Automatic Data Augmentation for Generalization in Reinforcement Learning

Roberta Raileanu
New York University
raileanu@cs.nyu.edu

Max Goldstein
New York University
mag1038@nyu.edu

Denis Yarats
New York University
Facebook AI Research
denisyarats@cs.nyu.edu

Ilya Kostrikov
New York University
kostrikov@cs.nyu.edu

Rob Fergus
New York University
fergus@cs.nyu.edu

Abstract

Deep reinforcement learning (RL) agents often fail to generalize beyond their training environments. To alleviate this problem, recent work has proposed the use of data augmentation. However, different tasks tend to benefit from different types of augmentations and selecting the right one typically requires expert knowledge. In this paper, we introduce three approaches for automatically finding an effective augmentation for any RL task. 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 actor-critic algorithms. Our method achieves a new state-of-the-art on the Procgen benchmark and outperforms popular RL algorithms on DeepMind Control tasks with distractors. In addition, our agent learns policies and representations which are more robust to changes in the environment that are irrelevant for solving the task, such as the background.

1 Introduction

Generalization to new environments remains a major challenge in deep reinforcement learning (RL). Current methods fail to generalize to unseen environments even when trained on similar settings [19, 51, 71, 11, 21, 12, 60]. This indicates that standard RL agents memorize specific trajectories rather than learning transferable skills. Several strategies have been proposed to alleviate this problem, such as the use of regularization [19, 71, 11, 28], data augmentation [11, 44, 69, 38, 41], or representation learning [72, 74]. In this work, we focus on the use of data augmentation in RL. We identify key differences between supervised learning and reinforcement learning which need to be taken into account when using data augmentation in RL.

More specifically, we show that a naive application of data augmentation can lead to both theoretical and practical problems with standard RL algorithms, such as unprincipled objective estimates and poor performance. 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 approach allows the agent to learn useful behaviors (outperforming strong RL baselines) in settings in which a naive use of data augmentation completely fails or converges to a sub-optimal policy. While we use Proximal Policy Optimization (PPO, Schulman et al. [56]) to describe and validate our approach, the method

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

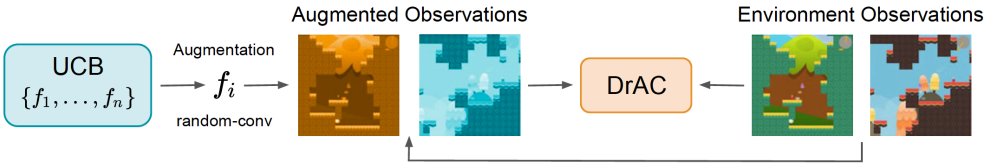

Figure 1: Overview of UCB-DrAC. A UCB bandit selects an image transformation (*e.g.* random-conv) and applies it to the observations. The augmented and original observations are passed to a regularized actor-critic agent (*i.e.* DrAC) which uses them to learn a policy and value function which are invariant to this transformation.

can be easily integrated with any actor-critic algorithm with a discrete stochastic policy such as A3C [49], SAC [24], or IMPALA [17].

The current use of data augmentation in RL either relies on expert knowledge to pick an appropriate augmentation [11, 44, 38] or separately evaluates a large number of transformations to find the best one [69, 41]. In this paper, we propose three methods for automatically finding a useful augmentation for a given RL task. The first two learn to select the best augmentation from a fixed set, using either a variant of the upper confidence bound algorithm (UCB, Auer [2]) or meta-learning (RL^2, Wang et al. [66]). We refer to these methods as **UCB-DrAC** and **RL2-DrAC**, respectively. The third method, **Meta-DrAC**, directly meta-learns the weights of a convolutional network, without access to predefined transformations (MAML, Finn et al. [20]). Figure 1 gives an overview of UCB-DrAC.

We evaluate these approaches on the *Procgen* generalization benchmark [12] which consists of 16 procedurally generated environments with visual observations. Our results show that UCB-DrAC is the most effective among these at finding a good 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 [28, 44, 41] on both train and test. In addition, we show that our agent learns policies and representations that are more invariant to changes in the environment which do not alter the reward or transition function (*i.e.* they are inconsequential for control), such as the background theme.

To summarize, our work makes the following contributions: (i) we introduce a principled way of using data augmentation with actor-critic algorithms, (ii) we propose a practical approach for automatically selecting an effective augmentation in RL settings, (iii) we show that the use of data augmentation leads to policies and representations that better capture task invariances, and (iv) we demonstrate state-of-the-art results on the Procgen benchmark and outperform popular RL methods on four DeepMind Control tasks with natural and synthetic distractors.

