Rethinking Value Function Learning for Generalization in Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

We focus on the problem of training RL agents on multiple training environments to improve the generalization performance and sample efficiency. In prior methods, policy and value function are separately optimized with the goal of avoiding interference and obtaining a more accurate value function using a separate network architecture. We identify that the value function is more prone to overfitting training data and an appropriate penalization of the value function is required for better training and test performance in the multi-environment setting. To this end, we introduce Delayed-Critic Policy Gradient (DCPG), which implicitly penalizes the value estimates by training the value function less frequently with more training data compared to the policy. Furthermore, we propose a simple self-supervised task that implicitly learns the forward and inverse dynamics of environments using a single discriminator, which can be jointly optimized with the value function. Compared to the prior methods, our proposed algorithms significantly improve generalization performance and sample efficiency in the Procgen Benchmark.

1 Introduction

In recent years, deep reinforcement learning (RL) has achieved remarkable success in various tasks such as robotic controls and games [24, 15, 28]. To apply RL algorithms to more practical applications such as autonomous vehicles or healthcare systems, they should be robust against the non-stationarity of the real world and capable of generalizing to unseen situations during their deployment. However, current RL algorithms often fail to generalize to unseen environments with the same goal but different contexts even if they achieve high sample efficiency in their training environments [12, 36, 7].

Training RL agents on a finite set of environments and testing them on the full distribution of unseen environments is the standard practice for evaluating generalization in RL [8]. In this framework, several methods have attempted to improve generalization by adopting the regularization techniques that originate from supervised learning or training robust representations via self-supervised learning [7, 16, 27, 23]. However, these methods have mainly focused on developing new auxiliary objectives on top of the existing actor-critic algorithms intended for the conventional single-environment setting such as PPO [30]. Some recent works identify the interference between policy and value optimization arising from the multi-environment setting and propose new RL training schemes to obtain an accurate value function by decoupling the policy and value function training with separate networks [9, 26].

In this work, we argue that obtaining an accurate value function in the multi-environment setting is more challenging than in the single-environment setting and it should be optimized with sufficient regularization. We identify that the value function trained on multiple environments is more prone to memorizing the training data and cannot generalize to unvisited states of the training environments, which acts detrimentally to the training performance. Moreover, we find that common regularization methods that penalize large estimates of the value function, originally developed for preventing the

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

overfitting problem in the single-environment setting, are also beneficial to improving both train and test performance in the multi-environment setting. However, this benefit comes at the cost of early convergence which hinders further performance enhancement.

Based on these observations, we introduce Delayed-Critic Policy Gradient (DCPG), where the value network is implicitly penalized by optimizing less frequently with more data than the policy network. We find that the value function with delayed updates suffers less from the overfitting problem and improves the train and test performance. We also demonstrate that it provides better representations for generalization to the policy network with a shared encoder. Furthermore, we propose a simple self-supervised task that implicitly learns the forward and inverse dynamics of environments using a single discriminator on top of DCPG. Our methods achieve state-of-the-art generalization performance and sample efficiency on the challenging Procgen benchmark [8].

2 Preliminaries

2.1 Generalization in RL

We consider a collection of environments $\mathcal{M}$ formulated as Markov Decision Processes (MDPs). Each environment $m \in \mathcal{M}$ is defined by a tuple $(\mathcal{S}_m, \mathcal{A}, T_m, r_m, \rho_m, \gamma)$, where $\mathcal{S}_m$ is the state space, $\mathcal{A}$ is the action space shared across all environments, $T_c : \mathcal{S}_m \times \mathcal{A} \to \mathcal{P}(\mathcal{S}_m)$ is the transition function, $r_m : \mathcal{S}_m \times \mathcal{A} \to \mathbb{R}$ is the reward function, ρ_m is the initial state distribution, and $\gamma \in (0, 1]$ is the discount factor. All states in $\mathcal{M}$ have equal dimensions and we denote the set of all possible states by $\mathcal{S}$. While the transition and reward functions are defined dependent on an environment m, we assume that they share some common structures. A policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is trained using a fixed number of training environments $\mathcal{M}_{\text{train}} = \{m_i\}_{i=1}^n$. The objective is to find the optimal policy π^* that maximizes the expected return on the unseen test environments $\mathbb{E}_{m,s_t,a_t}[\sum_{t=0}^{\infty} \gamma^t r_m(s_t, a_t)]$, where $m \sim \mathcal{U}(\mathcal{M} \setminus \mathcal{M}_{\text{train}})$, $s_0 \mid m \sim \rho_m$, $a_t \sim \pi(\cdot \mid s_t)$, and $s_{t+1} \mid m \sim T_m(\cdot \mid s_t, a_t)$.

In this paper, we use the Procgen benchmark as a testbed for evaluating generalization in RL [8]. The Procgen benchmark is a suite of 16 games with high diversity comparable to the ALE benchmark [3]. Each game consists of procedurally generated environment instances with varying contexts (*e.g.*, layout, background, and the position and spawn time of entities), which are also called levels. Each level is completely determined by an integer seed. The standard protocol for evaluating generalization on the Procgen benchmark is to train an agent on a fixed set of training levels and evaluate its performance on the unseen test levels sampled from the full distribution of levels [8].

