Adaptive Scheduling of Data Augmentation for Deep Reinforcement Learning

Byungchan Ko
GSAI
POSTECH
Pohang-si, Republic of Korea
kbc6723@postech.ac.kr

Jungseul Ok
CSE / GSAI
POSTECH
Pohang-si, Republic of Korea
ockjs1@gmail.com

Abstract

We consider data augmentation technique to improve data efficiency and generalization performance of reinforcement learning (RL). Our empirical study on Open AI Procgen shows that the timing of augmentation is critical, and that to maximize test performance, an augmentation should be applied either during the entire RL training, or after the end of RL training. More specifically, if the regularization imposed by augmentation is helpful only in testing, then augmentation is best used after training than during training, because augmentation often disturbs the training process. Conversely, an augmentation that provides regularization that is useful in training should be used during the whole training period to fully utilize its benefit in terms of both generalization and data efficiency. Considering our findings, we propose a mechanism to fully exploit a set of augmentations, which identifies an augmentation (or no augmentation) to maximize RL training performance, and then utilizes all the augmentations by network distillation to maximize test performance. Our experiment empirically justifies the proposed method compared to other automatic augmentation mechanism.

1 Introduction

Reinforcement Learning (RL) from visual observations is a fundamental problem, because visual data are among the most common form; e.g., video games [20], board games [25, 26] , and robots [29, 15]. However, images are high-dimensional, so RL from vision often suffers from poor sample efficiency and poor generalization capability. due to the high-dimensional nature of images. To overcoming these problems, regularization by data augmentation has been widely considered [17, 16]; in this process, visual data are augmented by transformations that preserve the meaning or context, e.g., by cropping out unimportant parts of images, or by randomizing colors. Transformations resolve the data scarcity, and also provide an explicit implementation of inductive bias for generalization performance.

Use of the appropriate type of data augmentation significantly improves both data efficiency and generalization performance [22]. However, the correct data augmentation scheme is highly task-dependent: a poor choice can degenerate the generalization and destabilize the training [17, 22]. Hence, a variety of transformations have been developed to enlarge the set of augmentation methods [18, 11]. Meanwhile, numerous regularization methods that use data augmentation e.g., self-supervised learning [22] and representation learning [27, 11] have been proposed to stabilize training process with data augmentation by reducing the interference between RL training and regularization. Previous work addressed *what* data augmentation to use and *how* to use it, but the understanding of *when* to apply it in the training process is limited. We test the hypothesis that applying augmentation method at different epochs can have different effects. This is a non-trivial question, because the timing of data

Presented at Deep RL Workshop, NeurIPS 2021.

augmentation is not critical in supervised learning (SL) [1, 9], whereas a curriculum learning can accelerate RL training [23].

To address our main question, we devise two frameworks with different timings of augmentation: **In**tra **D**istillation with **A**ugmented observations (InDA) and **Ex**tra **D**istillation with **A**ugmented observations (ExDA) (Section 3). Implementing the regularization by augmentation in a form of distillation to minimize interference in RL training, InDA interleave the distillation with RL training, whereas ExDA applies the distillation at the end of RL training. From experiments with InDA and ExDA, we find that:

Time does matter when using augmentation in RL

in contrast to the case of SL [9] , in which the effect of data augmentation is relatively insensitive to timing. The difference mainly comes from the fact that RL agent collects samples when training, whereas SL uses a fixed data set. To be specific, the main findings from experiments are:

(i) If augmentation can accelerate RL training, then it *must* be applied as *early* as possible for sample efficiency and generalization; e.g., cropping out an unnecessary part of an image induces an efficient attention mechanism. To maximize sample efficiency, RL training must be accelerated from the beginning. However, we observe that this kind of augmentation often connotes generalization that is transferable only by a diverse experience in training process, i.e., InDA fully exploits generalization gain whereas ExDA does not. Hence, to gain generalization, this augmentation must be applied during training.

(ii) If the regularization imposed by augmentation is helpful only in testing, then to ensure sample efficiency, augmentation *must* be *postponed to the end* of RL training; e.g., augmentation by changing colors is useless when the training task shows a single background, but the testing task has multiple backgrounds. This type of augmentation may interfere with RL training, but delay of augmentation does not degrade sample efficiency and increase generalization ability. Hence, in this case, ExDA, which never disturbs RL training, is better than InDA.