2 Background

We consider a distribution $q(m)$ of Partially Observable Markov Decision Processes (POMDPs, [6]) $m \in \mathcal{M}$, with m defined by the tuple $(\mathcal{S}_m, \mathcal{O}_m, \mathcal{A}, T_m, R_m, \gamma)$, where $\mathcal{S}_m$ is the state space, $\mathcal{O}_m$ is the observation space, $\mathcal{A}$ is the action space, $T_m(s'|s,a)$ is the transition function, $R_m(s,a)$ is the reward function, and γ is the discount factor. During training, we restrict access to a fixed set of POMDPs, $M_{train} = \{m_1, \ldots, m_n\}$, where $m_i \sim q$, $\forall\, i = \overline{1,n}$. The goal is to find a policy π_θ which maximizes the expected discounted reward over the entire distribution of POMDPs, $J(\pi_\theta) = \mathbb{E}_{q,\pi,T_m,p_m}\left[\sum_{t=0}^{T} \gamma^t R_m(s_t, a_t)\right]$.

In practice, we use the Procgen benchmark which contains 16 procedurally generated games. Each game corresponds to a distribution of POMDPs $q(m)$, and each level of a game corresponds to a POMDP sampled from that game's distribution $m \sim q$. The POMDP m is determined by the seed (*i.e.* integer) used to generate the corresponding level. Following the setup from Cobbe et al. [12], agents are trained on a fixed set of $n = 200$ levels (generated using seeds from 1 to 200) and tested on the full distribution of levels (generated by sampling seeds uniformly at random from all computer integers).

Proximal Policy Optimization (PPO, Schulman et al. [56]) 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 POMDP. PPO alternates between sampling data through interaction with the environment and maximizing a clipped surrogate objective function using stochastic gradient ascent.

One component of the PPO objective is the policy gradient term J_{PG}, which is estimated using importance sampling:

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

where $\hat{A}(\cdot)$ is an estimate of the advantage function, $\pi_{\theta_{\mathrm{old}}}$ is the behavior policy used to collect trajectories (*i.e.* that generates the training distribution of states and actions), and π_θ is the policy we want to optimize (*i.e.* that generates the true distribution of states and actions). See Appendix A for a full description of PPO.

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 [58, 9, 8, 39]. As noted by Kostrikov et al. [38], those tasks are invariant to certain image transformations such as rotations or flips, which is not always the case in RL. For example, if your observation is flipped, the corresponding reward will be reversed for the left and right actions and will not provide an accurate signal to the agent. While data augmentation has been previously used in RL settings without other algorithmic changes [11, 69, 41], we argue that this approach is not theoretically sound.

If transformations are naively applied to observations in PPO's buffer, as done in Laskin et al. [41], the PPO objective changes and equation (1) is replaced by

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

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 $\pi_\theta(a|s)$. Note that in the on-policy case when $\theta = \theta_{old}$, the ratio used to estimate the advantage should be equal to one, which is not necessarily the case when using equation (2). In fact, one can define certain transformations $f(\cdot)$ that result in an arbitrarily large ratio $\pi_\theta(a|f(s))/\pi_{\theta_{old}}(a|s)$.

Figure 3 shows examples where a naive use of data augmentation prevents PPO from learning a good policy in practice, suggesting that this is not just a theoretical concern. In the following section, we propose an algorithmic change that enables the use of data augmentation with actor-critic algorithms in a principled way.

3.2 Policy and Value Function Regularization

Inspired by the recent work of Kostrikov et al. [38], we propose two novel regularization terms for the policy and value functions that enable the proper use of data augmentation for actor-critic algorithms. Our algorithmic contribution differs from that of Kostrikov et al. [38] in that it constrains both the actor and the critic, as opposed to only regularizing the Q-function. This allows our method to be used with a different (and arguably larger) class of RL algorithms, namely those that learn a policy and a value function.

Following Kostrikov et al. [38], 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 such transformation of the input state, we propose an additional loss term for regularizing the policy,

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

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

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

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 (see Algorithm 1).

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 translations). 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, TRPO, ACER, SAC, or IMPALA) without any other changes.

Note that when using DrAC, as opposed to the method proposed by Laskin et al. [41], we still use the correct importance sampling estimate of the left hand side objective in equation (1) (instead of a wrong estimate as in equation (2)). 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. Without these extra terms, the only way to use data augmentation is as explained in Section 3.1, which leads to inaccurate estimates of the PPO objective. Hence, DrAC benefits from the regularizing effect of using data augmentation, while mitigating adverse consequences on the RL objective. See Appendix B for a more detailed explanation of why a naive application of data augmentation with certain policy gradient algorithms is theoretically unsound.

