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 π_θ and a value function V_θ 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

[1]Department of Computer Science, New York University, New York, USA [2]Facebook AI Research, New York, USA. Correspondence to: Roberta Raileanu <raileanu@cs.nyu.edu>.

Accepted at Workshop on Inductive Biases, Invariances and Generalization, International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

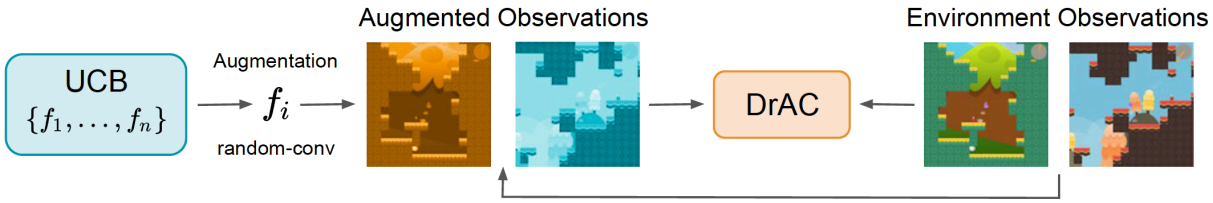

Figure 1. Overview of UCB-DrAC. A UCB bandit selects an image transformation (*e.g.* random-conv) and applies it to observations. The augmented and original observations are passed to a modified actor-critic RL agent (DrAC) which uses them to improve generalization.

estimated using importance sampling:

$$\sum_{a\in\mathcal{A}} \pi_\theta(a|s)\hat{A}_{\theta_{\text{old}}}(s,a) = \mathbb{E}_{a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right], \tag{1}$$

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

3. Automatic Data Augmentation for RL

3.1. Data Augmentation in RL

Image augmentation has been successfully applied in computer vision for improving generalization on object classification tasks (Simard et al., 2003; Cireşan et al., 2011; Ciregan et al., 2012; Krizhevsky et al., 2012). However, as noted by Kostrikov et al. (2020), these tasks are invariant to certain image transformations such as rotations or flips. In contrast, RL tasks are not always invariant to such augmentations (Kostrikov et al., 2020).

If transformations are naively applied to (some of the) observations in PPO's buffer, as done in Cobbe et al. (2018); Laskin et al. (2020), the PPO objective changes and equation (1) is replaced by

$$\sum_{a\in\mathcal{A}} \pi_\theta(a|s)\hat{A}_{\theta_{\text{old}}}(s,a) = \mathbb{E}_{a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a|f(s))}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\right], \tag{2}$$

where $f : \mathcal{S}\times\mathcal{H} \to \mathcal{S}$ is the image transformation. However, the right hand side of the above equation is not a sound estimate of the left hand side because $\pi_\theta(a|f(s)) \neq \pi_\theta(a|s)$, since nothing constrains $\pi_\theta(a|f(s))$ to be close to $\pi_\theta(a|s)$. Moreover, one can define certain transformations $f(\cdot)$ that result in an arbitrarily large ratio $\pi_\theta(a|f(s))/\pi_\theta(a|s)$.

3.2. Policy and Value Function Regularization

Following (Kostrikov et al., 2020), we define an optimality-invariant state transformation $f : \mathcal{S}\times\mathcal{H} \to \mathcal{S}$ as a mapping that preserves both the agent's policy π and its value function V such that $V(s) = V(f(s,\nu))$ and $\pi(a|s) = \pi(a|f(s,\nu))$, $\forall s \in \mathcal{S},\ \nu \in \mathcal{H}$, where ν are the parameters of $f(\cdot)$, drawn from the set of all possible parameters $\mathcal{H}$.

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,

$$G_\pi = KL\left[\pi_\theta(a|f(s,\nu)) \mid \pi(a|s)\right], \tag{3}$$

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

$$G_V = \left(V_\theta\left(f(s,\nu)\right) - V(s)\right)^2. \tag{4}$$

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

$$J_{\text{DrAC}} = J_{\text{PPO}} - \alpha_r(G_\pi + G_V) \tag{5}$$

where α_r is the weight of the regularization term.

