distinct acceleration of incremental learning adaptation. The weighting metric using instance quality (IW-IES-Qu) better improves the learning performance than the one using instance novelty (IW-IES-N) and combining the two weighting metrics together (IW-IES-Mix) achieves slightly better performance than IW-IES-Qu.

In Case II, both CA and IW-IES-Qu fail to find the new goal when the huge obstacle blocks the previous optimal path. In contrast, introducing the weighting mechanism with instance novelty will degrade the learning performance in initial generations. To bypass the obstacle in this "deceptive" case, both IW-IES-N and IW-IES-Mix need to encourage the learning agent to exhibit novel behaviors to a large extent, thus resulting in the temporarily pessimistic performance in the early stage. As the evolution proceeds, the effect of instance weighting becomes weaker, and they can gradually find the optimal path to the new goal instead of getting stuck in front of the wall. In this case, IW-IES-N better improves the learning performance than IW-IES-Qu, and IW-IES-Mix obtains the best learning adaptation to the dynamic environment where the obstacle changes over time.

In summary, the two weighting metrics of instance novelty and instance quality can provide distinguished advantages for different kinds of dynamic environments. Under most circumstances, combining the two weighting metrics together leads to the best learning adaptation.

Furthermore, we employ the Case I navigation task to serve as an illustrative example to verify the scalability of IW-IES. Table IV shows the runtime of the four variants of IW-IES with varying numbers of parallelized CPU workers. It is observed that IW-IES retains scalability and enables a near-linear speedup in runtime as more CPU workers are

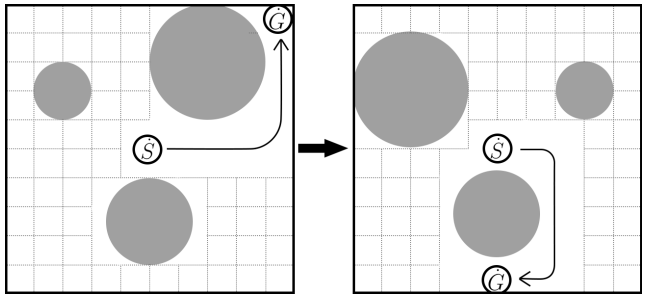


Fig. 6. 2-D navigation task in a complex stochastic dynamic environment, where both the goal and circular puddles may change over time.

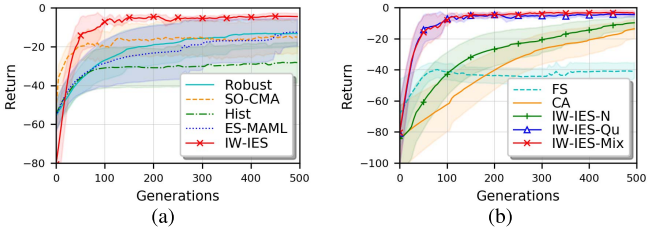


Fig. 7. Received return per generation of all tested methods implemented on the complex stochastic dynamic environment. (a) Baselines and IW-IES. (b) FS and four variants of IW-IES.

used. Obviously, introducing the instance novelty or instance quality as the weighting metric hardly increases the runtime of our method. The result empirically verifies the claim in Remark 4 that the designed weighting metrics hardly increase the computational complexity of ES.

4) Complex Stochastic Dynamic Environments: It is verified from the above results that IW-IES can successfully facilitate the learning adaptation to dynamic environments where the reward or state transition function may change over time. Here, we test IW-IES on a more complex case of dynamic environments, which is also a modified version of the benchmark puddle world environment presented in [53] and [54]. As shown in Fig. 6, the agent should drive to the goal while avoiding three circular puddles of different sizes. When hitting on the puddles, the agent will bounce to its previous position. The dynamic environment is created by moving both the goal and puddles within the unit square randomly. Hence, the environment changes in terms of both the reward and state transition functions. Fig. 7 shows the received return per generation of the baselines, FS, and the four IW-IES variants, and Table V presents the corresponding numerical results in terms of average return over 500 evolving generations of all tested methods. We can observe that IW-IES is still capable of achieving significantly faster learning adaptation to this complex stochastic dynamic environment in a statistical sense.