Algorithm 1 DrAC: **D**ata-**r**egularized **A**ctor-**C**ritic applied to PPO
Black: unmodified actor-critic algorithm.
Cyan: image transformation.
Red: policy regularization.
Blue: value function regularization.

Hyperparameters: image transformation f, regularization loss coefficient α_r, minibatch size M, replay buffer size T, number of updates K.
for $k = 1, \ldots, K$ **do**
  Collect a new set of transitions $\mathcal{D} = \{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{T}$ using π_θ.
  for $j = 1, \ldots, *\frac{T}{M}$ **do**
    $\{(s_i, a_i, r_i, s_{i+1})\}_{i=1}^{M} \sim \mathcal{D}$ ▷ Sample a minibatch of transitions
    for $i = 1, \ldots, M$ **do**
      $\nu_i \sim \mathcal{H}$ ▷ Sample the augmentation parameters
      $\hat{\pi}_i \leftarrow \pi_\phi(\cdot|s_i)$ ▷ Compute the policy targets
      $\hat{V}_i \leftarrow V_\phi(s_i)$ ▷ Compute the value function targets
    end for
    $G_\pi(\theta) = \frac{1}{M}\sum_{i=1}^{M} KL\left[\hat{\pi}_i \mid \pi_\theta(\cdot|f(s_i, \nu_i))\right]$ ▷ Regularize the policy
    $G_V(\phi) = \frac{1}{M}\sum_{i=1}^{M} \left(\hat{V}_i - V_\phi\left(f(s_i, \nu_i)\right)\right)^2$ ▷ Regularize the value function
    $J_{\text{DrAC}}(\theta, \phi) = J_{\text{PPO}}(\theta, \phi) - \alpha_r(G_\pi(\theta) + G_V(\phi))$ ▷ Compute the total loss function
    $\theta \leftarrow \arg\max_\theta J_{\text{DrAC}}$ ▷ Update the policy
    $\phi \leftarrow \arg\max_\phi J_{\text{DrAC}}$ ▷ Update the value function
  end for
end for

3.3 Automatic Data Augmentation

Since different tasks benefit from different types of transformations, we would like to design a method that can automatically find an effective transformation for any given task. Such a technique would significantly reduce the computational requirements for applying data augmentation in RL. 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 $\mathcal{F} = \{f^1, \ldots, f^n\}$. A popular algorithm for such settings is the upper confidence

bound or UCB [2], which selects actions according to the following policy:

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

where f_t is the transformation selected at time step t, $N_t(f)$ is the number of times transformation f has been selected before time step t and c is UCB's exploration coefficient. 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: $N_t(f) = N_{t-1}(f) + 1$. Next, we collect rollouts with the new policy and update the Q-function: $Q_t(f) = \frac{1}{K} \sum_{i=t-K}^{t} \mathcal{R}(f_i = f)$, 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** (Algorithm 4). Note that UCB-DrAC's estimation 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. [66]. Before each DrAC update, the meta-learner selects an augmentation, which is then used to update the agent using equation (5). We then 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** (Algorithm 4).

Meta-Learning the Weights of an 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. We meta-learn the weights of this network using an approach similar to the one proposed by Finn et al. [20]. For each agent update, we also perform a meta-update of the transformation function by splitting DrAC's buffer into meta-train and meta-test sets. We refer to this approach as **Meta-DrAC** (Algorithm 4). More details about the implementation of these methods can be found in Appendix D.

4 Experiments

In this section, we evaluate our methods on **four DeepMind Control environments with natural and synthetic distractors** and the **full Procgen benchmark** [12] which consists of 16 procedurally generated games (see Figure 6 in Appendix G). Procgen has a number of attributes that make it a good testbed for generalization in RL: (i) it has a diverse set of games in a similar spirit with the ALE benchmark [5], (ii) each of these games has procedurally generated levels which present agents with meaningful generalization challenges, (iii) agents have to learn motor control directly from images, and (iv) it has a clear protocol for testing generalization.

All environments use a discrete 15 dimensional action space and produce $64 \times 64 \times 3$ RGB observations. We use Procgen's *easy* setup, so for each game, agents are trained on 200 levels and tested on the full distribution of levels. We use PPO as a base for all our methods. More details about our experimental setup and hyperparameters can be found in Appendix E.