(iii) The optimal time to apply augmentation for each task can be *determined* automatically by the upper confidence bound [3] (UCB) based auto augmentation [22] algorithm. We show that the choice of *no augmentation* is necessary, because augmentation can disturb the training. Thus, auto augmentation with a 'no change' ('identity') function can be used as a discriminator to identify the benefit of augmentation during training.

The above findings suggest effective timings of augmentation in RL. Our contribution also includes the InDA and ExDA algorithms, in particular, which are equipped with the distillation augmentation (DA), which address the independent interest in developing a regularization method that uses augmentation with minimal interference with RL training. The potential advantages of the proposed method over existing methods DrAC [22], RAD [17], Rand-FM [18], are discussed in Section 3.

2 Related Works

Augmented experience in RL. To solve the problem of poor generalization and sparse data, a popular approach is to generate diverse (virtual) experiences and let the RL agent learn from them. Domain randomization is a technique to produce such experiences from a simulator of a targeted system [29, 21, 23]. Accurate simulators of practical systems are difficult to obtain, and this problem limits the spectrum of applications. However, visual augmentation has no such limit because the method uses simple image transformations such as cropping, tilting and color jitter, although applications require a careful understanding of the targeted system to guide design of an appropriate image transformer. A method of a curriculum learning for domain randomization, in which the difficulty is gradually increasing [23] provided insights that coincide with some of our findings. However, we provide further understanding of the types of visual augmentation that should early or late during training.

Regularization from augmented data in vision-based RL has been implemented in various learning frameworks, including but not limited to representation [11, 28], self-supervised [22], and contrast [27]. One proposed algorithm [22] applies the UCB algorithm [3] to automatically select the most

effective augmentations over RL training, where each augmentation is considered as an arm and then evaluate effectiveness of augmentation by using a sliding window average. The idea of adapting augmentation concurs with our main message regarding the timing of augmentation. In [22], 'not augmenting' is not an option, whereas our findings indicate that it should be. In addition, [22] does not consider post augmentation followed by RL training, as in ExDA.

Different time-sensitivity of augmentation than SL. During deep learning, the early state of training often has a significant effect [6, 1]. Therefore, we devised time-sensitive methods that adapt to the progress of training, such as learning rate decay [32] and curriculum learning [30]. Golatkar *et al*. [9] studied such a time-sensitivity of regularization techniques for SL, where the effect of data augmentation in different time does not change much. We find that the time-sensitivity of augmentation can be significant in RL. This contrast may occur because of the non-stationary nature of RL, which SL does not have. Although a set of techniques originally developed for SL such as convolutional neural network, weight decay, batch normalization, dropout and self-supervised learning improve deep RL [13, 4, 19, 7, 27, 31, 12], a thorough study should be conducted before introducing a method from different learning framework, because we find the contrasting time-sensitivities of data augmentation. This spirit is also shared with an application [14] of implicit bias in SL [10, 2, 8] to RL.

3 Method

Notation. We consider a standard agent-environment interface of *vision-based* reinforcement learning in a discrete Markov decision process of state space $\mathcal{S}$, action space $\mathcal{A}$ and kernel $P = P(s_{t+1}, r_t | s_t, a_t)$ which determines the state transition and reward distribution. The goal of the RL agent is to find a policy that maximizes the expectation of cumulative reward $\sum_{t=0}^{t'-1} \gamma^t r_t$, where t' is terminating time and $\gamma \in [0, 1]$ is discount factor. At each timestep t, the agent selects an action $a_t \in \mathcal{A}$ and receives reward r_t and an image $o_{t+1} = O(s_{t+1}) \in \mathbb{R}^{k \times k}$ as an observation(possibly partial) of the next state s_{t+1}. To augment observations, we consider image transformation function $\phi : \mathbb{R}^{k \times k} \mapsto \mathbb{R}^{k \times k}$ which maintains the dimension.

Baseline RL algorithm. As the baseline deep-RL algorithm, we use Proximal Policy Optimization (PPO) [24] which is an on-policy actor-critic RL algorithm to learn policy $\pi_\theta(a \mid o)$ and value function V_θ with network parameter θ. Storing a set of recent transitions $\tau_t := (o_t, a_t, r_t, o_{t+1})$ in experience buffer $\mathcal{D}$, the network parameter θ is updated to maximize the following objective function:

$$L_{\text{PPO}}(\theta) = L_\pi(\theta) - \alpha L_V(\theta) \ , \tag{1}$$

where α is a hyperparameter and some regularization terms are omitted. The clipped policy objective function L_π and value loss function L_V are defined as:

$$L_\pi(\theta) = \hat{\mathbb{E}}\Big[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{2}$$

$$L_V(\theta) = \hat{\mathbb{E}}\Big[\big(V_\theta(o_t) - V_t^{\text{targ}}\big)^2\Big] \ , \tag{3}$$

where the expectation $\hat{\mathbb{E}}$ is taken with respect to $\tau \sim \mathcal{D}$, θ_{old} is the network parameter before the update, $\rho_t(\theta)$ is the importance ratio $\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$, $\hat{A}_t$ is advantage from Generalized Advantage Estimator [24].