From the above comprehensive experimental results on 2-D navigation tasks, it can be well demonstrated that the following conditions hold.

A1: IW-IES is able to handle various dynamic environments that change in terms of the reward function [Fig. 2(a)], the state transition function [Fig. 2(b)], or both (Fig. 6).

A2: IW-IES successfully enables faster and more stable learning adaptation to these dynamic environments.

TABLE V
AVERAGE RETURN OVER 500 EVOLVING GENERATIONS OF ALL TESTED METHODS ON THE COMPLEX STOCHASTIC DYNAMIC ENVIRONMENT

Methods	Average return	Methods	Average return
Robust	-21.42 ± 3.63	FS	-42.91 ± 2.23
SO-CMA	-17.71 ± 3.54	CA	-39.67 ± 4.72
Hist	-31.65 ± 4.32	IW-IES-N	-29.97 ± 3.77
ES-MAML	-23.46 ± 4.74	IW-IES-Qu	-9.70 ± 0.85
IW-IES	$\mathbf{-8.85 \pm 1.06}$	IW-IES-Mix	$\mathbf{-8.85 \pm 1.06}$

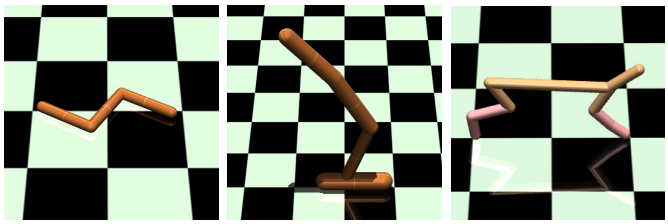


Fig. 8. Challenging MuJoCo locomotion tasks including (a) Swimmer, $|S| = 8$ and $|A| = 2$; (b) Hopper, $|S| = 11$ and $|A| = 3$; and (c) HalfCheetah, $|S| = 20$ and $|A| = 6$.

A3: The two weighting metrics of instance novelty and instance quality can offer distinct superiority for incremental learning in different kinds of dynamic environments, and combining them together usually achieves the best learning adaptation.

C. Locomotion Tasks

The above results illustrate that IW-IES is simply well suited to the 2-D navigation domains, and it significantly facilitates the learning adaption to various dynamic environments. A natural question is whether IW-IES can be successfully applied to more difficult domains. It is necessary to test IW-IES on a well-known problem of considerable difficulty. Thus, we also investigate three high-dimensional locomotion tasks with the MuJoCo simulator [52], aiming at testing whether IW-IES can achieve locomotion at the scale of DNNs on much more sophisticated dynamic environments. As shown in Fig. 8, the continuous control tasks require a swimmer/one-legged hopper/planar cheetah robot to move at a particular velocity in the positive x-direction. These three scenarios are representative locomotion tasks with growing dimensions of state and action spaces. The reward is an alive bonus plus a regular part that is negatively correlated with the absolute value between the agent's velocity and a goal. The goal velocity is randomly chosen between: $[0, 0.5]$ for Swimmer, $[0, 1]$ for Hopper, and $[0, 2]$ for HalfCheetah. We also simulate a stochastic dynamic environment by changing the goal velocity at random across ten independent runs.