2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is one of the most successful policy gradient algorithms that learns a policy π_θ and value function V_ϕ parameterized by neural networks [30]. PPO first collects trajectories τ using the old policy $\pi_{\theta_{\text{old}}}$ right before update. Then, the policy network π_θ is trained with the sampled trajectories for several epochs (often called sample reuse) to maximize the following clipped surrogate policy objective J_π:

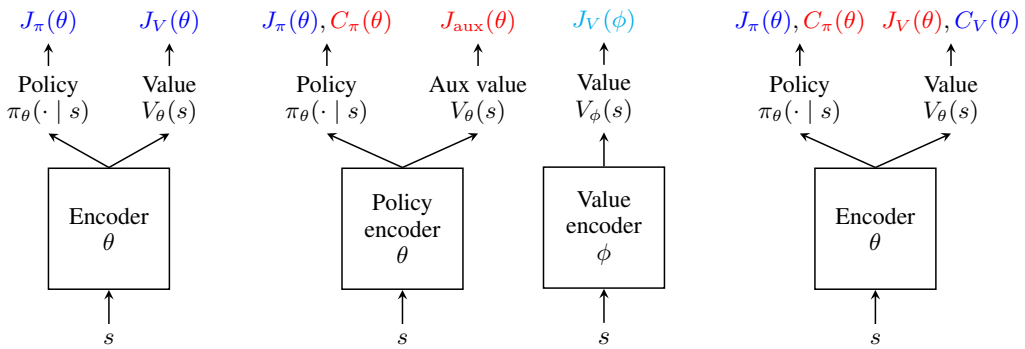
$$J_\pi(\theta) = \mathbb{E}_{s_t,a_t\sim\tau}\left[\min\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right],$$

where $\hat{A}_t$ is an advantage estimate at timestep t. The value network V_ϕ is trained to minimize the following value objective J_V:

$$J_V(\phi) = \mathbb{E}_{s_t\sim\tau}\left[\frac{1}{2}\left(V_\phi(s_t) - \hat{R}_t\right)^2\right],$$

where $\hat{R}_t = \hat{A}_t + V_\phi(s_t)$ is a value function target. It is used to compute the advantage estimates via generalized advantage estimator (GAE) [29].

In practice, the policy and value networks are optimized simultaneously with shared parameters (*i.e.*, $\theta = \phi$), especially in image-based RL [11, 34]. For example, they can be implemented using a shared encoder with separate linear heads as shown in Figure 1a. Sharing the parameters has an advantage in that features learned by each objective can be beneficial to the other. It also reduces the memory cost and accelerates the training time. However, shared network design complicates the optimization as a single network needs to be optimized over multiple objectives whose gradients may have varying scales and directions. This also constrains the policy and value networks to be trained with the same batch of data to the same degree of sample reuse, which severely limits the flexibility of PPO.



(a) PPO network (b) PPG network (c) DCPG network

Figure 1: Network architectures for PPO, PPG, and DCPG. The objectives J_π, J_V, and J_{aux} denote the policy, value, and auxiliary objectives, respectively. The regularizers C_π and C_V denote the policy and value regularizers, respectively. The blue, red, and cyan terms define optimization problem during the policy phase, auxiliary phase, and both, respectively.

2.3 Phasic Policy Gradient

Phasic Policy Gradient (PPG) addresses the problems of sharing parameters and uses separate networks as shown in Figure 1b [9]. It also introduces an auxiliary value head V_θ to distill useful features from the value function into the policy network. PPG alternates between policy and auxiliary phases. During the policy phase, the policy network π_θ and the value network V_ϕ are trained on newly-collected trajectories to optimize the policy objective J_π and the value objective J_V, respectively. All states s_t and value targets $\hat{R}_t$ in the trajectories are stored in a buffer $\mathcal{B}$. During the auxiliary phase, the auxiliary value head V_θ and the policy π_θ are jointly trained on all data in the buffer to optimize the following auxiliary objective J_{aux} and policy regularizer C_π:

$$J_{\text{aux}}(\theta) = \mathbb{E}_{s_t \sim \mathcal{B}}\left[\frac{1}{2}\left(V_\theta(s_t) - \hat{R}_t\right)^2\right], \; C_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{B}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \parallel \pi_\theta(\cdot \mid s_t))\right].$$

That is, the value function is distilled into the policy network while preserving the original policy $\pi_{\theta_{\text{old}}}$. Additionally, the value network V_ϕ is further trained on all data in the buffer to optimize the value objective J_V. This stems from the idea that the distillation of a better-trained value function with a separate network and the additional training for a more accurate value function can be beneficial to improving generalization performance and sample efficiency, as claimed in Cobbe et al. [9].

3 Motivation

3.1 Overfitting in Value Function on Multiple Training Environments

We begin by investigating the difficulty of training an accurate value network on multiple training environments. Obviously, a more accurate value network on given training environments leads to better training performance and sample efficiency [31]. However, even in a simple setting where an agent is trained on a single environment, it has been shown that value networks are prone to overfitting in the sense that it does not extrapolate well to unseen states of the training environment [19, 10]. Increasing the number of training environments could further exacerbate the problem. Intuitively, the value network might rely more on memorization when fed with fewer training samples per environment given the same number of environment steps.

