Automatic Data Augmentation for Generalization in Reinforcement Learning

Roberta Raileanu 1  Max Goldstein 1  Denis Yarats 1, 2  Ilya Kostrikov 1  Rob Fergus 1

Abstract
Deep reinforcement learning (RL) agents often fail to generalize to unseen scenarios, even when they are trained on many instances of semantically similar environments. Data augmentation has recently been shown to improve the sample efficiency and generalization of RL agents. However, different tasks tend to benefit from different kinds of data augmentation. In this paper, we compare three approaches for automatically finding an appropriate augmentation. These are combined with two novel regularization terms for the policy and value function, required to make the use of data augmentation theoretically sound for certain actor-critic algorithms. We evaluate our approach on the Procgen benchmark which consists of 16 procedurally-generated environments and show that it improves test performance by $\sim 40\%$.

1. Introduction
Generalization remains one of the key challenges of deep reinforcement learning (RL). A series of recent studies show that RL agents fail to generalize to new environments (Farebrother et al., 2018; Gamrian & Goldberg, 2019; Zhang et al., 2018; Cobbe et al., 2018; 2019; Song et al., 2020). This indicates that current RL methods memorize specific trajectories rather than learning transferable skills. To improve generalization in RL, several strategies have been proposed, such as the use of regularization (Farebrother et al., 2018; Zhang et al., 2018; Cobbe et al., 2018; Igl et al., 2019), data augmentation (Cobbe et al., 2018; Lee et al., 2020; Kostrikov et al., 2020; Laskin et al., 2020), or representation learning (Igl et al., 2019; Lee et al., 2020).

In this paper, we show that a naive application of data augmentation can lead to both theoretical and practical problems with standard RL algorithms. As a solution, we propose Data-regularized Actor-Critic or DrAC, a new algorithm that enables the use of data augmentation with actor-critic algorithms in a theoretically sound way. Specifically, we introduce two regularization terms which constrain the agent’s policy and value function to be invariant to various state transformations. Empirically, this method allows the agent to learn useful behaviors in settings in which a naive use of data augmentation completely fails or converges to a sub-optimal policy.

The current use of data augmentation in RL relies on expert knowledge to pick an appropriate augmentation (Lee et al., 2020; Kostrikov et al., 2020) or separately evaluates a large number of transformations to find the best one (Cobbe et al., 2018; Laskin et al., 2020). In this paper, we propose three methods for automatically finding a good augmentation for a given task. The first two automatically find an effective augmentation from a fixed set, using either the upper confidence bound (UCB, Auer (2002)) algorithm (UCB-DrAC), or meta-learning (Wang et al., 2016) (RL2-DrAC). The third method directly meta-learns the weights of a convolutional network, without access to predefined transformations (Meta-DrAC). Figure 1 gives an overview of UCB-DrAC.

We evaluate our method on the Procgen generalization benchmark (Cobbe et al., 2019), which consists of 16 procedurally-generated environments with visual observations. Our results show that UCB-DrAC is the most effective for automatically finding an augmentation, and is comparable or better than using DrAC with the best augmentation from a given set. UCB-DrAC also outperforms baselines specifically designed to improve generalization in RL (Igl et al., 2019; Lee et al., 2020; Laskin et al., 2020) on both train and test.

2. Proximal Policy Optimization
Proximal Policy Optimization (PPO, Schulman et al. (2017)) is an actor-critic algorithm that learns a policy $\pi_\theta$ and a value function $V_\rho$ with the goal of finding an optimal policy for a given MDP. PPO alternates between sampling data through interaction with the environment and maximizing a clipped surrogate objective function. One component of the PPO objective is the policy gradient term, which is
estimated using importance sampling:

\[
\sum_{a \in A} \pi_\theta(a|s) \hat{A}_{\theta_{old}}(s, a) = \mathbb{E}_{a \sim \pi_{\theta_{old}}} \left[ \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\theta_{old}}(s, a) \right],
\]

where \( \hat{A}(\cdot) \) is an estimate of the advantage function, \( \pi_{\theta_{old}} \) is the behavior policy used to collect trajectories, and \( \pi_\theta \) is the policy we want to optimize.