Overall framework. We propose two frameworks: **In**tra **D**istillation with **A**ugmented observations (InDA) and **Ex**tra **D**istillation with **A**ugmented observations (ExDA). To be specific, both of them use PPO for RL and the **D**istillation with **A**ugmented observation (DA) (Section 3.1), for regularization, although our frameworks can use other RL algorithms and augmentation-based regularization. InDA (Section 3.2), interleaves PPO and DA, whereas ExDA (Section 3.3), performs PPO first then DA. We design InDA and ExDA to conduct either DA or PPO in each epoch (Figure 1).

3.1 Distillation with Augmented observations (DA)

DA regularizes reinforcement learning by using distillation with data augmentation, in which we train the network to output the same policies and values for given both original and augmented observations.

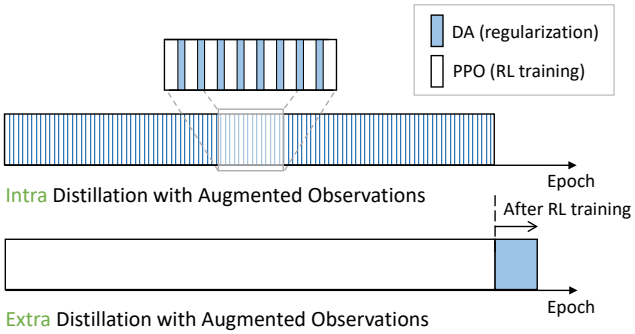


Figure 1: An illustration comparing InDA and ExDA

To do so, we fix the network θ_{old} to be distilled and store observation o_t, which is sampled from $\pi_{\theta_{\text{old}}}$, in $\mathcal{D}$. Their augmented observations are represented as $\phi(o_t)$, where $\phi : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$ is a transformation function. For given $\mathcal{D}$, θ_{old} and ϕ, we then train a network θ to minimize the following distillation loss function:

$$L_{\text{DA}}(\theta) = L_{\text{PD}}(\theta) + L_{\text{VD}}(\theta) \ , \tag{4}$$

where L_{PD} is Kullback–Leibler divergence between policies $\pi_{\theta_{\text{old}}}$ and π_θ and L_{VD} is the mean-squared deviation between value functions $V_{\theta_{\text{old}}}$ and V_θ, i.e.,

$$L_{\text{PD}}(\theta) = \hat{\mathbb{E}}_{o_t\sim\mathcal{D}}\left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|o_t), \pi_\theta(\cdot|o'_t)]\right] \ , \tag{5}$$

$$L_{\text{VD}}(\theta) = \hat{\mathbb{E}}_{o_t\sim\mathcal{D}}\left[(V_{\theta_{\text{old}}}(o_t) - V_\theta(o'_t))^2\right] \ . \tag{6}$$

Here o'_t is either the original observation o_t or the augmented one $\phi(o_t)$ with equal probability. The proposed method not only matches the outputs of θ for o_t and $\phi(o_t)$ and also conserves the behavior of θ for o_t to be identical to that of θ_{old} for o_t. This behavior can reduce the interference between reinforcement learning and distillation. Indeed, the performance of RL training can be degraded by distillation without careful consideration on the interference; e.g., [22] with distillation loss of $\hat{\mathbb{E}}_{o_t\sim\mathcal{D}}\left[\text{KL}[\pi_\theta(\cdot|o_t), \pi_\theta(\cdot|\phi(o_t))]\right]$ and $\hat{\mathbb{E}}_{o_t\sim\mathcal{D}}\left[(V_\theta(o_t) - V_\theta(\phi(o_t)))^2\right]$ can change the behavior learned in RL training. Separating distillation from RL training provides substantial performance gain (Table 1) compared to other existing methods e.g., [22, 11] (Section 4.1). In addition, the target behaviors ($\pi_{\theta_{\text{old}}}(\cdot|o_t)$ and $V_{\theta_{\text{old}}}(o_t)$), which are used several times during the distillation, are fixed in DA, so we can reduce the computational cost by pre-computing them.