Data Augmentation. In our experiments, we use a set of eight transformations: *crop, grayscale, cutout, cutout-color, flip, rotate, random convolution* and *color-jitter* [39, 15]. We use **RAD**'s [41] implementation of these transformations, except for *crop*, in which we pad the image with 12 (boundary) pixels on each side and select random crops of 64×64. We found this implementation of *crop* to be significantly better on Procgen, and thus it can be considered an empirical upper bound of RAD in this case. For simplicity, we will refer to our implementation as RAD. **DrAC** uses the same set of transformations as RAD, but is trained with additional regularization losses for the actor and the critic, as described in Section 3.2.

Automatic Selection of Data Augmentation. We compare three different approaches for automatically finding an effective transformation: **UCB-DrAC** which uses UCB [2] to select an augmentation from a given set, **RL2-DrAC** which uses RL^2 [67] to do the same, and **Meta-DrAC** which uses MAML [20] to meta-learn the weights of a convolutional network. Meta-DrAC is implemented using the *higher* library [22]. Note that we do not expect these approaches to be better than DrAC with the

Table 1: Train and test performance for the Procgen benchmark (aggregated over all 16 tasks, 10 seeds). (a) compares PPO with four 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: DrA regularizes only the actor, DrC regularizes only the critic, 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).

		PPO-Normalized Return (%)					
		Train			**Test**		
	Method	**Median**	**Mean**	**Std**	**Median**	**Mean**	**Std**
(a)	PPO	100.0	100.0	7.2	100.0	100.0	8.5
	Rand-FM	93.4	87.6	8.9	91.6	78.0	9.0
	IBAC-SNI	91.9	103.4	8.5	86.2	102.9	8.6
	Mixreg	95.8	104.2	3.1	105.9	114.6	3.3
	PLR	101.5	106.7	5.6	107.1	128.3	5.8
(b)	**DrAC (Best) (Ours)**	**114.0**	**119.6**	9.4	118.5	138.1	10.5
	RAD (Best)	103.7	109.1	9.6	114.2	131.3	9.4
(c)	**UCB-DrAC (Ours)**	102.3	118.9	8.8	**118.5**	**139.7**	8.4
	RL2-DrAC	96.3	95.0	8.8	99.1	105.3	7.1
	Meta-DrAC	101.3	100.1	8.5	101.7	101.2	7.3
(b)	DrA (Best)	102.6	117.7	11.1	110.8	126.6	9.0
	DrC (Best)	103.3	108.2	10.8	110.6	115.4	8.5
	Rand-DrAC	100.4	99.5	8.4	102.4	103.4	7.0
	Crop-DrAC	97.4	112.8	9.8	114.0	132.7	11.0
	UCB-RAD	100.4	104.8	8.4	103.0	125.9	9.5

best augmentation. In fact, DrAC with the best augmentation can be considered to be an upper bound for these automatic approaches since it uses the best augmentation during the entire training process.

Ablations. DrC and **DrA** are ablations to **DrAC** that use only the value or only the policy regularization terms, respectively. DrC can be thought of an analogue of DrQ [38] applied to PPO rather than SAC. **Rand-DrAC** uses a uniform distribution to select an augmentation each time. **Crop-DrAC** uses crop for all games (which is the most effective augmentation on half of the Procgen games). **UCB-RAD** combines UCB with RAD (*i.e.* it does not use the regularization terms).

Baselines. We also compare with **Rand-FM** [44], **IBAC-SNI** [28], **Mixreg** [66], and **PLR** [32], four methods specifically designed for improving generalization in RL and previously tested on Procgen environments. Rand-FM uses a random convolutional networks to regularize the learned representations, while IBAC-SNI uses an information bottleneck with selective noise injection. Mixreg uses mixtures of observations to impose linearity constraints between the agent's inputs and the outputs, while PLR samples levels according to their learning potential.

Evaluation Metrics. At the end of training, for each method and each game, we compute the average score over 100 episodes and 10 different seeds. The scores are then normalized using the corresponding PPO score on the same game. We aggregate the normalized scores over all 16 Procgen games and report the resulting mean, median, and standard deviation (Table 1). For a per-game breakdown, see Tables 6 and 7 in Appendix J.