To ensure that the policy and value functions are invariant to these transformation of the input state, we propose an additional loss term for regularizing the policy,

\[
G_\pi = KL[\pi_\theta(a|f(s, \nu)) \mid \pi(a|s)],
\]

as well as an extra loss term for regularizing the value function,

\[
G_V = (V_\theta(f(s, \nu)) - V(s))^2.
\]

Thus, our data-regularized actor-critic method, or DrAC, maximizes the following objective:

\[
J_{DrAC} = J_{PPO} - \alpha_r(G_\pi + G_V)
\]

where \( \alpha_r \) is the weight of the regularization term.

The use of \( G_\pi \) and \( G_V \) ensures that the agent’s policy and value function are invariant to the transformations induced by various augmentations. Particular transformations can be used to impose certain inductive biases relevant for the task (e.g., invariance with respect to colors or textures). In addition, \( G_\pi \) and \( G_V \) can be added to the objective of any actor-critic algorithm with a discrete stochastic policy (e.g., A3C (Mnih et al., 2013), TRPO (Schulman et al., 2015), SAC (Haarnoja et al., 2018), IMPALA (Espeholt et al., 2018)) without any other changes.

Note that when using DrAC, as opposed to the method proposed by Laskin et al. (2020), we still use the correct importance sampling estimate of the left hand side objective in equation (1). This is because the transformed observations \( f(s) \) are only used to compute the regularization losses \( G_\pi \) and \( G_V \), and thus are not used for the main PPO objective. Hence, DrAC benefits from the regularizing effect of using data augmentation, while mitigating the adverse effects it has on the RL objective.

### 3.3. Automatic Data Augmentation

Since different tasks benefit from different kinds of data augmentation, we would like to design a method that can automatically find an effective augmentation for any given task. In this section, we describe three approaches for doing so.
this. In all of them, the augmentation learner is trained at the same time as the agent learns to solve the task using DrAC. Hence, the distribution of rewards varies significantly as the agent improves, making the problem highly nonstationary.

**Upper Confidence Bound.** The problem of selecting a data augmentation from a given set can be formulated as a multi-armed bandit problem, where the action space is the set of available transformations $\mathcal{F} = \{f_1, \ldots, f_n\}$. A popular algorithm for multi-armed bandit problems is the Upper Confidence Bound or UCB (Auer, 2002), which selects actions according to the following policy:

$$f_t = \underset{f \in \mathcal{F}}{\text{argmax}} \left[ Q(f) + c \sqrt{\frac{\log(t)}{N(f)}} \right]$$

where $N(f)$ is the number of times transformation $f$ has been selected before time step $t$ and $c$ is UCB’s exploration coefficient. We use a modified version of UCB to select augmentations dynamically as the agent learns to solve the task. Before the $t$-th DrAC update, we use equation (6) to select an augmentation $f$. Then, we use equation (5) to update the agent’s policy and value function. We also update the counter: $N(f) = N(f) + 1$. Next, we collect rollouts with the new policy and update the $Q$-function:

$$Q(f) = \frac{1}{K} \sum_{i=t-K}^{t} R(f_i = f),$$

which is computed as a sliding window average of the past $K$ mean returns obtained by the agent after being updated using augmentation $f$. We refer to this algorithm as UCB-DrAC. Note that UCB-DrAC’s estimate of $Q(f)$ differs from that of a typical UCB algorithm which uses rewards from the entire history. However, the choice of estimating $Q(f)$ using only more recent rewards is crucial due to the nonstationarity of the problem.

**Meta-Learning the Selection of an Augmentation.** Alternatively, the problem of selecting a data augmentation from a given set can be formulated as a meta-learning problem. Here, we consider a meta-learner like the one proposed by (Wang et al., 2016), which is an actor-critic architecture parameterized by an LSTM (Hochreiter & Schmidhuber, 1997) that takes as inputs the previous reward and action (i.e. augmentation). Before each DrAC update, the meta-learner selects an augmentation, which is then used to update the agent using equation (5). Then, we collect rollouts using the new policy and update the meta-learner using the mean return of these trajectories. We refer to this approach as RL2-DrAC.

