JOINT SELF-SUPERVISED LEARNING FOR VISION-BASED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Vision-based reinforcement learning requires efficient and robust representations of image-based observations, especially when the images contain distracting (task-irrelevant) elements such as shadows, clouds, and light. It becomes even more difficult if those distractions are not exposed during training. Although several recent studies have shown that generalization can be improved, they still suffer from relatively low performance even in simple and static backgrounds. To enhance the quality of representation, we design an RL framework that combines two-way (weak and strong) data augmentations and three self-supervised learning methods; Adversarial Representation, Forward Dynamics, and Inverse Dynamics. For a set of continuous control tasks on the DeepMind Control suite, our joint self-supervised RL (JS2RL) efficiently learns the task control in both simple and distracting backgrounds, and significantly improves generalization performance for unseen backgrounds. In an autonomous driving task, CARLA, our JS2RL also achieved the best performance on complex and realistic observations containing a lot of task-irrelevant information.

1 INTRODUCTION

Vision-based reinforcement learning (RL) (Mnih et al., 2013; Nair et al., 2018; Hafner et al., 2019a) has been studied to learn optimal control using high dimensional image inputs. The demand for vision-based RL has continued to grow as more attempts are made to apply RL to real-world applications such as robotics and autonomous driving, which primarily use image data. However, to achieve this, vision-based RL must address two fundamental problems; data efficiency and generalization. Data efficiency refers to how quickly optimal control of a task can be learned using fewer experience samples. Learning control from high dimensional images such as raw pixels inevitably increases the learning difficulty. In particular, if the images contain task-irrelevant information (clouds, shadows, and light etc.), this unnecessary information interferes with learning optimal control. The more complex the observation, the worse this problem is. Therefore, representation learning has become more important for extracting meaningful features from high dimensional observations. In terms of generalization, task-irrelevant information may vary depending on the time and location of the actual tests. If those distracting elements are not exposed during training, control performance could be severely degraded. Some prior works present that using relatively weak data augmentations can improve data efficiency rather than using strong augmentations (Srinivas et al., 2020)(Kostrikov et al., 2020). However, we found that they could NOT work well if the backgrounds at testing time differ from the backgrounds at training time as shown in Table 1. Instead, some studies suggest bisimulation-metric based representation learning without augmentations, and show the method can be helpful to capture task-relevant elements in complex observations. Clearly the improved representations become robust and invariant against distracting elements. However, they still suffer with relatively low performances, even on simple and static backgrounds.

Our key insight is that representation learning needs to be designed from multiple perspectives to increase data efficiency and generalization performance together. In this paper, we first propose two-way data augmentations. For the same observation, we provide weak and strong data augmented versions because weak augmentations are helpful to increase data efficiency, and strong augmentations are necessary to improve generalization power. Second, we design joint self-supervised learning to effectively extract task-relevant features from the two-way augmentations. We incorporate three self-supervised learning methods; Adversarial Representation, Inverse Dynamics, and

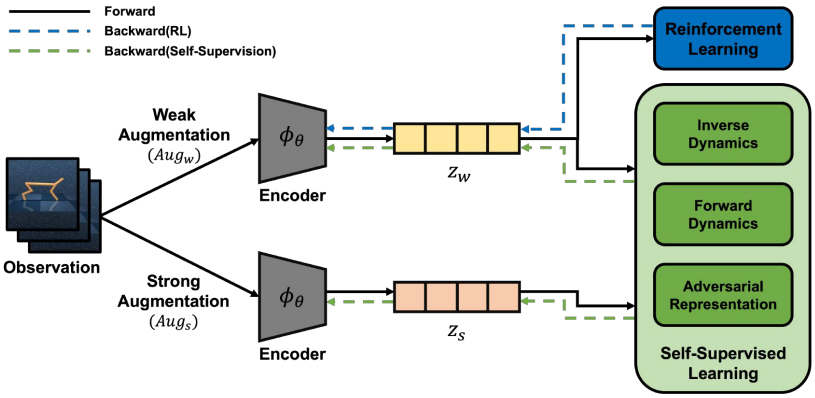


Figure 1: Our Framework Overview: we use a shared encoder for RL and joint self-supervised learning. An observation is augmented in two ways; Aug_w uses Random-Shift only, and Aug_s uses Random-Shift and other randomly chosen augmentation method. The encoded latent state z_w is used to train an RL algorithm, and both z_w and z_s are passed to joint self-supervised learning which consists of Forward Dynamics, Inverse Dynamics, and Adversarial Representation.

Forward Dynamics. In Adversarial Representation, a discriminator learns to distinguish between weak and strong augmented versions, while an encoder learns to fool the Discriminator. This allows the encoder to capture invariant features in two-way augmented versions. Inverse Dynamics infers actions between successive (latent) states for each weak and strong augmentation way. The inferred actions are used to predict the identical future states from (two-way augmented) current states by Forward Dynamics. In our ablation tests, the effectiveness of the individual self-supervised learning method was insignificant, but the synergy greatly increased when they were integrated together over two-way augmentations.

For a set of continuous control tasks (the DeepMind Control suite (Tassa et al., 2018)), we add distracting elements into the backgrounds as proposed in (Zhang et al., 2020). Compared to prior studies, our joint self-supervised RL (JS2RL) is substantially more efficient in both simple and distracting backgrounds. We also showed that JS2RL significantly outperformed existing studies when the testing backgrounds differed from the training backgrounds, which means our approach achieved a higher generalization ability. In an autonomous driving task, CARLA (Dosovitskiy et al., 2017), our method also achieved best performance on complex observations containing a lot of task-irrelevant information in realistic driving scenes.

2 Related Works

Data efficiency of learning with fewer amount of data and generalization to unseen environments are major challenges for reinforcement learning, especially in vision-based applications. Many studies have been conducted on improving data efficiency and generalization, and they mainly use data augmentation techniques and self-supervised learning methods.