To corroborate this claim, we measure the stiffness (cosine similarity) of the value loss gradients between states (s, s') following the practice in Fort et al. [13], Bengio et al. [4]. It is defined by

$$\rho(s, s') = \frac{\nabla_\phi J_V(\phi; s)^\intercal \nabla_\phi J_V(\phi; s')}{\|\nabla_\phi J_V(\phi; s)\|_2 \|\nabla_\phi J_V(\phi; s')\|_2}.$$

Note that low stiffness indicates that updating the parameters towards minimizing the value loss for one state has a negative impact on minimizing the value loss for other states. It has been shown that a value network with lower stiffness is more likely to memorize the train data [4]. Specifically, we train PPG agents on the Procgen games while varying the number of training levels from 1 to 200 and compute the average stiffness across all state pairs in a mini-batch of size 2^{14} throughout training. The detailed experimental settings can be found in Appendix A.

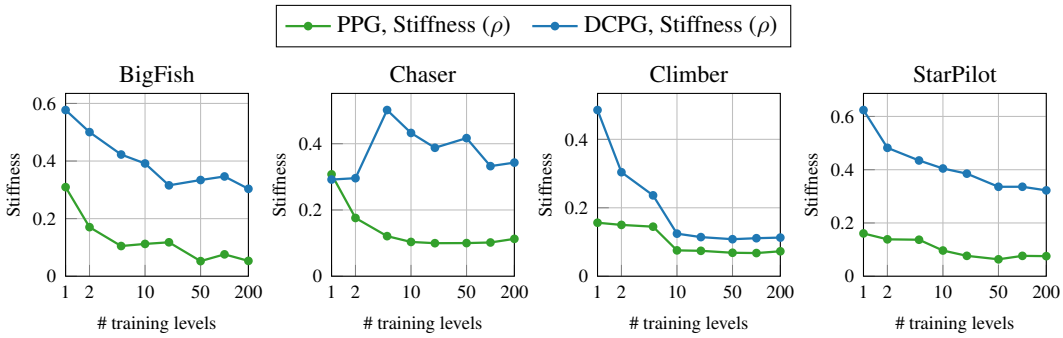


Figure 2: Plots of stiffness of value networks for PPG and DCPG agents while varying the number of training levels on 4 Procgen games.

The green lines in Figure 2 show that the stiffness of the value network decreases as the number of training environments increases as expected. This indicates that the value function trained on multiple environments is more likely to memorize the training data and cannot accurately predict the values for unseen states from the training environments. This overfitting problem brings us to train the value network with sufficient regularization.

3.2 Training Value Function with Explicit Regularization

Now, we examine the effectiveness of value function regularization in the multi-environment setting. We introduce two common approaches taken from the single-environment setting to prevent the overfitting in value functions. First, discount regularization (DR) is a method that trains a value function with a lower discount factor γ' [25]. Second, activation regularization (AR) is a technique that trains a value network with L_2 penalty on the output of the value function [2]:

$$J_V^{\text{reg}}(\phi) = \mathbb{E}_{s_t \sim \tau}\left[\frac{1}{2}\left(V_\phi(s_t) - \hat{R}_t\right)^2 + \frac{\alpha}{2}\left(V_\phi(s_t)\right)^2\right],$$

where $\alpha > 0$ is the regularization coefficient.

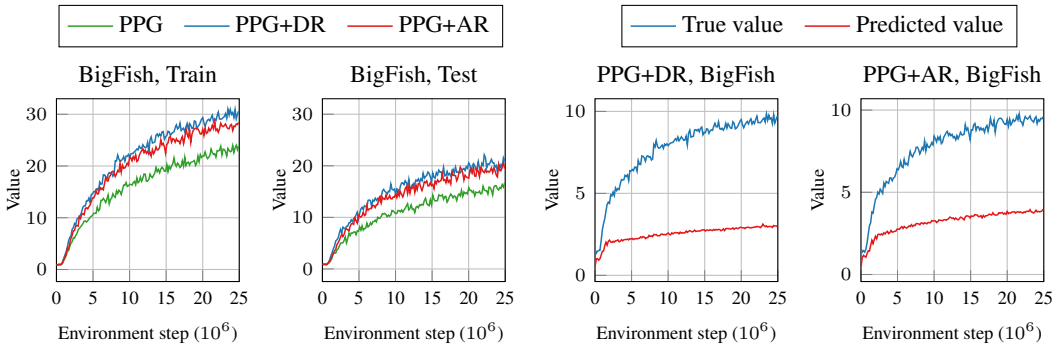
To validate the effectiveness of the regularization in the multi-environment setting, we train PPG agents with each of these two methods using 200 training levels on the Procgen games. Specifically, we reduce the discount factor from $\gamma = 0.999$ to $\gamma' = 0.995$ for PPG+DR and use $\alpha = 0.05$ for PPG+AR. We measure the average train and test returns to evaluate the training performance and its transferability to unseen test environments.

As shown in Figure 3a, value function regularization improves both the train and test performance of PPG to some extent. This implies that explicitly suppressing the value function also helps to reduce overfitting in the multi-environment setting. We also find that these regularization methods achieve better train and test performance in all Procgen games on average. More details on the experimental setting can be found in Appendix B.

Despite its effectiveness, explicit value function regularization can lead to a suboptimal solution when the environment step increases. Figure 3b shows the average of the true and predicted values measured at the initial states of training environments for PPG+DR and PPG+AR. We find that the value network trained with regularization reaches a plateau too quickly, which implies that excessive regularization later hinders the value network from learning the correct value function. This motivates us to find a more flexible regularization that boosts training performance while allowing the value network to converge to true values.