4.1 Generalization Performance on Procgen

Table 1 shows train and test performance on Procgen. UCB-DrAC significantly outperforms PPO, Rand-FM, IBAC-SNI, PLR, and Mixreg. As shown in Jiang et al. [32], combining PLR with UCB-

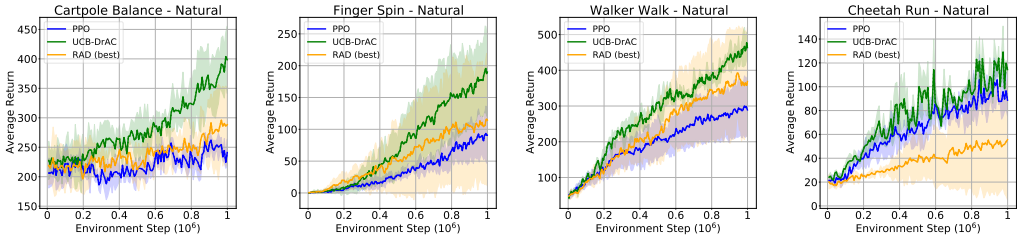

Figure 2: Average return on DMC tasks with natural video backgrounds with mean and standard deviation computed over 5 seeds. UCB-DrAC outperforms PPO and RAD with the best augmentations.

DrAC achieves a new state-of-the-art on Procgen leading to a 76% gain over PPO. 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). In addition, we demonstrate the importance of regularizing both the policy and the value function rather than either one of them by showing that DrAC is superior to both DrA and DrC (see Figures 9 and 10 in Appendix K). Our experiments show that the most effective way of automatically finding an augmentation is UCB-DrAC. As expected, meta-learning the weights of 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 8 and 7 in Appendix K for a comparison of these three approaches on each game. Moreover, automatically selecting the augmentation from a given set using UCB-DrAC performs similarly 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. Nevertheless, UCB-DrAC is better than Crop-DrAC, which uses crop for all the games (which is the best augmentation for eight of the Procgen games as shown in Tables 4 and 5 from Appendix H).

4.2 DeepMind Control with Distractors

In this section, we evaluate our approach on the DeepMind Control Suite from pixels (DMC, Tassa et al. [64]). We use four tasks, namely Cartpole Balance, Finger Spin, Walker Walk, and Cheetah Run, in three settings with different types of backgrounds, namely the *default*, *simple* distractors, and *natural* videos from the Kinetics dataset [35], as introduced in Zhang et al. [73]. See Figure 11 for a few examples. Note that in the simple and natural settings, the background is sampled from a list of videos at the beginning of each episode, which creates spurious correlations between the backgrounds and the rewards. In the simple and natural distractor settings, as shown in Figure 13, UCB-DrAC outperforms PPO and RAD on all these environments in the most challenging setting with natural distractors. See Appendix L for results on DMC with default and simple distractor backgrounds, where our method also outperforms the baselines.

4.3 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 3 shows that not regularizing the policy and value 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.

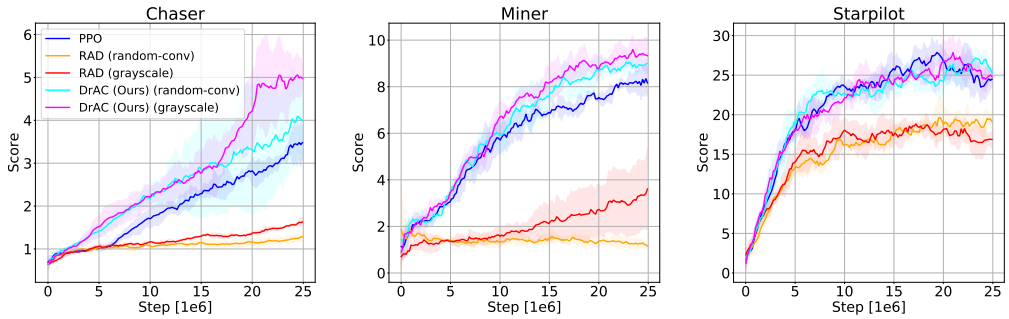

Figure 3: Comparison between RAD and DrAC with the same augmentations, grayscale and random convolution, on the test environments of Chaser (left), Miner (center), and StarPilot (right). While DrAC's performance is comparable or better than PPO's, not using the regularization terms, *i.e.* using RAD, significantly hurts performance relative to PPO. This is because, in contrast to DrAC, RAD does not use a principled (importance sampling) estimate of PPO's objective.