2.1 Improving Data Efficiency

Model-based RL, one of the branches of RL, learns a world model such as Forward Dynamics which uses to future rollouts and plan to increase data efficiency (Ha & Schmidhuber, 2018; Hafner et al., 2019b;a). Model-free RL algorithms have been also studied to improve data efficiency. Reconstructing the current observation (Jaderberg et al., 2016; Yarats et al., 2019; Lee et al., 2019a) helps to extract compact representations. Predicting the future observation (Jaderberg et al., 2016) or latent state (Oord et al., 2018) are more effective in obtaining a good representation. Several RL studies proposed to use data augmentations which provide multi-views of the data (Laskin et al., 2020; Kostrikov et al., 2020; Srinivas et al., 2020). RAD (Laskin et al., 2020) showed that using data augmentations improves data efficiency without modifying RL algorithms. DrQ (Kostrikov et al., 2020) improved data efficiency using both augmentation methods and modified Q-functions.

CURL (Srinivas et al., 2020) combined data augmentations and contrastive learning (Chen et al., 2020; Henaff, 2020; He et al., 2020) to learn representation more efficiently. Most prior studies select one augmentation technique that provides the best data efficiency for each environment, and they are NOT working well in unseen backgrounds. Our work suggests two-way (weak and strong) data augmentations, and can improve generalization performance while achieving high data efficiency.

2.2 Improving Generalization

In vision-based applications, image observations may contain lots of task-irrelevant information such as clouds, shadows and light. It becomes more challenging when these distracting elements are constantly changing during the time of testing. Therefore, learning invariant features that are directly relevant to the task control is a high priority for generalization. Recent studies attempt to increase generalization with supervised learning, including regularization (Cobbe et al., 2019; Farebrother et al., 2018), stochasticity (Zhang et al., 2018b), noise injection (Igl et al., 2019; Zhang et al., 2018a), more diverse training conditions (Rajeswaran et al., 2017; Witty et al., 2018) and self-attention architectures (Tang et al., 2020). DeepMDP (Gelada et al., 2019) and DBC (Zhang et al., 2020) used bisimulation metrics to provide effective downstream control by learning invariant features from the images including task-irrelevant details. Although these studies improved generalization power, there was a decrease in data efficiency. SODA (Hansen & Wang, 2021) learns representation by maximizing the mutual information between augmented data and non-augmented data. SECANT (Fan et al., 2021) first learns an expert policy with weak augmentations, and imitates the expert policy with strong augmentations. Inverse Dynamics was introduced in (Pathak et al., 2017; Burda et al., 2018) to provide intrinsic rewards for exploration. PAD (Hansen et al., 2020) proposed Inverse Dynamics to fine-tune representation during deployment. Our work does NOT require any pre-training or fine-tuning at testing, and achieves the best generalization performance and data efficiency.

2.3 Adversarial Learning

Adversarial learning (Goodfellow et al., 2014; Miyato et al., 2016; Kurakin et al., 2016) has been leveraged in several RL studies. GAIL (Ho & Ermon, 2016) proposed an imitation learning method that lets a discriminator distinguish between expert trajectories and those of a generator; here, the generator tries to match expert behavior to fool the discriminator, like the concept of GAN (Goodfellow et al., 2014). In inverse RL, AIRL (Fu et al., 2017) uses adversarial reward learning to be robust to changes in dynamics. AGAC (Flet-Berliac et al., 2021) is based on typical Actor-Critic algorithm, but introduces an adversary component. While the adversary tries to imitate the actor, the actor tries to solve the task and behaves differently from the adversary. It showed excellent performance in various hard-exploration environments. In our study, we proposes an adversarial representation method to improve data efficiency and generalization without modifying the RL algorithm.

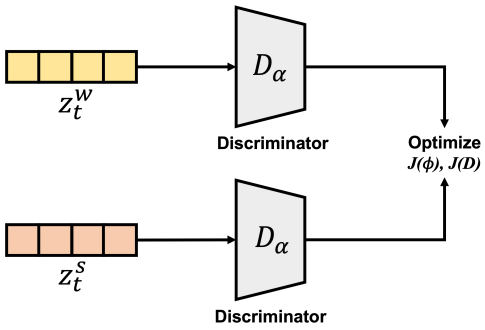
3 Joint Self-Supervised RL (JS2RL)

In this section, we introduce our reinforcement learning framework using the joint self-supervised learning method. Our framework doesn't require any changes to the underlying RL algorithm, and any RL algorithm can be used.

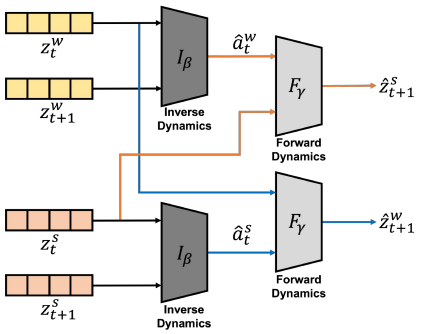
3.1 Model Overview

We design the model architecture to share represented features that feed into reinforcement learning and joint self-supervised learning. We define encoder ϕ, Inverse Dynamics I, Forward Dynamics F, Discriminator D. Our goal is to train the encoder ϕ to extract task-control relevant information efficiently so that the RL agent can learn the generalized optimal policy. Our encoder ϕ is updated by gradients from Inverse Dynamics I, Forward Dynamics F, Adversarial Representation and RL algorithm. (Our framework is illustrated by Figure 1 and Algorithm 1)

3.2 Two-way Data Augmentations



(a) Adversarial Representation.



(b) Inverse Dynamics and Forward Dynamics.

Figure 3: (a) Adversarial Representation learns invariant features between weak and strong augmented views of the same observation. (b) Inverse Dynamics predicts an action $\hat{a}_t$ given the current latent state z_t and the next latent state z_{t+1}. With two-way augmentations, we can predict $\hat{a}_t^w$ and $\hat{a}_t^s$ respectively. Forward Dynamics predicts the next latent state $\hat{z}_{t+1}$ given the current latent state z_t and the action $\hat{a}_t$ predicted by Inverse Dynamics. We cross-enter the predicted actions $\hat{a}_t^w$ and $\hat{a}_t^s$ into Forward Dynamics.