In the locomotion domains, the behavior characterization for calculating instance novelty should be able to distinguish the robot's gaits in the x-direction, in accordance with the internal learning tasks. According to this principle, a feasible behavior characterization is the offset of the robot's coordinate in the x-direction through all time steps

$$b(\pi_z) = \left\{x_z^1 - x_z^0, \ldots, x_z^H - x_z^0\right\} \tag{21}$$

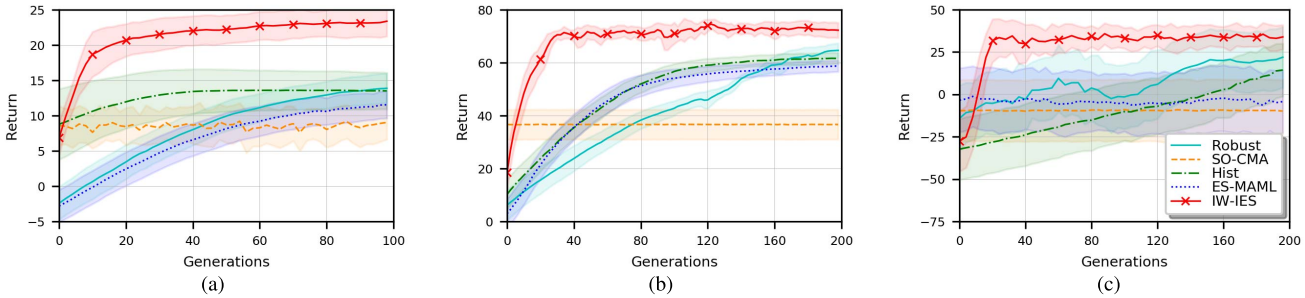


Fig. 9. Received return per generation of baselines and IW-IES on the challenging locomotion tasks. (a) Swimmer. (b) Hopper. (c) HalfCheetah.

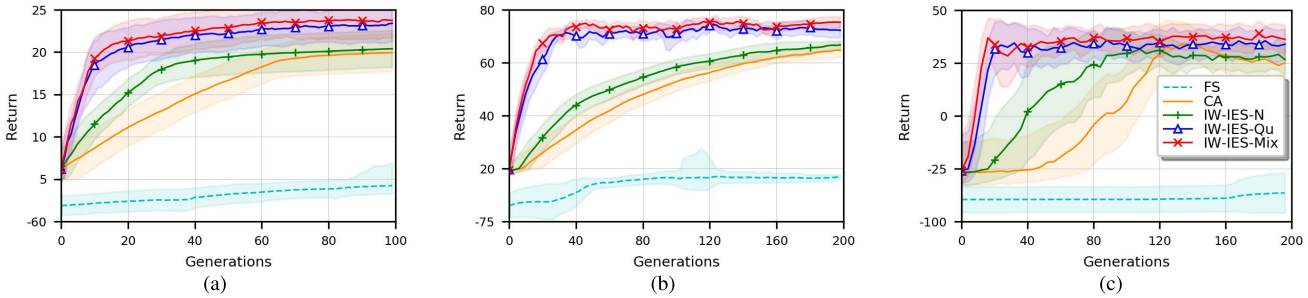


Fig. 10. Received return per generation of FS and the four variants of IW-IES on locomotion tasks. (a) Swimmer. (b) Hopper. (c) HalfCheetah.

where x^0 and x^i $(i = 1, \ldots, H)$ correspond to the offsets of center of mass along the x-direction at the initial and the ith time steps. Consistently with the 2-D navigation domains, we use the average Euclidean distance of the 1-D coordinates as the distance function between behavior characterizations of instance $\boldsymbol{z}$ and the previous optimum $\boldsymbol{\theta}^*_{t-1}$

$$\text{dist}\left(b(\pi_z), b\left(\pi_{\boldsymbol{\theta}^*_{t-1}}\right)\right) = \frac{1}{H}\sum_{i=1}^{H}\left|\left(x_z^i - x_z^0\right) - \left(x_{t-1}^i - x_{t-1}^0\right)\right| \tag{22}$$

where $\{x_{t-1}^i - x_{t-1}^0\}_{i=1}^H$ is the behavior characterization associated with the previous optimal policy $\pi_{\boldsymbol{\theta}^*_{t-1}}$.