4 Delayed-Critic Policy Gradient

In this section, we propose a new policy gradient algorithm that builds upon PPG, namely *Delayed-Critic Policy Gradient* (DCPG), which deals with the overfitting problem of the value function in a simple and flexible fashion. The key idea is that the value network should be optimized less frequently than the policy network to suppress the value estimates and reduce overfitting. Also, it should be trained with more data to prevent overfitting to small, recent training data, as mentioned in [21].



(a) Average train and test returns.

(b) True and predicted values.

Figure 3: (a) Average train and test returns for PPG, PPG+DR ($\gamma' = 0.995$), and PPG+AR ($\alpha = 0.05$) on BigFish. (b) True and predicted values for initial states of training environments for PPG+DR and PPG+AR on BigFish. The mean is computed over 10 runs.

Algorithm 1 Dynamics-aware Delayed-Critic Policy Gradient (DDCPG)

Require: Policy network π_θ, value network V_θ, dynamics head f_θ

for phase = $1, 2, \ldots$ **do**
  Initialize empty buffer $\mathcal{B}$
  for iter = $1, 2, \ldots, N_\pi$ **do** ▷ Policy phase
    Sample trajectories τ using π_θ and compute value function target $\hat{R}_t$ for each state $s_t \in \tau$
    Add $(s_t, a_t, \hat{R}_t)$ to $\mathcal{B}$
    for epoch = $1, 2, \ldots, E_\pi$ **do**
      Optimize policy objective $J_\pi(\theta)$ and value regularizer $C_V(\theta)$ with τ
    end for
  end for
  for iter = $1, 2, \ldots, E_{\text{aux}}$ **do** ▷ Auxiliary phase
    Optimize value objective $J_V(\theta)$, dynamics objective $J_f(\theta)$, and policy regularizer $C_\pi(\theta)$ with $\mathcal{B}$
  end for
end for

4.1 Algorithm

DCPG proceeds by alternating the policy and auxiliary phases similar to PPG, but it uses a *shared encoder* and does not require an auxiliary value head as shown in Figure 1c. During the policy phase, new trajectories are collected and the policy network π_θ is trained on the trajectories to optimize the policy objective J_π. However, the value network V_θ is trained on the trajectories to optimize the value regularizer C_V to preserve the outputs of the original value function $V_{\theta_{\text{old}}}$, which is given by

$$C_V(\theta) = \mathbb{E}_{s_t \sim \tau}\left[\frac{1}{2}\left(V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t)\right)^2\right],$$

in the sense that updating the value network frequently can further increase approximation errors for unseen states as observed in Section 3.1. Finally, all states and value targets in the trajectories are stored in a buffer $\mathcal{B}$.

During the auxiliary phase, the value network V_θ and policy network π_θ are trained with a larger number of data in the buffer to optimize the value objective J_V and the policy regularizer C_π. Note that since the value network has the same encoder as the policy network, optimizing the value objective also plays the role of representation learning for the policy network. Thus, we do not require an additional auxiliary head for feature distillation, unlike PPG. The overall procedure of DCPG can be found in Algorithm 1.

4.2 Value Analysis of Delayed Value Update

The delayed value update can operate as an implicit penalization of the value function. Since the policy improvement is not reflected in the value network immediately, it encourages smaller value estimates than the true values. To validate this claim, we train DCPG agents using 200 training levels

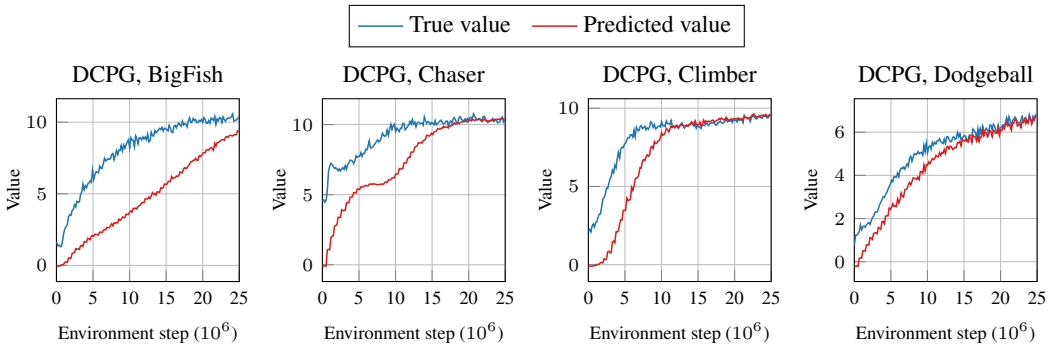


Figure 4: True and predicted values for initial states of train environments for DCPG on 4 Procgen games. The mean is computed over 10 runs.

on the Procgen games and measure the true and predicted values for the initial states of the training levels. Figure 4 shows the outputs of the value network are kept lower than the true values at the early stage of training but recover the true values as training progresses. More results can be found in Appendix C.

We also evaluate the effectiveness of the delayed value update in mitigating the overfitting problem by measuring the stiffness of the value function while varying the number of training levels, The blue lines in Figure 2 show that DCPG has higher stiffness compared to PPG. It implies that our value network is more robust against overfitting as addressed in Section 3.1. We observe that the delayed update of the value network in DCPG serves as a good regularizer for the value network. More results can be found in Appendix A.