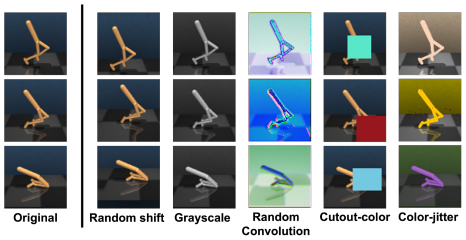


Figure 2: Data augmentations used in our framework.

In our framework, as shown in Figure 2, we use multiple data augmentation techniques together during training; Random-shift (Kostrikov et al., 2020), Grayscale, Random Convolution (Lee et al., 2019b), Color-jitter and Cutout-color (Cobbe et al., 2019). Random-shift pads each side and then selects a random crop back to the original image size. Grayscale converts RGB images to grayscale images based on certain probabilities. Random convolution transforms an image through a randomly initialized convolutional layer. Color-jitter converts RGB image to HSV image which adds noise to each channel of HSV. Cutout-color randomly inserts a small random color occlusion into the input image.

We design two-way data augmentations of Aug_w and Aug_s shown in Figure 1. Aug_w means weak augmentation by using only Random-shift, and Aug_s represents strong augmentation which consists of Random-shift plus a randomly chosen technique other than the Random-shift. While our weak augmentation focuses on position transformation, the strong augmentation includes position, color, and texture transformation. Aug_w, Aug_s are applied randomly across the batch and the parameters of these augmentations applied to current and next observations are also different.

3.3 Adversarial Representation

The goal of Adversarial Representation is to allow the encoder to extract robust representations that are invariant between weak and strong augmented views. The augmented observations by Aug_w and Aug_s pass through the encoder ϕ to obtain a latent state $z_w = \phi(Aug_w(obs))$, $z_s = \phi(Aug_s(obs))$, where obs represents an image observation. Instead of using a traditional GAN, we use a relativistic GAN (Jolicoeur-Martineau, 2018), which is known to be more stable and faster. For z_w and z_s, we define encoder (as a generator) and Discriminator objective functions as follows, where σ represents a sigmoid function.

$$J(\phi) = -\log(\sigma(D(z_s) - D(z_w))), \tag{1}$$

$$J(D) = -\log(\sigma(D(z_w) - D(z_s))). \tag{2}$$

$J(\phi)$ optimizes z_s to have a higher value than z_w in Equation 1. Conversely, $J(D)$ optimizes z_w to have a higher value than z_s in Equation 2. By alternately optimizing Equation 1 and Equation 2, the encoder ϕ is updated so that the representations of z_w and z_s become similar as illustrated by Figure 3a. Eventually, Adversarial Representation can learn invariant features regardless of the position shifts and the changes in color and texture of the observations.

3.4 INVERSE DYNAMICS AND FORWARD DYNAMICS

We introduce Inverse Dynamics and Forward Dynamics based on two-way data augmentations. For given sequential latent states z_t^w and z_{t+1}^w, and another pair of z_t^s and z_{t+1}^s, Inverse Dynamics I infers the actions $\hat{a}_t^w = I(z_t^w, z_{t+1}^w)$ and $\hat{a}_t^s = I(z_t^s, z_{t+1}^s)$. Even if the input image is augmented in different ways, the two inferred actions must be identical to each other, and must be identical to the action a_t that was actually performed.

The inferred actions $\hat{a}_t^w$ and $\hat{a}_t^s$ are fed into Forward Dynamics F along with the current latent states z_t^w and z_t^s. F predicts the next latent states $\hat{z}_{t+1}^w$ and $\hat{z}_{t+1}^s$ as shown in Figure 3b. It is important to note that the inferred actions are cross-entered such as $\hat{z}_{t+1}^w = F(z_t^w, \hat{a}_t^s)$ and $\hat{z}_{t+1}^s = F(z_t^s, \hat{a}_t^w)$. Literally, z_{t+1}^w is predicted by using a_t^s, inferred from the pair of z_t^s and z_{t+1}^s, to which different augmentations were applied. This cross-entered prediction allows our encoder to learn more robust representation by using the inferred knowledge from different perspectives on the same observation.

The Inverse Dynamics objective function Equation 3 is defined as the mean squared error between actual action and inferred action.

$$J(I) = \frac{(I(z_t^w, z_{t+1}^w) - a_t)^2 + (I(z_t^s, z_{t+1}^s) - a_t)^2)}{2} \tag{3}$$

The Forward Dynamics objective function Equation 4 is defined as negative cosine similarity Δ between the predicted next latent state and the actual next latent state that encodes the next observation.

$$J(F) = \frac{\Delta(\hat{z}_{t+1}^w, z_{t+1}^w) + \Delta(\hat{z}_{t+1}^s, z_{t+1}^s)}{2} \tag{4}$$

The final joint self-supervised objective function is defined as a combination of Inverse Dynamics, Forward Dynamics and Adversarial Representation as shown in Equation 5, and it can send a training signal to the encoder ϕ to efficiently represent task-relevant features.

$$J(I, F, \phi, D) = \lambda_I J(I) + \lambda_F J(F) + \lambda_A J(\phi, D) \tag{5}$$

where λ_I, λ_F and λ_A are hyper parameters.

Algorithm 1 Joint Self Supervised Reinforcement Learning

Initialize: Encoder ϕ, Policy π, Critic Q,
Discriminator D, Inverse Dynamics I, Forward Dynamics F, Buffer B.

for each iteration **do**
  for each environment step **do**
    Encode state $z_t = \phi(s_t)$ ▷ $s_t \leftarrow env.step(a_{t-1})$
    Execute action $a_t = \pi(z_t)$
    Store transition: $B \leftarrow B \cup \{s_t, a_t, s_{t+1}, r_{t+1}\}$
  end for
  for each update step **do**
    Sample mini-batch: $(S, A, S', R) \sim B$
    Apply augmentation: $Z_w, Z'_w = Aug_w(S), Aug_w(S')$
    Apply augmentation: $Z_s, Z'_s = Aug_s(S), Aug_s(S')$
    Train joint self-supervision: $E_{Z_w, Z'_w, Z_s, Z'_s, A}[J(I, F, \phi, D)]$
    Train RL policy: $E_{Z_w, Z'_w}[J(\pi)]$
  end for
end for
return Optimal Policy π

Algorithm 1 describes how the joint self-supervised RL works. In the algorithm, s_t, s_{t+1} are the image observations obtained by interacting with the environment. We divide the training phase of JS2RL into two steps. First, train the three self-supervised learning methods (Inverse Dynamics, Forward Dynamics and Adversarial Representation), and then train the RL policy. We repeat this learning process and JS2RL objective functions refer to Equation 5. This algorithm version is based on an off-policy RL algorithmm, such as Soft Actor-Critic (SAC) (Haarnoja et al., 2018), but our JS2RL can work with any RL algorithms, as shown in Appendix E (such as on-policy algorithms like PPO (Schulman et al., 2017) and other off-policy algorithms like TD3 (Fujimoto et al., 2018)).