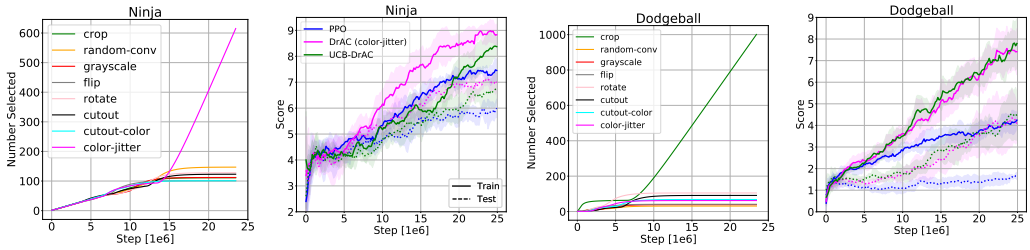

Figure 4: Cumulative number of times UCB selects each augmentation over the course of training for Ninja (a) and Dodgeball (c). Train and test performance for PPO, DrAC with the best augmentation for each game (color-jitter and crop, respectively), and UCB-DrAC for Ninja (b) and Dodgeball (d). UCB-DrAC finds the most effective augmentation from the given set and reaches the performance of DrAC. Our methods improve both train and test performance.

4.4 Automatic Augmentation

Our experiments indicate there is not a single augmentation that works best across all Procgen games (see Tables 4 and 5 in Appendix H). Moreover, our intuitions regarding the best transformation for each game might be misleading. For example, at a first sight, Ninja appears to be somewhat similar to Jumper, but the augmentation that performs best on Ninja is color-jitter, while for Jumper is random-conv (see Tables 4 and 5). In contrast, Miner seems like a different type of game than Climber or Ninja, but they all have the same best performing augmentation, namely color-jitter. These observations further underline the need for a method that can automatically find the right augmentation for each task.

Table 1 along with Figures 8 and 7 in the Appendix compare different approaches for automatically finding an augmentation, showing that UCB-DrAC performs best and reaches the asymptotic performance obtained when the most effective transformation for each game is used throughout the entire training process. Figure 4 illustrates an example of UCB's policy during training on Ninja and Dodgeball, showing that it converges to always selecting the most effective augmentation, namely color-jitter for Ninja and crop for Dodgeball. Finally, Figure 5 in Appendix F illustrates how UCB's behavior varies with its exploration coefficient.

4.5 Robustness Analysis

To further investigate the generalizing ability of these agents, we analyze whether the learned policies and state representations are invariant to changes in the observations which are irrelevant for solving the task.

Table 2: JSD and Cycle-Consistency (%) (aggregated across all Procgen tasks) for PPO, RAD and UCB-DrAC, measured between observations that vary only in their background themes (*i.e.* colors and patterns that do not interact with the agent). UCB-DrAC learns more robust policies and representations that are more invariant to changes in the observation that are irrelevant for the task.

			Cycle-Consistency (%)			
	JSD		2-way		3-way	
Method	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**
PPO	0.25	0.23	20.50	18.70	12.70	5.60
RAD	0.19	0.18	24.40	22.20	15.90	8.50
UCB-DrAC	**0.16**	**0.15**	**27.70**	**24.80**	**17.30**	**10.30**

We first measure the Jensen-Shannon divergence (JSD) between the agent's policy for an observation from a training level and a modified version of that observation with a different background theme (*i.e.* color and pattern). Note that the JSD also represents a lower bound for the joint empirical risk across train and test [30]. The background theme is randomly selected from the set of backgrounds available for all other Procgen environments, except for the one of the original training level. Note that the modified observation has the same semantics as the original one (with respect to the reward function), so the agent should have the same policy in both cases. Moreover, many of the backgrounds are not uniform and can contain items such as trees or planets which can be easily misled for objects the agent can interact with. As seen in Table 2, UCB-DrAC has a lower JSD than PPO, indicating that it learns a policy that is more robust to changes in the background.

To quantitatively evaluate the quality of the learned representation, we use the cycle-consistency metric proposed by Aytar et al. [3] and also used by Lee et al. [44]. See Appendix C for more details about this metric. Table 2 reports the percentage of input observations in the seen environment that are cycle-consistent with trajectories in modified unseen environments, which have a different background but the same layout. UCB-DrAC has higher cycle-consistency than PPO, suggesting that it learns representations that better capture relevant task invariances.

5 Related Work

Generalization in Deep RL. A recent body of work has pointed out the problem of overfitting in deep RL [53, 46, 34, 51, 71, 75, 50, 11, 12, 33, 52, 40, 23]. A promising approach to prevent overfitting is to apply regularization techniques originally developed for supervised learning such as dropout [62, 28] or batch normalization [31, 19, 28]. For example, Igl et al. [28] use selective noise injection with a variational information bottleneck, while Lee et al. [44] regularize the agent's representation with respect to random convolutional transformations. The use of state abstractions has also been proposed for improving generalization in RL [72, 74, 1]. Similarly, Roy and Konidaris [54] and Sonar et al. [59] learn domain-invariant policies via feature alignment, while Stooke et al. [63] decouple representation from policy learning. Igl et al. [29] reduce non-stationarity using policy distillation, while Mazoure et al. [47] maximize the mutual information between the agent's representation of successive time steps. Jiang et al. [32] improve efficiency and generalization by sampling levels according to their learning potential, while Wang et al. [67] use mixtures of observations to impose linearity constraints between the agent's inputs and the outputs. More similar to our work, Cobbe et al. [11], Ye et al. [69] and Laskin et al. [41] add augmented observations to the training buffer of an RL agent. However, as we show here, naively applying data augmentation in RL can lead to both theoretical and practical issues. Our algorithmic contributions alleviate these problems while still benefitting from the regularization effect of data augmentation.