3.2 Intra Distillation with Augmented observations

InDA (Algorithm 1), iteratively optimizes PPO and DA, with PPO and DA explicitly separated. Such a separation reduces their interference [11], whereas they are often optimized simultaneously in other methods [22]. This separation increases the robustness of our algorithm, in addition to the conservative distillation loss functions in (5) and (6). We varied the timing of augmentation by adjusting the time S of starting DA and time T of terminating DA. We can control the frequency and timing of applying distillation with hyperparameters I, S' and T', where we perform DA after each I rounds of RL training only if the number RL training rounds n is in the interval of $[S', T']$ (or equivalently, the number of timesteps that have been observed is in the range of $[S, T]$). We provide further details on InDA in supplementary material.

3.3 Extra Distillation with Augmented observations

ExDA (Algorithm 2) performs the distillation after the end of RL training, where the lengths of DA and RL training are parameterized by M and N, respectively. Computational cost can be reduced by replacing L_{DA} with L_{PD} in DA, because value function is not necessary after DA. We check empirically that this reduction does not degrade RL performance. We also consider re-initialization after pre-training, because we expect that diminishing of non-stationarity can improve generalization, as mentioned in [14]. However, training performance is not preserved after re-initialization because $\pi_{\theta_{old}}$ is not completely distilled by low data diversity. Thus, we do not use re-initialization for DA. We leave more interesting details in the supplementary material.

Algorithm 1 InDA

Hyperparameter: N, I, ϕ and (S', T') in rounds (or (S, T) in time steps)
Initialize θ close to origin.
for $n = 1, 2, \ldots, N$ **do**
  // RL training
  Store sampled transitions to $\mathcal{D}$;
  Optimize RL objective $L_{\text{PPO}}(\theta)$ with $\mathcal{D}$;
  // Distillation
  if $n \in [S', T']$ and $\text{mod}(n-1, I) = 0$ **then**
    Store $\theta_{\text{old}} \leftarrow \theta$;
    Minimize $L_{\text{DA}}(\theta)$ for $\mathcal{D}$, θ_{old} and ϕ;
  end if
end for

Algorithm 2 ExDA

Hyperparameter: N, M, ϕ
Initialize θ close to origin.
//Pre-training phase with RL algorithm
for $n = 1, 2, \ldots, N$ **do**
  Store sampled transitions to $\mathcal{D}$;
  Optimize RL objective $L_{\text{PPO}}(\theta)$ with $\mathcal{D}$;
end for
Store $\theta_{\text{old}} \leftarrow \theta$;
// Distillation at the end of RL training
for $m = 1, 2, \ldots, M$ **do**
  Minimize $L_{\text{DA}}(\theta)$ for $\mathcal{D}$, θ_{old} and ϕ;
end for

3.4 Auto Augmentation Discriminator

The training benefit by augmentation differs according to the task, and this dependency complicates the choice of whether to use InDA or ExDA for augmentation. We were inspired by an auto-augmentation method, UCB-DrAC [22] that uses an upper confidence bound [3] (UCB). We make a small change to apply the auto augmentation method as a discriminator to check the necessity of augmentation during training. We combine the InDA instead of DrAC with UCB, and call the method UCB-InDA. Also, we add an identity function, which returns the image without a transformation, to the set of image transformations. For a clear explanation, we describe the auto augmentation method as a multi-armed bandit problem. The action space is the set of image transformations $\Phi = \{\phi_1, \ldots, \phi_k\}$ which contains the identity function. The gain of the augmentation $G(s)$ at the sth sampling is the average return during Interval I. The return is the sum of estimated advantage $\hat{A}$ and predicted value V_θ. The general UCB algorithm uses a mean of rewards from the entire sampling process, but in RL, the distribution of return is non-stationary [22], so we use the window-average gain $\bar{G}_\phi(s)$ as a reward of each transformation ϕ. The UCB [3] selects actions each time using the sum of a window average gain $\bar{G}_\phi(s)$ and a degree of exploration:

$$\phi_t = \arg\max_{\phi \in \Phi} \left[\bar{G}_\phi(s) + c\sqrt{\frac{\log(s)}{N_\phi(s)}} \right] \tag{7}$$

where c is the UCB exploration coefficient and $N_\phi(s)$ is the selected number of each augmentation after the sth sample. We find c with in adaptive manner, because the appropriate c is different for each training as a result of drastic change of return during the transient time (details in supplementary material).