4 EXPERIMENTS

This section demonstrates how efficiently JS2RL can learn tasks with distracting elements (task-irrelevant information) and can generalize well against unseen test environments. On a set of continuous control tasks in the DeepMind Control suite, JS2RL shows excellent performance in most settings. For CARLA (Dosovitskiy et al., 2017), a more realistic and autonomous driving environment with various distractors (e.g. shadows, changing weather, and light), we also show better performance than prior studies. We benchmark JS2RL against the following algorithms; SAC is plain Soft Actor-Critic with no augmentation. DrQ (Kostrikov et al., 2020) applies data augmentations in SAC. CURL (Srinivas et al., 2020) introduces a contrastive representation learning method. DeepMDP (Gelada et al., 2019) and DBC (Zhang et al., 2020) learns a latent state representation by predicting rewards and next latent states using the bisimulation metric. SODA (Hansen & Wang, 2021) learns representation by maximizing the mutual information between augmented and non-augmented data. PAD (Hansen et al., 2020) fine-tunes representation at testing environments through self-supervision.

4.1 NETWORK ARCHITECTURE

We implement our joint self-supervised method on top of Soft Actor Critic (SAC) for the visual input version (Yarats et al., 2019) architecture, which updates the encoder only with Q-function back-propagation. The RL parts Actor, Critic and the self-supervised part Inverse Dynamics, Forward Dynamics and Discriminator networks share the Encoder ϕ which consists of 4 convolutional layers and 1 fully connected layer. Both Actor and Critic consists of 3 fully connected layers. Inverse Dynamics I, Forward Dynamics F and Discriminator D consists of 2 fully connected layers. For CARLA environment, we modify the Encoder ϕ slightly. Implementation details and hyper parameters are in Appendix F.

4.2 DEEPMIND CONTROL SUITE

Figure 4: We use three different background types. There are examples on a Cheetah task in the Deepmind Control suite; Default (left), Simple Distractor (center), and Natural Video (right)

The DeepMind Control suite is a vision-based simulator that provides a set of continuous control tasks. We experiment with nine tasks; Walker Walk, Cheetah Run, Hopper Hop and additional tasks in the appendix. And we evaluate the performances on two metrics; one is Data efficiency and the other is Generalization. Each RL method is trained for 500K environment steps, and every 5,000 steps, we tested the currently trained model by calculating the average return for 10 episodes. We trained each RL method over three different seeds. [1] As shown in Table 1, JS2RL shows performance similar to the best performance of the prior works in data efficiency experiments, and significantly outperforms the prior works in generalization experiments. More experiment details are in Appendix F.

4.2.1 DATA EFFICIENCY

For the data efficiency evaluation, we used three background configurations; Default, Simple Distractor and Natural Video, as shown in Figure 4. Default is a clean and static background provided by the DeepMind Control suite. Simple Distractor is a non-stationary background with randomly plotted circles with different colors. Natural Video is also a non-stationary background which consists of real car-driving scenes in Kinetics dataset (Kay et al., 2017). In this evaluation, the test is carried out in the same environment (the same background setup) used for training. As shown in Figure 5, JS2RL almost shows the top performance compared with other baselines. On Hopper Hop, JS2RL achieves **10%** higher than the best performance of the other baselines. The results for the additional task environments and backgrounds are in Appendix A, C.

[1] There is a reproducing issue on DBC as already reported in the author's GitHub, so we received the result logs shown in the DBC paper from the author.

	SAC	DrQ	CURL	SODA	PAD	DeepMDP	DBC	JS2RL (Ours)
Data Efficiency (evaluate on seen backgrounds)								
Walker Walk	37.1±4.5	493.5±105.2	**917.4±12.0**	869.1±12.0	861.7±1.8	537.6±74.2	503.2±56.6	894.0±7.7
Cheetah Run	230.2±20.4	272.8±31.4	335.5±0.3	263.3±8.7	293.0±0.4	**353.2±69.2**	174.4±35.7	298.3±7.8
Hopper Hop	92.4±5.6	91.5±31.8	73.6±27.5	86.4±43.5	129.0±39.2	8.7±12.9	4.1±5.2	**142.0±2.8**
Generalization (evaluate on unseen backgrounds)								
Walker Walk	57.8±16.9	270.5±81.6	407.6±35.0	754.2±25.4	835.3±1.4	329.2±116.9	547.1±9.4	**849.1±45.5**
Cheetah Run	49.0±16.8	218.6±25.2	189.5±31.1	228.7±17.8	256.7±0.5	64.8±8.0	142.7±10.0	**314.3±17.9**
Hopper Hop	14.8±5.2	81.4±30.0	42.7±23.6	59.2±32.0	54.0±10.6	0.6±0.2	27.2±18.6	**121.8±7.8**

Table 1: Performance of JS2RL and baselines on three tasks in the DeepMind Control suite. We train for 500K environment steps on Simple Distractor. We evaluate the trained model on the same Simple Distractor for data efficiency experiments, and evaluate the model on unseen Natural Video for generalization experiments. The results show the mean and standard deviation over three different seeds.