The use of G_π 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_π 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_π 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

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 $\mathcal{F} = \{f_1, \dots, f_n\}$. 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:

$$f_t = \text{argmax}_{f \in \mathcal{F}} \left[Q(f) + c \sqrt{\frac{\log(t)}{N(f)}} \right] \tag{6}$$

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

		PPO-Normalized Return (%)			
		Train		Test	
	Method	**Median**	**Mean**	**Median**	**Mean**
(a)	PPO	100.0	100.0	100.0	100.0
	Rand-FM	91.83	87.26	90.16	75.5
	IBAC-SNI	95.34	104.1	88.19	91.27
(b)	DrAC (Best)	**109.0**	116.2	**117.0**	137.4
	RAD (Best)	102.0	109.7	112.2	134.6
(c)	UCB-DrAC	100.3	**116.2**	115.3	**139.2**
	RL2-DrAC	99.1	97.7	108.0	105.2
	Meta-DrAC	79.7	79.3	85.3	86.0
(d)	Rand-DrAC	101.0	100.3	101.2	103.3
	Crop-DrAC	98.7	113.3	112.6	129.8
	UCB-RAD	93.1	104.2	103.1	126.9

Table 1. Train and test performance for the Procgen benchmark (aggregated over all 16 tasks, 5 seeds). (a) compares PPO with two baselines specifically designed to improve generalization in RL and shows that they do not significantly help. (b) compares using the best augmentation from our set with and without regularization, corresponding to DrAC and RAD respectively, and shows that regularization improves performance on both train and test. (c) compares different approaches for automatically finding an augmentation for each task, namely using UCB or RL^2 for selecting the best transformation from a given set, or meta-learning the weights of a convolutional network (Meta-DrAC). (d) shows additional ablations: Rand-DrAC selects an augmentation using a uniform distribution, Crop-DrAC uses image crops for all tasks, and UCB-RAD is an ablation that does not use the regularization losses. UCB-DrAC performs best on both train and test, and achieves a return comparable with or better than DrAC (which uses the best augmentation).

function with respect to the transformations used can lead to drastically worse performance than vanilla RL methods, further emphasizing the importance of these loss terms. In contrast, using the regularization terms as part of the RL objective (as DrAC does) results in an agent that is comparable or, in some cases, significantly better than PPO.

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 exampler, 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

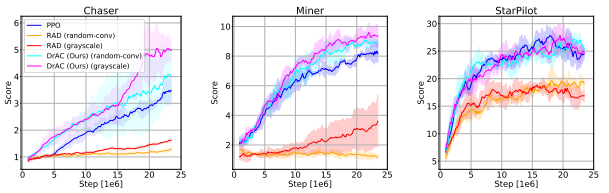

Figure 2. Comparison between RAD and DrAC with the same augmentations, grayscale and random convolution, on the test environments of Chaser (left), Miner (center), and StarPilot (right). While DrAC's performance is comparable or better than PPO's, not using the regularization terms, *i.e.* using RAD, significantly hurts performance relative to PPO. This is because, in contrast with DrAC, RAD does not use a principled (importance sampling) estimate of PPO's objective.

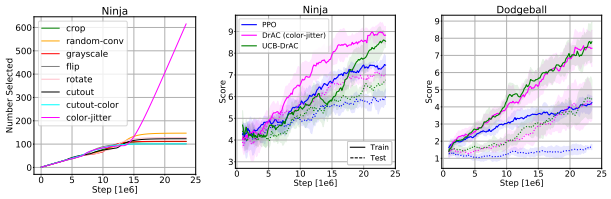

Figure 3. (a) Cumulative number of times UCB selects each augmentation over the course of training for Ninja. Train and test performance for PPO, DrAC with the best augmentation for each game (color-jitter and crop, respectively), and UCB-DrAC for Ninja (b) and Dodgeball (c). UCB-DrAC finds the most effective augmentation from the given set and reaches the performance of DrAC. Our methods improve both train and test performance.

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 $\sim 16\%$ and test performance by $\sim 40\%$ 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

Auer, P. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Ciregan, D., Meier, U., and Schmidhuber, J. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3642–3649. IEEE, 2012.

Cireşan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Farebrother, J., Machado, M. C., and Bowling, M. H. Generalization and regularization in dqn. *ArXiv*, abs/1810.00123, 2018.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Gamrian, S. and Goldberg, Y. Transfer learning for related reinforcement learning tasks via image-to-image translation. *ArXiv*, abs/1806.07377, 2019.

Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

Igl, M., Ciosek, K., Li, Y., Tschiatschek, S., Zhang, C., Devlin, S., and Hofmann, K. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pp. 13956–13968, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

Kostrikov, I., Yarats, D., and Fergus, R. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

Lee, K., Lee, K., Shin, J., and Lee, H. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. In *ICML*, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Simard, P. Y., Steinkraus, D., Platt, J. C., et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

Song, X., Jiang, Y., Tu, S., Du, Y., and Neyshabur, B. Observational overfitting in reinforcement learning. *ArXiv*, abs/1912.02975, 2020.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Zhang, A., Ballas, N., and Pineau, J. A dissection of overfitting and generalization in continuous reinforcement learning. *ArXiv*, abs/1806.07937, 2018.