**Meta-Learning the Weights of the Augmentation.** Another approach for automatically finding an appropriate augmentation is to directly learn the weights of a certain transformation rather than selecting an augmentation from a given set. In this work, we focus on meta-learning the weights of a convolutional network which can be applied to the observations to obtain a perturbed image (with the same dimensions as the original observation). We meta-learn the weights of this network using an approach similar to the one proposed by (Finn et al., 2017) which we implement using the higher library (Grefenstette et al., 2019). For each agent update, we also perform a meta-update of the transformation function by splitting PPO’s buffer into training and validation sets. We refer to this approach as Meta-DrAC.

### 4. Experiments

See Appendix A for a full description of the baselines, experimental setup, and hyperparameters used.

#### 4.1. Generalization Ability

Table 1 shows train and test performance on Procgen. UCB-DrAC significantly outperforms standard RL (PPO) and other baselines designed to improve generalization in RL (Rand-FM (Lee et al., 2020) and IBAC-SNI (Igl et al., 2019)). Regularizing the policy and value function leads to improvements over merely using data augmentation, and thus the performance of DrAC is better than that of RAD (both using the best augmentation for each game). Our experiments show that the most effective way of automatically finding an augmentation is UCB-DrAC. As expected, meta-learning the weights of a transformation function parameterized by a CNN using Meta-DrAC performs reasonably well on the games in which the random convolution augmentation helps. But overall, Meta-DrAC and RL2-DrAC are worse than UCB-DrAC. In addition, UCB is generally more stable, easier to implement, and requires less fine-tuning compared to meta-learning algorithms. See Figures 4 and 5 in the Appendix for a comparison of these three approaches on each game. Moreover, automatically selecting the augmentation from a given set using UCB-DrAC performs just as well or even better than a method that uses the best augmentation for each task throughout the entire training process. UCB-DrAC also achieves higher returns than an ablation that uses a uniform distribution to select an augmentation each time, Rand-DrAC. Similarly, UCB-DrAC is better than Crop-DrAC, which uses crop for all the games (which is the best augmentation for half of the Procgen games).

#### 4.2. Regularization Effect

In Section 3.1, we argued that additional regularization terms are needed in order to make the use of data augmentation in RL theoretically sound. However, one might wonder if this problem actually appears in practice. Thus, we empirically investigate the effect of regularizing the policy and value function. For this purpose, we compare the performance of RAD and DrAC with grayscale and random convolution augmentations on Chaser, Miner, and StarPilot. Figure 2 shows that not regularizing the policy and value...
4.3. Automatic Augmentation

Our experiments indicate there is not a single augmentation that works best across all Procgen games, which is consistent with the observations of (Laskin et al., 2020). Moreover, our intuitions regarding the best transformation for each game might be misleading. For example, at a first sight, Climber appears to be somewhat similar to CoinRun or Jumper, but the augmentation that performs best on Climber is color-jitter, while for CoinRun and Jumper is random-conv. In contrast, Miner looks like a very different game from CoinRun or Jumper, but they all have the same best performing augmentation, namely random-conv. These observations further underline the need for a method that can automatically find the right augmentation for each task.

Table 1 along with Figures 4 and 5 in the Appendix compare different approaches for automatically finding an augmentation, showing that using a simple multi-armed bandit algorithm like UCB performs best and reaches the asymptotic performance obtained when the most effective transformation for each game is used throughout the entire training process. Figure 3 illustrates an example of UCB’s policy during training on Ninja, showing that it converges to always selecting the most effective augmentation (i.e. color-jitter).

5. Discussion

In this work, we propose UCB-DrAC, a method for automatically finding an effective data augmentation for RL tasks. Our approach enables the principled use of data augmentation with actor-critic algorithms by regularizing the policy and value functions with respect to state transformations. We show that UCB-DrAC avoids the theoretical and empirical pitfalls typical in naive applications of data augmentation in RL. Our approach improves training performance by ∼ 16% and test performance by ∼ 40% on the Procgen benchmark, relative to standard RL methods such as PPO (Schulman et al., 2017). UCB-DrAC outperforms, on both train and test environments, several methods specifically designed to aid generalization in RL (Igl et al., 2019; Lee et al., 2020; Laskin et al., 2020).
References