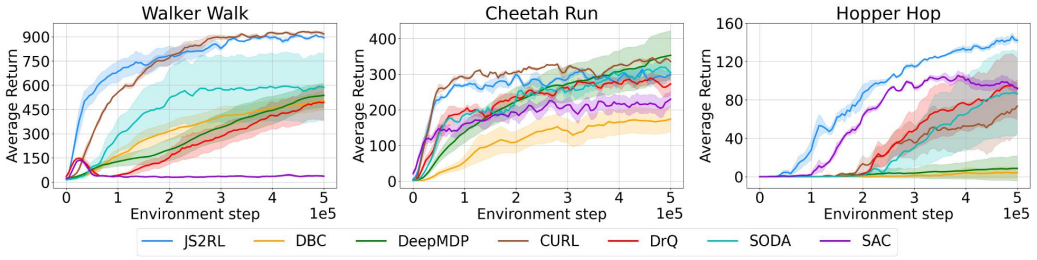


Figure 5: Data efficiency comparisons on three tasks in the DeepMind Control suite; Walker (left columns), Cheetah (center columns), and Hopper (right column). All RL methods are trained and evaluated on the same Simple Distractor backgrounds. We show the learning curves for each RL method on three different seeds with 1.0 standard error shaded.

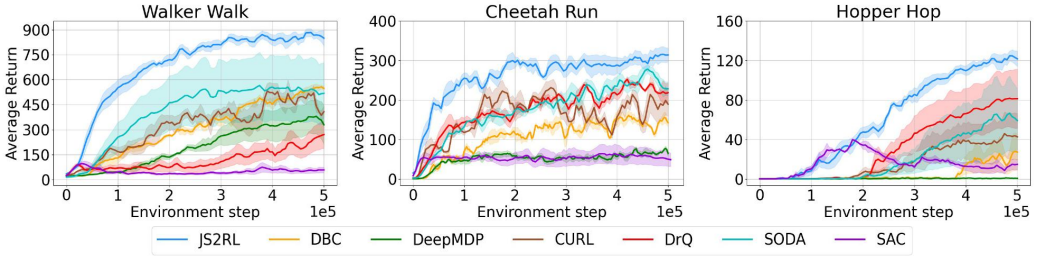


Figure 6: Generalization comparisons on three tasks in the DeepMind Control suite; Walker (left), Cheetah (center), and Hopper (right). Each task is trained on Simple Distractor backgrounds, and then is evaluated on Natural Video backgrounds. We show a comparison of our algorithm and baselines on three different seeds with 1.0 standard error shaded.

4.2.2 GENERALIZATION

In this experiment, we first trained each RL method in the Simple Distractor background and then evaluated it in the Natural Video background, which was not seen during the training phase. Figure 6 presents that JS2RL not only learns each task very quickly and achieves the highest performance, but also generalizes well to the unseen environment. At 500K training steps, we achieve **2%**, **22%** and **50%** higher performances compared to the highest performance among other RL methods for each task. The results for the additional task environments are in Appendix A. In Figure 7, we also visualize the state embedding of Walker Walk using t-SNE. Even if unseen backgrounds are dramatically different, a well-generalized encoder should capture invariant features when observations are behaviorally equivalent. JS2RL can encode semantically similar observations to be most closely located.

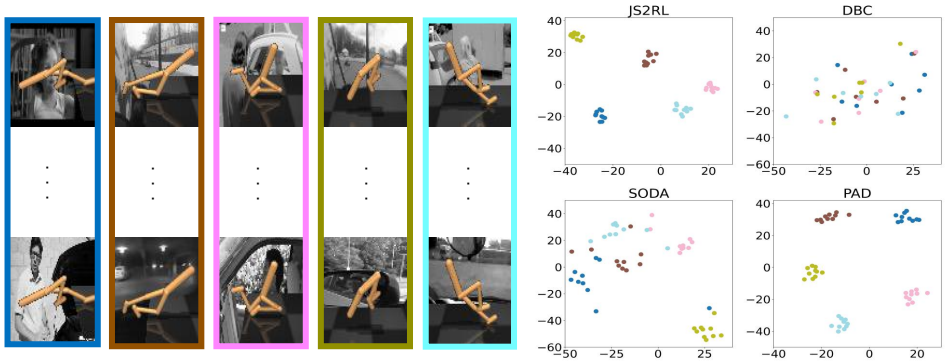


Figure 7: t-SNE of representations learned by JS2RL, DBC, SODA and PAD. Even if the background is dramatically different, JS2RL can encode behaviorally-equivalent observations (blue, brown, pink, olive, sky blue) to be most closely located.

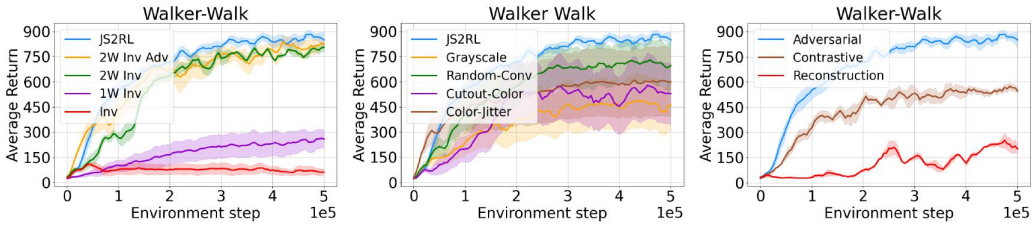


Figure 8: (left) Ablation studies for JS2RL where 2W stands for our proposed 2-way data augmentations, and 1W stands for 1-way data augmentation using a single encoder and Random-shift only. (center) Using multiple strong augmentation techniques together shows better performance than using a single strong augmentation. (right) Our framework can replace Adversarial Representation with other self-supervised learning method such as Reconstruction or Contrastive methods. We show each ablation studies on three different seeds with 1.0 standard error shaded.