4 Experiment

Setups. We evaluate the time-sensitivity of applying augmentation on the OpenAI Procgen benchmark of 16 games, [5], where at each time t, visual observation o_t is given as an image of size 64×64, and contains full or partial information on the system state. A training or testing environment is defined as a pair of game and mode, where mode determines a set of levels and backgrounds shown in the environment. As the training environment, we use one of two modes: *easy* and *easybg*. Easy mode provided by Cobbe *et al.* [5] contains a set of 200 levels, where an agent can learn basic dynamics of game and experience various backgrounds. To see clear advantage from visual augmentation, we further easicate easy mode and devise easybg mode of which only difference from easy mode is showing only a single background. To evaluate generalization capabilities, we use two modes: *test-bg* and *test-lv*, which contain unseen backgrounds and levels, respectively, in addition to the mode that we use for training. The details of modes in our evaluation is provided in supplementary material.

For clarity, we mainly focus on two visual augmentations, each of which has clearly distinguishing inductive bias:

(a) *Random convolution* transforms an image by passing a single convolutional layer initialized randomly [18]. Augmentation with this can impose invariant behavior on color changes, and thus is anticipated to provide strong generalization on background changes.

(b) *Crop* leaves a randomly-selected rectangle and zero-pads the rest of the image [22]. This augmentation is particularly useful in the fully-observable scenarios, because it imposes an efficient attention mechanism.

We also report the result with other visual augmentations including *color jitter, gray and cutout color* in the supplementary material, where the same main messages can be found. All results in the main paper are averages over five runs.

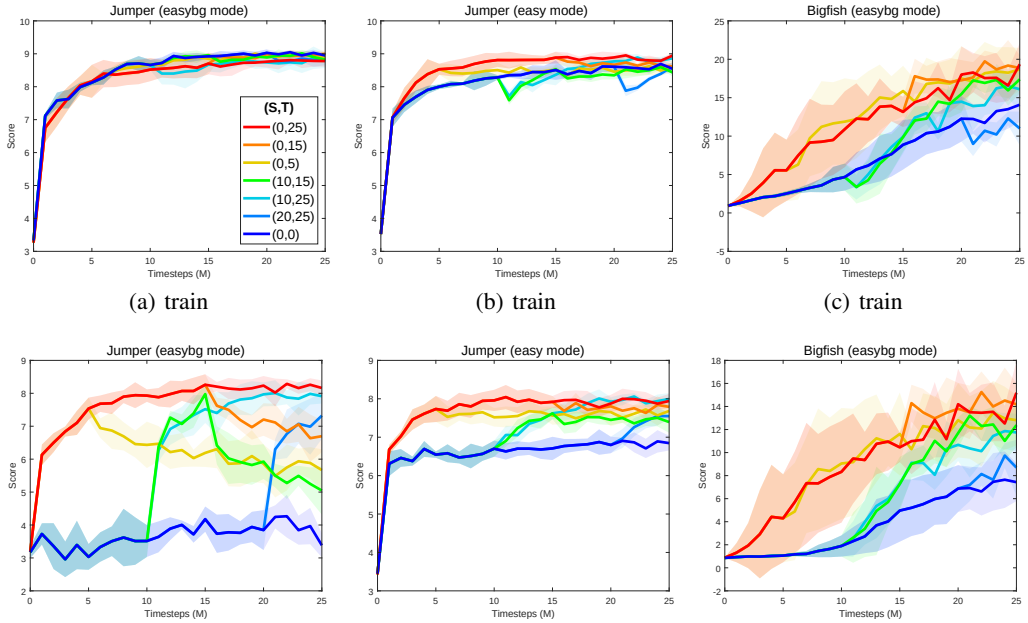
Augmentation		PPO	Oracle	DrAC	RAD	Rand-FM	InDA	ExDA
Rand conv	Train	1.00	0.85	0.88	**0.98**	0.88	0.88	**0.98**
	Test-bg	1.00	2.33	1.86	1.08	1.04	1.92	**2.11**
Color jitter	Train	1.00	0.85	0.95	0.94	-	0.96	**0.98**
	Test-bg	1.00	2.33	1.44	1.37	-	1.43	**1.48**
Grayscale	Train	1.00	0.85	0.93	0.94	-	0.95	**0.99**
	Test-bg	1.00	2.33	1.03	1.04	-	0.97	**1.13**
Cutout color	Train	1.00	0.82	0.82	0.72	-	0.76	**0.94**
	Test-bg	1.00	2.51	1.27	1.33	-	1.19	**1.53**
	Test-lv	1.00	-	0.83	0.69	-	0.69	**0.93**
Crop	Train	1.00	-	1.08	0.28	-	**1.25**	0.91
	Test-lv	1.00	-	1.52	0.46	-	**1.80**	1.09