4.3 Learning Forward and Inverse Dynamics with Single Discriminator

To further improve generalization performance and sample efficiency, we utilize the buffer $\mathcal{B}$ to learn an auxiliary task. Suppose the buffer contains a transition $(s_t, a_t, s_{t+1}) \in \mathcal{B}$. Since the transition functions are assumed to have a shared structure, it is natural to train a neural network that models forward dynamics in feature spaces. Concretely, given an encoder e_θ, a dynamics head f_θ takes the current state embedding of $e_\theta(s_t)$ and the current action a_t as inputs and predicts the next state embedding of $e_\theta(s_{t+1})$. However, it is well known that a forward dynamics model using a neural network tends to overfit on a small number of train data and make poor predictions on unseen states [6]. To address this, we consider learning the forward dynamics implicitly by training a discriminator that distinguishes whether the next state is valid, given the current state and action:

$$f_\theta(e_\theta(s_t), a_t, e_\theta(s_{t+1})) = 1, \;\; f_\theta(e_\theta(s_t), a_t, e_\theta(\hat{s}_{t+1})) = 0 \;\text{ for }\; \hat{s}_{t+1} \in \mathcal{B} \setminus \{s_{t+1}\}.$$

Such a discriminator might discard information about the action to distinguish whether a transition is valid. For example, the discriminator can predict the valid transition by using only the proximity of the current and next states. To avoid this, we train the discriminator to jointly distinguish whether the current action is valid, given the current and next states (*i.e.*, inverse dynamics):

$$f_\theta(e_\theta(s_t), a_t, e_\theta(s_{t+1})) = 1, \;\; f_\theta(e_\theta(s_t), \hat{a}_t, e_\theta(s_{t+1})) = 0 \;\text{ for }\; \hat{a}_t \in \mathcal{A} \setminus \{a_t\}.$$

Finally, we optimize the following objective during the auxiliary phase:

$$J_f(\theta) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{B}} \big[\log\left(f_\theta\left(e_\theta(s_t), a_t, e_\theta(s_{t+1})\right)\right) + \log\left(1 - f_\theta\left(e_\theta(s_t), a_t, e_\theta(\hat{s}_{t+1})\right)\right) + \eta \log\left(1 - f_\theta\left(e_\theta(s_t), \hat{a}_t, e_\theta(s_{t+1})\right)\right) \big],$$

where $\hat{s}_{t+1} \sim \mathcal{U}(\mathcal{B} \setminus \{s_{t+1}\})$, $\hat{a}_t \sim \mathcal{U}(\mathcal{A} \setminus \{a_t\})$, and $\eta > 0$ is the coefficient for the inverse dynamics. We name the resulting algorithm *Dynamics-aware Delayed-Critic Policy Gradient* (DDCPG). The full algorithm is described in Algorithm 1 (difference with DCPG is marked in blue).

5 Experiments

5.1 Experimental Settings

We evaluate the generalization performance and sample efficiency of our methods on all 16 games in the Procgen benchmark. We use the "easy" difficulty mode, which most prior works have focused on.

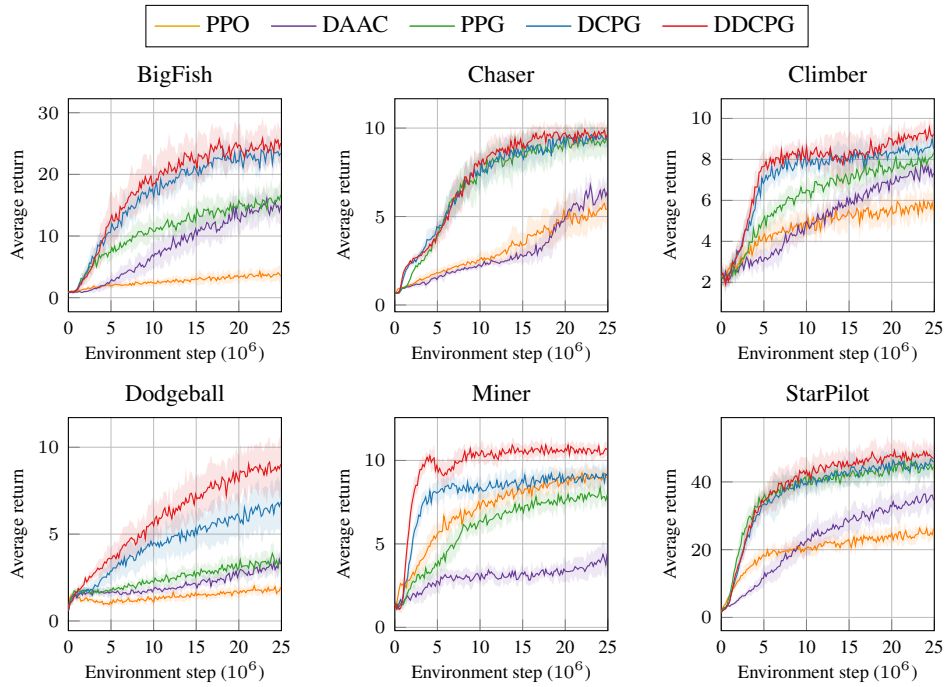


Figure 5: Test performance curves of each method on 6 Procgen games. Each agent is trained on 200 training levels for 25M environment steps and evaluated on 100 unseen test levels. The mean and standard deviation are computed over 10 runs.