With the above settings, we present the results of all tested methods implemented on the challenging locomotion domains. Fig. 9 shows the received return per generation of baselines and IW-IES, and Fig. 10 presents the received return per generation of FS and the four variants of IW-IES. The corresponding numerical results in terms of average received return over all training generations are reported in Table VI. Primarily, it is observed that IW-IES achieves more stable and faster learning adaptation in these locomotion tasks than the four baseline methods, which demonstrates the effectiveness of our method for addressing incremental learning problems in Mujoco locomotion domains.

Next, the four variants of IW-IES usually exhibit much faster learning adaptation than FS, especially in the early stage. The phenomenon indicates that the locomotion skills learned in the original environment can benefit the new learning process a lot. It also demonstrates the effectiveness of the proposed incremental learning mechanism, that is, the previously learned policy empirically performs better than a randomly initialized one because it has learned some of the feature representations of the state–action space. Furthermore, IW-IES-N, IW-IES-Qu, and IW-IES-Mix always exhibit faster learning adaptation than CA. It verifies that using either instance novelty or instance quality as the weighting metric can already enhance the incremental learning performance and enable significantly rapid learning adaptation. In Mujoco locomotion domains, the weighting metric using instance quality can better boost the incremental learning performance than using instance novelty, and using the mixing variant usually leads to the best learning adaptation to these challenging dynamic environments. By the instance weighting mechanism that emphasizes new knowledge, IW-IES rapidly guides the policy toward regions of parameter space that better fit in the new environment. In summary, the results demonstrate that IW-IES is also capable of facilitating learning adaptation to

TABLE VI
AVERAGE RECEIVED RETURN OVER ALL GENERATIONS OF BASELINES, FS, AND FOUR VARIANTS OF IW-IES ON MUJOCO LOCOMOTION TASKS

Methods	Swimmer	Hopper	HalfCheetah
Robust	8.22 ± 0.96	40.74 ± 1.15	6.56 ± 7.98
SO-CMA	8.55 ± 1.00	36.64 ± 2.49	-9.49 ± 8.02
Hist	12.75 ± 1.4	48.89 ± 1.39	-10.63 ± 7.61
ES-MAML	6.62 ± 0.97	46.57 ± 1.58	-4.13 ± 8.00
IW-IES	$\mathbf{21.78 \pm 0.92}$	$\mathbf{70.95 \pm 0.64}$	$\mathbf{32.48 \pm 2.73}$
FS	-17.48 ± 5.68	-8.64 ± 3.51	-70.12 ± 8.47
CA	15.41 ± 0.96	48.35 ± 1.62	3.57 ± 6.02
IW-IES-N	17.63 ± 0.89	53.19 ± 1.44	15.31 ± 5.18
IW-IES-Qu	21.14 ± 0.94	68.81 ± 0.84	29.19 ± 3.08
IW-IES-Mix	$\mathbf{21.78 \pm 0.92}$	$\mathbf{70.95 \pm 0.64}$	$\mathbf{32.48 \pm 2.73}$

stochastic dynamic environments for these high-dimensional locomotion tasks.

V. Conclusion

ES algorithms are recently shown to be capable of solving challenging, high-dimensional RL tasks, while being much faster with many CPUs due to better parallelization. In this article, we investigate incremental ES algorithms for RL in dynamic environments. We hybridize an instance weighting mechanism with ES to enable rapid learning adaptation while preserving scalability of ES. During parameter updating, higher weights are assigned to instances that contain more new knowledge, thus encouraging the search distribution to move toward new promising areas in the parameter space. The designed weighting metrics, instance novelty and instance quality, "reinforce" the evolutionary process that searches for new well-behaving policies. The proposed IW-IES algorithm is tested on traditional navigation and challenging locomotion domains with varying configurations. Experiments verify that IW-IES is capable of significantly facilitating learning adaptation to various dynamic environments.