Table 1: Train and test score of InDA and ExDA on Open AI Procgen, compared to baselines PPO, Oracle, Drac [22], RAD [17], Rand-FM [18]. Oracle is trained with test backgrounds. **Boldface** indicates the best method, and red indicates Oracle. ExDA outperforms other baselines except when we use *crop*, which is evaluated on unseen levels.

4.1 Improving generalization on Procgen

We compare the train and test performances of InDA and ExDA with those of several baselines (Table 1). Every method are trained on 200 levels, using *easybg* mode, which contains a single background. InDA and other baselines are trained for 25M time steps. ExDA is trained with PPO for 20M time steps, then distills for 30 epochs with 0.5M time steps, after training with PPO for 20M time steps. *Test-bg* is used for *random convolution, color jitter, grayscale* and *cutout color*, which give information of color diversity. *Test-lv* is used for *cutout color* and *crop*, which give a consistency of partial observation. Further details about implementation and hyperparameter are described in the supplementary material. ExDA outperforms other baselines with most augmentation

on *test-bg*. Especially, ExDA with *random convolution* has a comparable performance to Oracle, despite being trained on a single background. Moreover, ExDA consumes only 0.5M time steps to inject knowledge from augmented images, whereas the others use all of the training data. We describe the computation issue about both ExDA and InDA in the supplementary material. However, InDA have a better train and test performance with *crop* on *test-lv*. These results demonstrate that each combination of environment and augmentation has a suitable time at which to apply augmentation. Thus, we analyze the proper condition to use InDA or ExDA in the next section.



(a) train (b) train (c) train

(d) test-bg (e) test-bg (f) test-lv

Figure 2: Comparison according to timing of augmentation with InDA: (S, T) are hyperparameters of InDA for the start and terminal time step of distillation. As an augmentation method, InDA uses random convolution on Jumper, and uses crop on Bigfish. Test results are evaluated on test-bg (unseen backgrounds) and test-lv (unseen levels). Shaded region: standard deviation of five runs. The difference between Jumper (easybg) and Jumper (easy) shows that the benefits of data efficiency and the maintenance of generalization can be changed by the diversity of factors in observations. Augmentation can accelerate the training, such as Bigfish. Furthermore, delayed augmentation commonly improves generalization as much as fully-used cases.

4.2 Time dependency of augmentation in RL

In this section, we study *when* and *how* the agent is particularly helped by augmentation during RL training. For this purpose, we varied DA time (S, T) of InDA to see how generalization's effect depends on the time at which augmentation is used. We evaluate InDA with seven different pairs of start and terminal time of distillation (S, T), where the number of entire timesteps used is 25M: (0, 25), (0, 5), (0, 15), (10, 0), (20, 0), (10, 15), (0, 0). Note that InDA with (S, T) = (0, 0) means RL training only without augmentation, i.e., vanilla PPO. We explain with Jumper and Bigfish in the main paper, and experiments on other environments are described in the supplementary material. Figure 2 represents three environments, Jumper with easybg (Figure 2(a), 2(d)) and easy (Figure 2(b), 2(e)) mode and Bigfish (Figure 2(c), 2(f)) with easybg mode. In the following, we call the curve using a parameter (S,T).

Interrupted augmentation . To determine how generalization would change after regularization stopped, we stop the DA during training, such as (0, 5), (0, 15). Jumper with easybg mode rapidly lost generalization performance (after interruption at both (0, 5) and (0, 15)) (Figure 2(d)), whereas Jumper with easy mode do not (Figure 2(e)). InDA, which uses augmentation throughout training, performs better than PPO during training (Figure 2(b)), but augmentation does not improve the