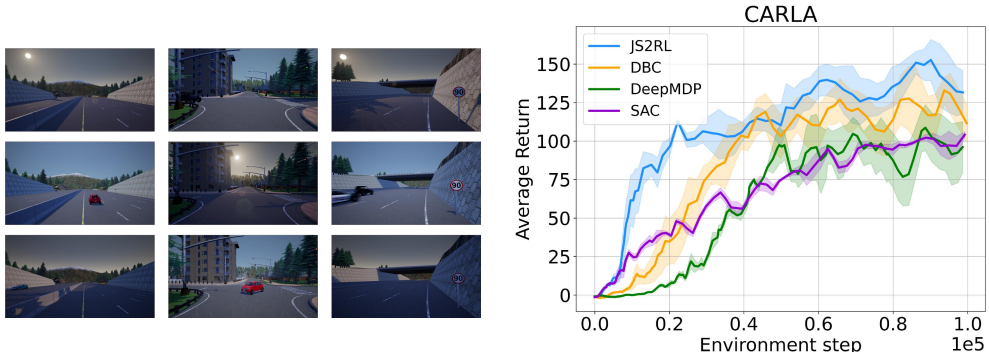
4.2.3 Ablation Studies

We present the ablation studies to examine the synergy of our two-way data augmentations and joint self-supervised learning method. Our ablation experiment is conducted in the same environment setup as the Generalization experiment in Section 4.2.2.

In Figure 8 (left), Inv stands for SAC with Inverse Dynamics but no augmentation, and it hardly learns. 1W-Inv stands for one-way augmentation (Randon-shift only) with Inverse Dynamics, and shows a little improved performance. 2W-Inv is our proposed two-way data augmentations with Inverse Dynamics, and highly improves performance over 1W-Inv. It demonstrates the importance of our two-way data augmentations. 2W-Inv-Adv shows that adding Adversarial Representation can improve early learning efficiency. The whole integration achieves the best performance. Additional ablation of self-supervised learning methods is in Appendix B.

For our strong augmentations, JS2RL uses Random-shift and randomly adds one of Grayscale, Random-Convolution, Cutout-color and Color-jitter every mini-batch. Figure 8 (center) compares the performance when using only one type of strong augmentation each. Random-Convolution seems to have the greatest impact on performance, but when all the data augmentations are used together shows the best performance.

Lastly, in our framework, Adversarial Representation can be replaced with with other self-supervised learning methods such as Reconstruction (Kingma & Welling, 2013) or Contrastive (Srinivas et al., 2020) methods. Figure 8 (right) shows clear differences in achieved task performance according to the different representation learning methods. Using Adversarial Representation greatly outperforms both Reconstruction and Contrastive methods.



(a) Scenes in the CARLA simulator.

(b) Comparison of performance in CARLA.

Figure 9: (a) Scenes in CARLA simulations classified as Highway (left column), Town (center column) and Bridge (right column). Each column is captured in the same spot but contains different task-irrelevant information such as the Sun, rain, shadows, clouds, etc. (b) Performance comparison in the autonomous driving environment CARLA. JS2RL outperforms all other RL methods.

4.3 CARLA Environment

CARLA is a first-person view simulator for studying autonomous driving systems. In the CARLA simulations, we can evaluate the performance of RL methods on more realistic visual observations. As shown in Figure 9a, there are diverse types of distractors (e.g., the Sun, rain, shadows, clouds, etc.) around the agent, and it changes dynamically with every episode, and even within the same episode. Therefore, it becomes more important to extract control-related features (e.g., road, collision, speed, brake, steer, etc.). The basic experimental setup is configured the same as DBC. Visual observation is a 300 degree view from the vehicle roof and the image size is $3\times84\times420$. The reward is defined by the function of driving distance, speed, and the penalty of collision, steering and breaking. Each method is trained for 100K environment steps, and the average return for 20 test episodes is calculated. We run each RL method across three seeds. Figure 9b shows the performance comparison with three seeds in CARLA. DBC using bisimulation metric performs better than SAC and DeepMDP, and JS2RL learns much faster and achieves the highest performance. For another aspect of the representation quality comparison, we suggest the representation distance between two observations. We can intuitively assume that the representation distance should be close if their task-relevant context is similar regardless of other distracting elements. We first took 50 random observations at three locations; Highway, Town, and Bridge in CARLA. We repeatedly collected observations from almost the same spots, but these observation look different because of varying task-irrelevant information (e.g. the Sun, shadows, clouds, rain, car types & colors, etc.), as shown in Figure 9a. We measured the L2 distance in the latent space between varying observations taken from almost the same situation. Table 2 presents the average representation distance normalized to our result. JS2RL has minimal average distance compared to other methods, and we believe this is why our method performs best.

	SAC	DeepMDP	DBC	JS2RL
Highway	3.86	3.04	2.66	**1.00**
Town	6.23	4.14	3.76	**1.00**
Bridge	3.82	1.71	1.67	**1.00**

Table 2: Average representation distance in latent spaces according to task-irrelevant information changes on CARLA simulations. (The numbers are normalized to JS2RL)

5 Conclusion

In this work, we propose a joint self-supervised learning method to enhance a vision-based RL. Based on two-way data augmentations, combining Adversarial Representation, Inverse Dynamics, and Forward Dynamics, can significantly improve the data efficiency and generalization of learning optimal control when operating on complex or unseen background images. In the future, we plan to design sequence-based approaches such as representing video inputs and allowing Inverse & Forward Dynamics to predict multi-steps in latent space.

6 REPRODUCIBILITY STATEMENT

Our code is open-sourced and available at `https://drive.google.com/file/d/1OOd3539XnEvG0YwMkyXH8BlJOqYQ4Y9E/view?usp=sharing`.
We also provide overview of network architecture, implementation details and hyper-parameter setting in Section 4.1, Appendix F.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A Efros. Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*, 2018.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PMLR, 2020.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pp. 1–16. PMLR, 2017.

Linxi Fan, Guanzhi Wang, De-An Huang, Zhiding Yu, Li Fei-Fei, Yuke Zhu, and Anima Anandkumar. Secant: Self-expert cloning for zero-shot generalization of visual policies. *arXiv preprint arXiv:2106.09678*, 2021.

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

Yannis Flet-Berliac, Johan Ferret, Olivier Pietquin, Philippe Preux, and Matthieu Geist. Adversarially guided actor-critic. *arXiv preprint arXiv:2102.04376*, 2021.

Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *arXiv preprint arXiv:1710.11248*, 2017.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596. PMLR, 2018.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G Bellemare. Deepmdp: Learning continuous latent space models for representation learning. In *International Conference on Machine Learning*, pp. 2170–2179. PMLR, 2019.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019a.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019b.

Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13611–13617. IEEE, 2021.

Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International Conference on Machine Learning*, pp. 4182–4192. PMLR, 2020.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29:4565–4573, 2016.

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. *arXiv preprint arXiv:1910.12911*, 2019.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Alexia Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard gan. *arXiv preprint arXiv:1807.00734*, 2018.

Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv preprint arXiv:1705.06950*, 2017.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

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.

Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

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

Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019a.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019b.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *arXiv preprint arXiv:1807.04742*, 2018.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. *arXiv preprint arXiv:1703.02660*, 2017.

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

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

Yujin Tang, Duong Nguyen, and David Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 414–424, 2020.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Sam Witty, Jun Ki Lee, Emma Tosch, Akanksha Atrey, Michael Littman, and David Jensen. Measuring and characterizing generalization in deep reinforcement learning. *arXiv preprint arXiv:1812.02868*, 2018.

Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. `https://github.com/denisyarats/pytorch_sac`, 2020.

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

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.

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

A ADDITIONAL DEEPMIND CONTROL SUITE RESULTS

Figure 10, 11, 12 show performance of Data efficiency on Default, Simple Distractor and Natural Video background setting for nine environments. Figure 13 show performance of Generalization for nine environments. We evaluate performance of Generalization through average return differences when evaluating on Simple Distractor and Natural Video background after training on Simple Distractor background.

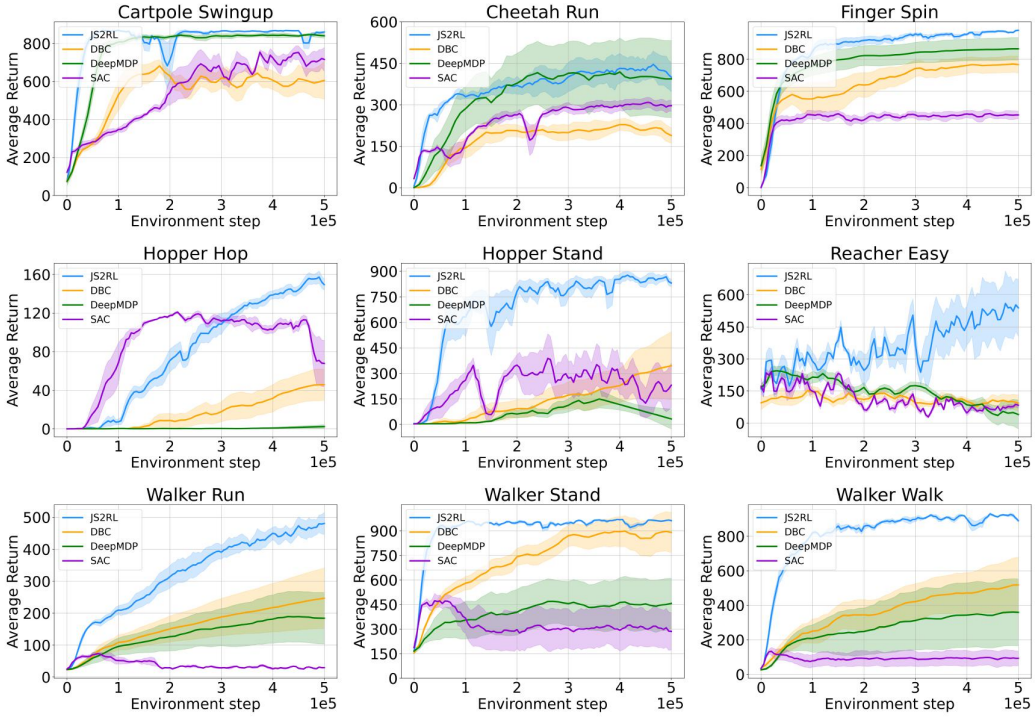


Figure 10: Results of data efficiency evaluation for JS2RL and baselines on Default background. We show the learning curves of each tasks on three different seeds with 1.0 standard error shaded.

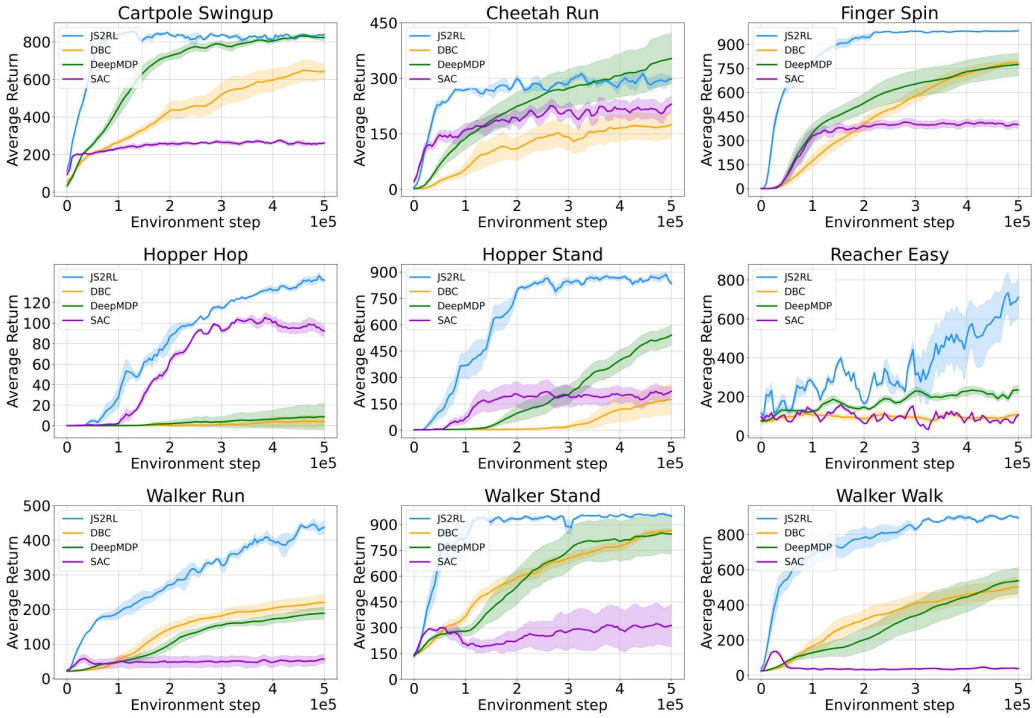


Figure 11: Results of data efficiency evaluation for JS2RL and baselines on Simple Distractor background. We show the learning curves of each tasks on three different seeds with 1.0 standard error shaded.