We train agents on 200 training levels generated by seeds ranging from 1 to 200 for 25M environment steps and test them on held-out test levels, following the practice in Cobbe et al. [8]. For evaluation, we measure the average return of 100 train and test episodes and report its mean and standard deviation over 10 runs with different initialization.

We compare our methods with PPO and 4 other baselines that use PPO as the backbone algorithm: UCB-DrAC, PLR, PPG, and DAAC [27, 18, 9, 26]. UCB-DrAC is a data augmentation method for training policy and value networks robust against image transformations. PLR is a sampling algorithm for procedurally generated environments that selects the next training level based on its future learning potential. PPG is the previous state-of-the-art method on the Procgen benchmark. DAAC builds upon PPG and distills advantage function into the policy network. For each method, we use the same set of hyperparameters across all games, following the standard protocol in the ALE and Procgen benchmarks [24, 8]. For our methods, we use the same hyperparameter setting as PPG for a fair comparison. To compare the performance of each method with a single score, we calculate the PPO-normalized score averaged over all Procgen games, which is computed by dividing the average return of each method by PPO, following the practice in Raileanu et al. [27]. For the implementation details and hyperparameters, please refer to Appendix D.

5.2 Generalization Performance on Procgen Benchmark

Table 1 shows the average test returns of each method on all 16 Procgen games after training. DCPG significantly outperforms all the baselines, yielding a 24%p improvement on average relative to the previous state-of-the-art method. Notably, the delayed value update without any auxiliary task leads to better test performance in 8 of 16 games. Furthermore, DDCPG improves upon DCPG with an 18%p increase in the normalized score. Adding dynamics learning to DCPG can boost the test performance in 11 of 16 games, showing its effectiveness and consistency across various types of games. We also provide the average training returns on all Procgen games in Appendix E, showing that DCPG and DDCPG also have better sample efficiency in the multi-environment setting.

Figure 5 shows the test performance curves of each method on 6 Procgen games throughout training. We find that our methods not only achieve superior final test performance but also exhibit faster convergence compared to the baselines. It implies that our methods can acquire generalization ability

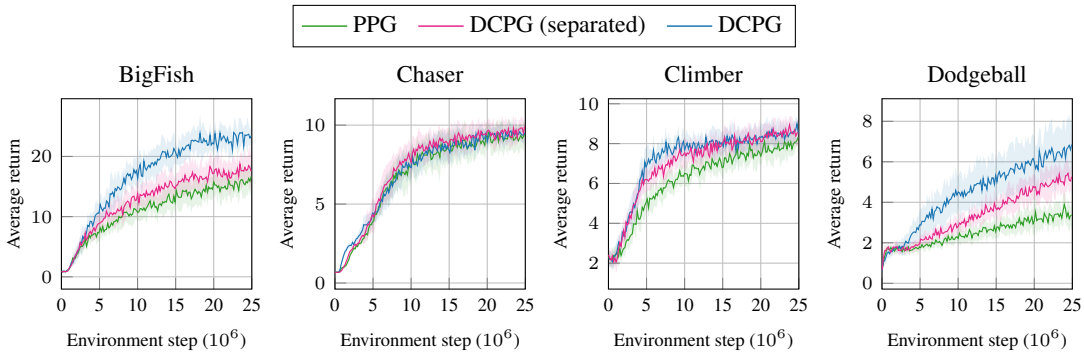


Figure 6: Test performance curves of PPG, DCPG, and separated DCPG on 4 procgen games. Each agent is trained on 200 training levels for 25M environment steps and evaluated on 100 unseen test levels. The mean and standard deviation are computed over 10 runs.

Table 1: Average test returns of each method on all 16 Procgen games. Each agent is trained on 200 training levels for 25M environment steps and evaluated on 100 unseen test levels. The mean and standard deviations are computed over 10 runs.