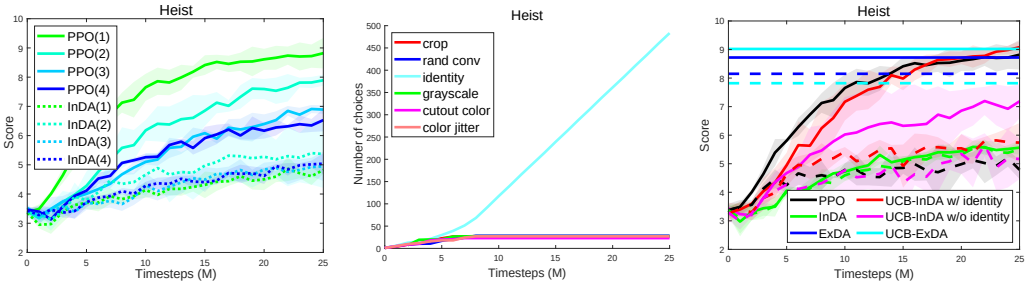
training performance (Figure 2(a)). These results mean that the random convolution alleviates the difficulty by various backgrounds.

On the contrary, random convolution can induce a growing difficulty by increasing the number of factors on a single background. Therefore, the generalization rapidly decreases after augmentation is interrupted during training with a single background because the learning direction toward generalization about various backgrounds is not helpful to train. In contrast, the training can help when their difficulty is solved by augmentation (Figure 2(b), 2(c)). Thus, in deep RL, neural networks maintain the regularization when augmentation helps the training.

Regularization biases toward regions of loss landscape can have several equivalent generalized solutions [9]. For the same reason, augmentation regularizes a neural network model by imposing a bias toward generalization in deep RL. Moreover, in Bigfish, (0, 5) and (0, 15) increase the training performance and generalization, similar to (0, 25), although they use augmented observations only for a while. Therefore, the augmentation may not be necessary during the whole training process in some tasks.

Delayed augmentation . To determine when we start to use augmentation, we delayed its use until after 10M or 20M steps. The generalization rapidly increases after using augmentation at 10M and 20M (Figre 2(d), 2(e)). Although we impose augmentation late, the augmentation helps the generalization regardless of the start timing. In SL, delayed augmentation cannot achieve as much as using augmentation during whole training [9]. However, (10, 25) improves the generalization to be comparable with that of (0, 25), which use augmentation throughout training; this result differs from the case of supervised learning. However, when augmentation noticeably helps the training, the performance achieved using delayed augmentation may not catch up (Figure 2(e)) to the performance achieved using early augmentation (Figure 2(f)), because the RL gradually improves the policy and trajectory, as a result of its Markov property. Furthermore, the number of samples is limited for RL, but not for supervised learning, so using augmentation from the initial time is more critical than supervised learning if augmentation helps the training.

However, we confirm that delayed augmentation can induce bias toward generalization after the interruption, although the augmentation is not used during the initial transient time. For example, curves (10, 15) (Figure 2(e), 2(f)) equivalent results to those of with (10, 25) even after 15M time steps. This result indicates that a bias toward generalization can occur regardless of the timing of augmentation, but usage from the start is essential when the augmentation gives important knowledge during training.



(a) Background diversity (b) Augmentation selection (c) Performance

Figure 3: Comparison of train performance according to background diversity on Heist *easybg* when InDA use random convolution in 3(a). We describe the degree of background diversity in numbers, e.g., PPO(1). As the diversity of the background increases, the performance of the PPO approaches that of InDA. Figure 3(b) show the selected number of augmentations by UCB on Heist. We compare two UCB-InDAs, w/ and w/o identity function with PPO, InDA, ExDA, UCB-ExDA on Heist *easybg* in Figure 3(c). UCB-InDA is trained after UCB-InDA w/ identity, we use *random convolution* as a data augmentation in InDA, ExDA. Solid line: train performance; dotted line: test performance. ExDA achieves larger test performance than InDA by preserving train performance. Moreover, UCB-InDA w/ identity outperforms UCB-InDA w/o identity in the training.

When ExDA should be used. ExDA have different response than other methods to the timing of applying augmentation. Most methods combined with data augmentation in RL use augmentation throughout RL training. We show that augmentation rapidly increases the generalization performance in spite of the late usage of augmentation (Figure 2). These results motivate suggest that generalization by augmentation after training in RL.

The performance of PPO approached that of InDA as the diversity of the background increased (Figure 3(a)). This result means that the various backgrounds increase the difficulty of training, and also that random convolution has a similar effect to diverse background images. In contrast, ExDA preserves its training score after pure RL training. These gaps in training are quantified by the test performance.