Thus, we provide an option for not only taking advantage of the scalability of ES but also pursuing better learning adaptation to dynamic environments from an incremental learning perspective. The latter scenario is supposed to hold for most challenging, real-world domains that ES/RL practitioners will wish to tackle in the future. Our future work will focus on learning adaptation in more challenging dynamic environments where the state–action space changes over time or learning in more intensively changing environments (e.g., change between consecutive learning episodes). Automatic detection and identification of environmental change is also a crucial direction to be addressed. Another insightful direction would be to conduct an empirical investigation on systematically comparing the derivative-free ES algorithms and the derivative-based optimization methods [55] in RL domains or to develop possible off-policy solutions for incorporating the experience replay mechanism [56] with ES.

References

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

[2] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 18, no. 4, pp. 973–992, Jul. 2007.

[3] B. Luo, D. Liu, T. Huang, and D. Wang, "Model-free optimal tracking control via critic-only Q-learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 10, pp. 2134–2144, Oct. 2016.

[4] 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.

[5] H. Li, Z. Qichao, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 2064–2076, Jun. 2020.

[6] J.-A. Li *et al.*, "Quantum reinforcement learning during human decision-making," *Nature Hum. Behav.*, vol. 4, no. 3, pp. 294–307, Mar. 2020.

[7] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

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

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

[10] V. Heidrich-Meisner and C. Igel, "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, 2009, pp. 401–408.

[11] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," *IEEE Trans. Comput. Intell. AI Games*, vol. 9, no. 1, pp. 25–41, Mar. 2017.

[12] H.-G. Beyer, *The Theory of Evolution Strategies*. Berlin, Germany: Springer, 2013.

[13] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017, *arXiv:1703.03864*.

[14] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5027–5038.

[15] S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1194–1206.

[16] G. Liu *et al.*, "Trust region evolution strategies," in *Proc. AAAI Conf. Artif. Intell.*, 2019, vol. 33, no. 1, pp. 4352–4359.

[17] 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.

[18] 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.

[19] A. Nagabandi *et al.*, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] Z. Wang, H.-X. Li, and C. Chen, "Incremental reinforcement learning in continuous spaces via policy relaxation and importance weighting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 1870–1883, Jun. 2020.

[25] Z. Wang, C. Chen, and D. Dong, "Lifelong incremental reinforcement learning with online Bayesian inference," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Feb. 11, 2021, doi: 10.1109/TNNLS.2021.3055499.

[26] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.

[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.*, 2014, pp. 3320–3328.

[28] 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.

[29] 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.

[30] 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.

[31] D. Kulić, 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.

[32] Z. Wang and H.-X. Li, "Incremental spatiotemporal learning for online modeling of distributed parameter systems," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 12, pp. 2612–2622, Dec. 2019.

[33] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[34] D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 949–980, 2014.

[35] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.

[36] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep. 2017, pp. 23–30.

[37] F. Muratore, M. Gienger, and J. Peters, "Assessing transferability from simulation to reality for reinforcement learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 4, pp. 1172–1183, Apr. 2021.

[38] M. Sheckells, G. Garimella, S. Mishra, and M. Kobilarov, "Using data-driven domain randomization to transfer robust control policies to mobile robots," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 3224–3230.

[39] T. Chen, A. Murali, and A. Gupta, "Hardware conditioned policies for multi-robot transfer learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9333–9344.

[40] W. Yu, J. Tan, C. K. Liu, and G. Turk, "Preparing for the unknown: Learning a universal policy with online system identification," in *Proc. Robot., Sci. Syst.*, Cambridge, MA, USA, Jul. 2017.

[41] W. Yu, C. K. Liu, and G. Turk, "Policy transfer with strategy optimization," in *Proc. Int. Conf. Learn. Represent.*, 2019.

[42] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2018, pp. 1–8.