Data Augmentation has been extensively used in computer vision for both supervised [42, 4, 43, 58, 9, 10, 39] and self-supervised [16, 48] learning. More recent work uses data augmentation for contrastive learning, leading to state-of-the-art results on downstream tasks [70, 26, 25, 7]. Domain randomization can also be considered a type of data augmentation, which has proven useful for transferring RL policies from simulation to the real world [65]. However, it requires access to a physics simulator, which is not always available. Recently, a few papers propose the use of data augmentation in RL [11, 44, 61, 38, 41], but all of them use a fixed (set of) augmentation(s) rather than automatically finding the most effective one. The most similar work to ours is that

of Kostrikov et al. [38], who propose to regularize the Q-function in Soft Actor-Critic (SAC) [24] using random shifts of the input image. Our work differs from theirs in that it automatically selects an augmentation from a given set, regularizes both the actor and the critic, and focuses on the problem of generalization rather than sample efficiency. While there is a body of work on the automatic use of data augmentation [14, 13, 18, 57, 45], these approaches were designed for supervised learning and, as we explain here, cannot be applied to RL without further algorithmic changes.

6 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 19% and test performance by 40% on the Procgen benchmark, and sets a new state-of-the-art on the Procgen benchmark. In addition, the learned policies and representations are more invariant to spurious correlations between observations and rewards. One limitation of our work is the assumption that using a single type of data augmentation throughout the entire training process is optimal. In future work, we plan to study the effect of using multiple augmentations at different stages during the training of the agent's policy. Another promising avenue for future research is to use a more expressive function class for meta-learning the transformations in order to capture a wider range of inductive biases.

Acknowledgments and Disclosure of Funding

We would like to thank our NeurIPS reviewers for their valuable feedback on this work. Roberta and Max were supported by the DARPA Machine Commonsense program.

References

[1] Rishabh Agarwal, Marlos C. Machado, P. S. Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. 2021.

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

[3] Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *Advances in Neural Information Processing Systems*, pages 2930–2941, 2018.

[4] Suzanna Becker and Geoffrey E. Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355:161–163, 1992.

[5] 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.

[6] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[7] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *ArXiv*, abs/2002.05709, 2020.

[8] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3642–3649. IEEE, 2012.

[9] Dan C Cireşan, Ueli Meier, Jonathan Masci, Luca M Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *arXiv preprint arXiv:1102.0183*, 2011.

[10] Dan C. Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification. *ArXiv*, abs/1102.0183, 2011.

[11] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. *arXiv preprint arXiv:1812.02341*, 2018.

[12] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[13] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv: Computer Vision and Pattern Recognition*, 2019.

[14] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, V. Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 113–123, 2019.

[15] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[16] Alexey Dosovitskiy, Philipp Fischer, Jost Tobias Springenberg, Martin A. Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with exemplar convolutional neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38:1734–1747, 2016.

[17] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[18] Boli Fang, Miao Jiang, and Jerry J. Shen. Paganda : An adaptive task-independent automatic data augmentation. 2019.

[19] Jesse Farebrother, Marlos C. Machado, and Michael H. Bowling. Generalization and regularization in dqn. *ArXiv*, abs/1810.00123, 2018.

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[21] Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. *ArXiv*, abs/1806.07377, 2019.

[22] Edward Grefenstette, Brandon Amos, Denis Yarats, Phu Mon Htut, Artem Molchanov, Franziska Meier, Douwe Kiela, Kyunghyun Cho, and Soumith Chintala. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

[23] Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *ArXiv*, abs/2010.06740, 2020.

[24] 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.

[25] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *ArXiv*, abs/1911.05722, 2019.

[26] Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aäron van den Oord. Data-efficient image recognition with contrastive predictive coding. *ArXiv*, abs/1905.09272, 2019.

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

[28] 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 *Advances in Neural Information Processing Systems*, pages 13956–13968, 2019.