Environment	PPO	UCB-DrAC	PLR	DAAC[1]	PPG	DCPG	DDCPG
BigFish	3.4 ± 1.0	6.6 ± 2.5	10.8 ± 2.5	15.3 ± 3.0	16.4 ± 2.0	$\mathbf{23.1 \pm 2.3}$	$\mathbf{25.9 \pm 2.3}$
BossFight	7.4 ± 0.7	7.2 ± 1.0	8.8 ± 0.7	9.5 ± 0.8	10.4 ± 0.6	10.2 ± 0.4	$\mathbf{10.6 \pm 0.5}$
CaveFlyer	5.2 ± 0.6	4.4 ± 0.8	6.3 ± 0.7	5.1 ± 0.6	$\mathbf{7.7 \pm 0.7}$	6.3 ± 0.4	6.0 ± 0.3
Chaser	5.5 ± 0.8	6.7 ± 0.5	7.5 ± 0.8	6.3 ± 0.8	8.9 ± 0.9	$\mathbf{9.7 \pm 0.4}$	$\mathbf{9.9 \pm 0.6}$
Climber	5.5 ± 0.6	6.2 ± 0.6	6.4 ± 0.5	7.4 ± 0.5	8.1 ± 0.4	$\mathbf{8.5 \pm 0.7}$	$\mathbf{9.3 \pm 0.7}$
CoinRun	8.8 ± 0.3	8.6 ± 0.4	8.9 ± 0.3	$\mathbf{9.3 \pm 0.2}$	8.9 ± 0.3	8.5 ± 0.4	8.4 ± 0.3
Dodgeball	1.9 ± 0.3	4.5 ± 1.4	2.2 ± 0.4	3.2 ± 0.5	3.6 ± 0.7	$\mathbf{6.7 \pm 0.4}$	$\mathbf{9.0 \pm 1.6}$
FruitBot	27.5 ± 1.5	26.9 ± 1.7	27.7 ± 1.3	28.5 ± 1.4	$\mathbf{29.1 \pm 1.0}$	29.0 ± 1.2	28.8 ± 0.6
Heist	2.6 ± 0.6	3.4 ± 1.0	3.1 ± 0.6	3.0 ± 0.5	2.8 ± 0.5	3.3 ± 0.5	$\mathbf{4.4 \pm 0.8}$
Jumper	5.7 ± 0.4	5.6 ± 0.4	5.9 ± 0.4	5.8 ± 0.4	6.0 ± 0.4	$\mathbf{6.2 \pm 0.5}$	$\mathbf{6.4 \pm 0.4}$
Leaper	5.6 ± 1.5	3.6 ± 0.7	7.3 ± 0.2	$\mathbf{8.2 \pm 1.1}$	7.2 ± 2.4	6.9 ± 1.7	6.8 ± 1.5
Maze	5.3 ± 0.7	$\mathbf{5.7 \pm 1.1}$	5.6 ± 0.5	5.6 ± 0.6	5.3 ± 0.4	$\mathbf{5.7 \pm 0.7}$	$\mathbf{6.5 \pm 0.6}$
Miner	9.2 ± 0.6	10.1 ± 0.7	9.4 ± 0.7	4.4 ± 1.1	7.6 ± 0.7	9.1 ± 0.8	$\mathbf{10.6 \pm 0.5}$
Ninja	5.7 ± 0.5	5.7 ± 0.5	$\mathbf{7.1 \pm 0.5}$	6.7 ± 0.7	6.7 ± 0.4	6.3 ± 0.6	6.6 ± 0.5
Plunder	5.0 ± 0.5	8.1 ± 1.5	8.7 ± 1.3	5.6 ± 0.6	$\mathbf{13.7 \pm 1.7}$	$\mathbf{13.7 \pm 1.3}$	12.9 ± 2.5
StarPilot	25.8 ± 2.0	27.0 ± 3.2	24.9 ± 3.6	35.9 ± 3.3	44.8 ± 2.6	$\mathbf{46.1 \pm 2.1}$	$\mathbf{46.6 \pm 4.1}$
Norm score (%)	100.0 ± 3.1	120.7 ± 10.7	130.0 ± 6.0	136.7 ± 7.6	160.3 ± 6.3	$\mathbf{184.5 \pm 5.2}$	$\mathbf{202.2 \pm 10.2}$

with a fewer number of interactions between environments. For example, our methods can reach the final performance of PPG using only 20% of total environment steps in BigFish and Climber and 10% of total environment steps in Dodgeball and Miner. We provide the train and test performance curves on all Procgen games in Appendix E.

5.3 Ablation Study for Delayed Value Update

The delayed value update not only alleviates the overfitting problem but also provides better representations to the policy network that can generalize to unseen test environments. To validate this, we train DCPG agents with an additional value network V_ϕ using a separate encoder (separated DCPG). The separate value network V_ϕ is trained in the same way as PPG without delay and is only used for computing advantages for policy optimization. The original value network V_θ is only used for learning representations for the policy network with delayed updates. Figure 6 shows that separate DCPG achieves better test performance than PPG, which uses an aggressively optimized value function to learn representations for the policy network. This implies that the delayed value update itself produces more generalizable representations than the value function without any regularization. For the detailed experimental settings and results, please refer to Appendix F.

5.4 Ablation Study for Dynamics Learning

We conduct an ablation study to show the effectiveness of training forward and inverse dynamics using a single discriminator. We compare the test performance of DDCPG to DCPG with forward dynamics learning (DCPG+F), inverse dynamics learning (DCPG+I), and forward and inverse

[1]Results reported in the original paper use a different hyperparameter setting for each game.

dynamics learning using separate discriminators (DCPG+FI). Table 2 shows that inverse dynamics provides additional performance gain only when jointly trained with forward dynamics using a single discriminator. We provide the detailed experimental settings and results in Appendix F.

Table 2: PPO-normalized score of each method with dynamics learning. Each agent is trained on 200 training levels for 25M environment steps and evaluated on 100 unseen test levels. The mean and standard deviations are computed over 10 runs.

	DCPG	DCPG+F	DCPG+I	DCPG+FI	DDCPG
Norm score (%)	184.5 ± 5.2	195.9 ± 7.7	185.5 ± 6.5	194.6 ± 6.2	$\mathbf{202.2 \pm 10.2}$

6 Related Works

Generalization in deep RL One major approach for improving generalization in deep RL is to apply regularization techniques originally developed for supervised learning to prevent overfitting, such as batch normalization [17]. It has been shown that batch normalization and L_2 weight regularization provide considerable benefits to generalization [12, 7]. SNI proposes a regularization technique that selectively injects noises into policy parameters and learns compressed state representations using VIB [16]. Several works employ data augmentation techniques that are commonly used in computer vision to tackle the generalization problem [20, 33]. UCB-DrAC proposes methods that automatically find an effective augmentation and introduces new regularization terms to learn robust state representations under various augmentations [27].