[43] M. Andrychowicz *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[44] B. M. Lake, T. D. Ullman, J. B. Tenebaum, and S. J. Gershman, "Building machines that learn and think like people," *Behav. Brain Sci.*, vol. 40, p. e253, 2017. [Online]. Available: https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/building-machines-that-learn-and-think-like-people/A9535B1D745A0377E16C590E14B94993#

[45] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[46] A. Gajewski, J. Clune, K. O. Stanley, and J. Lehman, "Evolvability ES: Scalable and direct optimization of evolvability," in *Proc. Genet. Evol. Comput. Conf.*, Jul. 2019, pp. 107–115.

[47] R. Houthooft *et al.*, "Evolved policy gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 5400–5409.

[48] X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang, "ES-MAML: Simple Hessian-free meta learning," in *Proc. Int. Conf. Learn. Represent.*, 2020.

[49] K. Kawaguchi, "Deep learning without poor local minima," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 586–594.

[50] P. Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 265–286, Apr. 2007.

[51] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evol. Comput.*, vol. 19, no. 2, pp. 189–223, Jun. 2011.

[52] 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.

[53] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 1038–1044.

[54] A. Tirinzoni, A. Sessa, M. Pirotta, and M. Restelli, "Importance weighted transfer of samples in reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 4936–4945.

[55] M. Plappert *et al.*, "Parameter space noise for exploration," in *Proc. Int. Conf. Learn. Represent.*, 2018.

[56] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent.*, 2016.

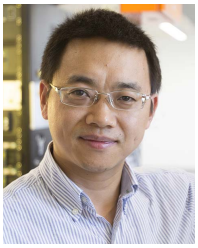
Zhi Wang (Member, IEEE) received the B.E. degree in automation from Nanjing University, Nanjing, China, in 2015, and the Ph.D. degree in machine learning from the Department of Systems Engineering and Engineering Management, City University of Hong Kong, Hong Kong, China, in 2019.

He had a visiting position at the University of New South Wales, Canberra, ACT, Australia. He is currently an Assistant Professor with the Department of Control and Systems Engineering, Nanjing University. His current research interests include reinforcement learning, machine learning, and robotics.

Chunlin Chen (Senior Member, IEEE) 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.

He was with the Department of Chemistry, Princeton University, Princeton, NJ, USA, from September 2012 to September 2013. He had visiting positions at the University of New South Wales, Canberra, ACT, Australia, and the City University of Hong Kong, Hong Kong. He is currently a Professor and the Head of 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 serves as the Chair for the Technical Committee on Quantum Cybernetics, IEEE Systems, Man and Cybernetics Society.



Daoyi Dong (Senior Member, IEEE) received the B.E. degree in automatic control and the Ph.D. degree in engineering from the University of Science and Technology of China, Hefei, China, in 2001 and 2006, respectively.

He was an Alexander von Humboldt Fellow at AKS, University of Duisburg-Essen, Duisburg, Germany. He was with the Institute of Systems Science, Chinese Academy of Sciences, Beijing, China, and Zhejiang University, Hangzhou, China. He had visiting positions at Princeton University, NJ, USA; RIKEN, Wako, Japan; and The University of Hong Kong, Hong Kong. He is currently a Scientia Associate Professor at the University of New South Wales, Canberra, ACT, Australia. His research interests include quantum control and machine learning.

Dr. Dong received the ACA Temasek Young Educator Award by the Asian Control Association. He was a recipient of the International Collaboration Award and the Australian Post-Doctoral Fellowship from the Australian Research Council and a Humboldt Research Fellowship from the Alexander von Humboldt Foundation of Germany. He is a Member-at-Large, Board of Governors, and the Associate Vice President for Conferences and Meetings, IEEE Systems, Man and Cybernetics Society. He served as an Associate Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS from 2015 to 2021. He is also an Associate Editor of the IEEE TRANSACTIONS ON CYBERNETICS and a Technical Editor of the IEEE/ASME TRANSACTIONS ON MECHATRONICS.