The two analyses suggest that ExDA is appropriate in environments in which training is difficult as a result of various factors such as background, and object color in the image. Thus, for generalization, ExDA should be used when only one background is present.

When InDA should be used. ExDA does not always guarantee improvement of generalization. InDA is better generalized to unseen levels with *crop* than ExDA is (Table 1). ExDA cannot surpass InDA in some environments, for two reasons.

First, the diversity of data has an important ability to generalize about unseen levels [5]. InDA is trained with various observations during training, whereas ExDA applies the augmentation only on a pre-trained policy's trajectories. Thus, augmentation after training may have difficulty overcoming the limitation of data diversity when the generalization needs the diversity of data distribution. Second, InDA can accelerate the training such as 2(c), so ExDA cannot overcome the gap in training performance. Thus, InDA should be used in both cases.

How to choose between InDA and ExDA. InDA is appropriate when the augmentation methods help to train, and ExDA is appropriate when the augmentation methods increase the difficulty of training. However, we cannot know in advance whether certain augmentation helps the training. Thus, we use UCB-InDA to automatically determine the necessity of the augmentation during training. UCB selects the identity function most often (Figure 3(b)). This means that other augmentations are not helpful to train on Heist(*easybg*), so ExDA is more appropriate than InDA. Furthermore, UCB-InDA w/ identity performs better than w/o identity. It suggests that identity should be included in the UCB action, because of the environments do not need augmentation during training. In contrast, PPO is the same as InDA with identity, which is the best transformation on Heist(*easybg*). Thus, we can use UCB-InDA as pre-trained method for ExDA, e.g., UCB-ExDA, because UCB-InDA w/ identity achieves comparable train performance to that of PPO (Figure 3(c)). As the result, we can automatically select InDA or ExDA appropriately for each task.

5 Discussion

We have showed that the timing of visual augmentation affects the performance of RL, although not affect the performance of SL. The difference is a result of non-stationary data generation in RL. If the regularization imposed by augmentation is useful only for testing, then augmentation should be delayed to the end of RL training than being use throughout learning, because sample and computation complexity since it can disturb RL training. However, an augmentation that provides useful regularization in training should be used during the whole training period to fully utilize its benefit in terms of both generalization and data efficiency. We believe that our findings provide useful insights into auto-augmentation to adjust the use of augmentation round-by-round, where DA at the end of RL training would provide substantial gains as ExDA does. However, design of auto-augmentation for RL remains an open problem, because the gain from augmentation has highly non-stationary characteristics, so its evaluation is challenging.

Acknowledgments and Disclosure of Funding

This work was supported by Institute of Information and communications Technology Planning and evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, AI Innovation Hub). This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021M3E5D2A01023887). The authors would like to thank Kimin Lee and Alexandre Proutiere for fruitful feedback.

References

[1] A. Achille, M. Rovere, and S. Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[2] S. Arora, N. Cohen, W. Hu, and Y. Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7411–7422, 2019.

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

[4] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning, 2019.

[5] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[6] D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[7] J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in dqn, 2020.

[8] G. Gidel, F. Bach, and S. Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In *Advances in Neural Information Processing Systems*, pages 3196–3206, 2019.

[9] A. S. Golatkar, A. Achille, and S. Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *Advances in Neural Information Processing Systems*, pages 10678–10688, 2019.

[10] S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[11] N. Hansen and X. Wang. Generalization in reinforcement learning by soft data augmentation. *arXiv preprint arXiv:2011.13389*, 2020.

[12] N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[13] I. Higgins, A. Pal, A. A. Rusu, L. Matthey, C. P. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. Darla: Improving zero-shot transfer in reinforcement learning, 2018.

[14] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

[15] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

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

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

[18] K. Lee, K. Lee, J. Shin, and H. Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv*, pages arXiv–1910, 2019.

[19] Z. Liu, X. Li, B. Kang, and T. Darrell. Regularization matters in policy optimization – an empirical study on continuous control, 2020.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[21] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[22] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

[23] S. C. Raparthy, B. Mehta, F. Golemo, and L. Paull. Generating automatic curricula via self-supervised active domain randomization, 2020.

[24] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[25] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550 (7676):354–359, 2017.

[26] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[27] A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

[28] A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

[29] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[30] X. Wu, E. Dyer, and B. Neyshabur. When do curricula work?, 2020.

[31] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images, 2020.

[32] K. You, M. Long, J. Wang, and M. I. Jordan. How does learning rate decay help modern neural networks?, 2019.