[29] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Böhmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *ArXiv*, abs/2006.05826, 2020.

[30] Maximilian Ilse, Jakub M Tomczak, and Patrick Forré. Designing data augmentation for simulating interventions. *arXiv preprint arXiv:2005.01856*, 2020.

[31] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[32] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. *ArXiv*, abs/2010.03934, 2020.

[33] Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, J. Harper, Hunter Henry, Adam Crespi, J. Togelius, and D. Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *ArXiv*, abs/1902.01378, 2019.

[34] Niels Justesen, R. Torrado, Philip Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation. *arXiv: Learning*, 2018.

[35] W. Kay, J. Carreira, K. Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, F. Viola, T. Green, T. Back, A. Natsev, Mustafa Suleyman, and Andrew Zisserman. The kinetics human action video dataset. *ArXiv*, abs/1705.06950, 2017.

[36] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[37] Ilya Kostrikov. Pytorch implementations of reinforcement learning algorithms. `https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail`, 2018.

[38] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[40] Heinrich Kuttler, Nantas Nardelli, Alexander H. Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *ArXiv*, abs/2006.13760, 2020.

[41] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[42] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. 1998.

[44] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations. https://openreview. net/forum*, 2020.

[45] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy M. Hospedales, Neil M. Robertson, and Yongxing Yang. Dada: Differentiable automatic data augmentation. *ArXiv*, abs/2003.03780, 2020.

[46] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew J. Hausknecht, and Michael H. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. In *IJCAI*, 2018.

[47] Bogdan Mazoure, R'emi Tachet des Combes, Thang Doan, Philip Bachman, and R. Devon Hjelm. Deep reinforcement and infomax learning. *ArXiv*, abs/2006.07217, 2020.

[48] Ishan Misra and Laurens van der Maaten. Self-supervised learning of pretext-invariant representations. *ArXiv*, abs/1912.01991, 2019.

[49] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

[50] Alex Nichol, V. Pfau, Christopher Hesse, O. Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *ArXiv*, abs/1804.03720, 2018.

[51] Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Xiaodong Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018.

[52] Roberta Raileanu and Tim Rocktäschel. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *ArXiv*, abs/2002.12292, 2020.

[53] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham M. Kakade. Towards generalization and simplicity in continuous control. *ArXiv*, abs/1703.02660, 2017.

[54] Josh Roy and George Konidaris. Visual transfer for reinforcement learning via wasserstein domain confusion. *arXiv preprint arXiv:2006.03465*, 2020.

[55] John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

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

[57] Yinghuan Shi, Tiexin Qin, Yong Liu, Jiwen Lu, Yang Gao, and Dinggang Shen. Automatic data augmentation by learning the deterministic policy. *ArXiv*, abs/1910.08343, 2019.

[58] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Icdar*, volume 3, 2003.

[59] Anoopkumar Sonar, Vincent Pacelli, and Anirudha Majumdar. Invariant policy optimization: Towards stronger generalization in reinforcement learning. *ArXiv*, abs/2006.01096, 2020.

[60] Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. *ArXiv*, abs/1912.02975, 2020.

[61] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *ArXiv*, abs/2004.04136, 2020.

[62] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[63] Adam Stooke, Kimin Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *ArXiv*, abs/2009.08319, 2020.

[64] Y. Tassa, Yotam Doron, Alistair Muldal, T. Erez, Y. Li, D. Casas, D. Budden, Abbas Abdolmaleki, J. Merel, Andrew Lefrancq, T. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *ArXiv*, abs/1801.00690, 2018.

[65] Joshua Tobin, Rachel H Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[66] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[67] K. Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization. *ArXiv*, abs/2010.10814, 2020.

[68] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

[69] Chang Ye, Ahmed Khalifa, Philip Bontrager, and Julian Togelius. Rotation, translation, and cropping for zero-shot generalization. *arXiv preprint arXiv:2001.09908*, 2020.

[70] Mang Ye, Xu Zhang, Pong C. Yuen, and Shih-Fu Chang. Unsupervised embedding learning via invariant and spreading instance feature. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6203–6212, 2019.

[71] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *ArXiv*, abs/1806.07937, 2018.

[72] Amy Zhang, Clare Lyle, Shagun Sodhani, Angelos Filos, Marta Kwiatkowska, Joelle Pineau, Yarin Gal, and Doina Precup. Invariant causal prediction for block mdps. *arXiv preprint arXiv:2003.06016*, 2020.

[73] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. 2020.

[74] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[75] Chiyuan Zhang, Oriol Vinyals, Rémi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *ArXiv*, abs/1804.06893, 2018.