Another line of work uses self-supervised learning techniques to learn invariant representations that can generalize to unseen environments. DIM learns a state representation that can predict the representations of successive timesteps using self-supervised learning [22]. DBC trains encoders such that the distances in latent space are equivalent to the bisimulation metrics to discard irrelevant information for policy [35]. PSE learns state representations that capture long-term behavior proximity between states using contrastive learning [1]. CTRL proposes a new clustering algorithm that groups representations of behaviorally similar trajectories [23].

Most closely related to our work is PPG, which suggests a better training regime to improve both generalization and sample efficiency in multi-environment settings [9]. It trains the policy and value function on separate networks with different levels of sample reuse to avoid interference and obtain a better-trained value function. DAAC also uses decoupled policy and value networks to train a more accurate value function [26]. However, it distills the advantage function into the policy, which is less prone to overfitting to environment-specific features than the value function. We focus on regularizing the value function while obtaining better representation, which distinguishes our work from above approaches.

Learning value function with regularization Discount regularization is one common regularization method to prevent overfitting in value function, especially when training data is limited [5]. A lower discount factor can lead to better performance in approximated dynamic programming when the approximation error is large [25]. It can also be helpful for mitigating overfitting when dealing with limited data in POMDPs [14]. Activation regularization penalizes the norm of the value function and has regularization effect similar to that of discount regularization [2]. It has been shown that activation regularization is equivalent to the discount regularization in batch TD learning and also effective in online RL [2].

7 Conclusion

We have investigated the difficulty of learning an accurate value function in a multi-environment setting and shown that suppressing a value function with appropriate regularization can be beneficial to improving the train and test performance. Based on this observation, we propose Delayed-Critic Policy Gradient (DCPG), where the value network is implicitly regularized to have lower predicted values at the early stage of training with delayed updates compared to the policy network. We find that the delayed value update not only prevents the memorization of training data in the multi-environment setting but also produces more generalizable representations that can extend to unseen test environments. Furthermore, we propose a simple self-supervised task that implicitly learns both forward and inverse dynamics using a single discriminator, which can be easily combined with DCPG. Our methods exhibits state-of-the-art performance in the challenging Procgen benchmark.

References

[1] Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. In *ICLR*, 2021.

[2] Ron Amit, Ron Meir, and Kamil Ciosek. Discount factor as a regularizer in reinforcement learning. In *ICML*, 2020.

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. In *ICML*, 2020.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-dynamic programming.* Athena Scientific, Belmont, MA, 1996.

[6] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

[7] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *ICML*, 2019.

[8] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, 2020.

[9] Karl W Cobbe, Jacob Hilton, Oleg Klimov, and John Schulman. Phasic policy gradient. In *ICML*, 2021.

[10] Will Dabney, André Barreto, Mark Rowland, Robert Dadashi, John Quan, Marc G Bellemare, and David Silver. The value-improvement path: Towards better representations for reinforcement learning. In *AAAI*, 2021.

[11] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[12] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[13] Stanislav Fort, Paweł Krzysztof Nowak, Stanislaw Jastrzebski, and Srini Narayanan. Stiffness: A new perspective on generalization in neural networks. *arXiv preprint arXiv:1901.09491*, 2019.

[14] Vincent François-Lavet, Guillaume Rabusseau, Joelle Pineau, Damien Ernst, and Raphael Fonteneau. On overfitting and asymptotic bias in batch reinforcement learning with partial observability. *Journal of Artificial Intelligence Research*, 65:1–30, 2019.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[16] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *NeurIPS*, 2019.

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[18] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *ICML*, 2021.

[19] J Zico Kolter and Andrew Y Ng. Regularization and feature selection in least-squares temporal difference learning. In *ICML*, 2009.

[20] Misha Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *NeurIPS*, 2020.

[21] Nir Levine, Tom Zahavy, Daniel J Mankowitz, Aviv Tamar, and Shie Mannor. Shallow updates for deep reinforcement learning. In *NeurIPS*, 2017.

[22] Bogdan Mazoure, Remi Tachet des Combes, Thang Long Doan, Philip Bachman, and R Devon Hjelm. Deep reinforcement and infomax learning. In *NeurIPS*, 2020.

[23] Bogdan Mazoure, Ahmed M Ahmed, R Devon Hjelm, Andrey Kolobov, and Patrick MacAlpine. Cross-trajectory representation learning for zero-shot generalization in RL. In *ICLR*, 2022.

[24] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] Marek Petrik and Bruno Scherrer. Biasing approximate dynamic programming with a lower discount factor. In *NeurIPS*, 2008.

[26] Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *ICML*, 2021.

[27] Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. In *NeurIPS*, 2021.

[28] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[29] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

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

[31] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, 1999.

[32] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, 2012.

[33] Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *ICLR*, 2021.

[34] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *AAAI*, 2021.

[35] Amy Zhang, Rowan Thomas McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *ICLR*, 2021.

[36] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We clearly reflect the main claims in Section 3 and Section 4.
   (b) Did you describe the limitations of your work? [N/A]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We have read the ethics review guidelines and checked our paper has no conflict.

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We provide the code for reproducing the main results in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We provide the implementation details and hyperparameters of all baselines and our methods in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We conduct experiments over 10 training seeds and report mean and standard deviation.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We describe the computational resource in the supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite the benchmark used for our experiments.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]