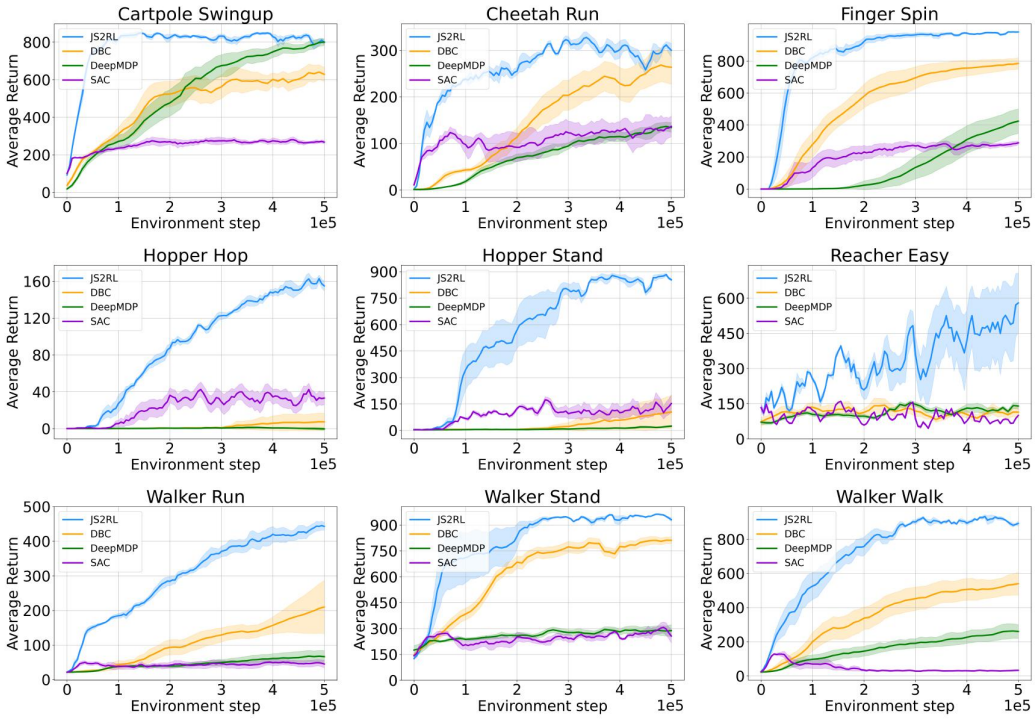


Figure 12: Results of data efficiency evaluation for JS2RL and baselines on Natural Video background. We show the learning curves of each tasks on three different seeds with 1.0 standard error shaded.

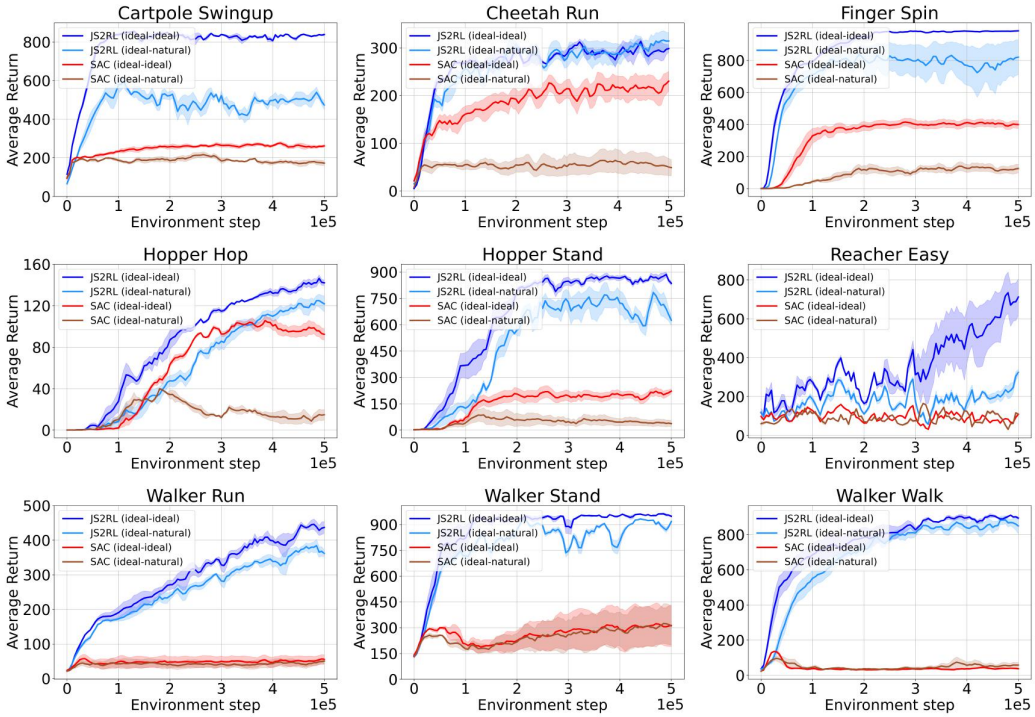


Figure 13: Results of generalization evaluation for JS2RL and baselines. We show the learning curves of each tasks on three different seeds with 1.0 standard error shaded.

B Additional Self-supervision Ablation

In Figure 14, this ablation study is for our framework in Walker-Walk and Hopper-stand environments. Inv stands for SAC with Inverse Dynamics but no augmentation, and it hardly learns. 1W-Inv stands for one-way augmentation (Randon-shift only) with Inverse Dynamics, and shows a little improved performance. 2W-Inv is our proposed two-way data augmentations with Inverse Dynamics, and highly improves performance over 1W-Inv. It demonstrates the importance of our two-way data augmentations. 2W-Inv-Adv shows that adding Adversarial Representation can improve early learning efficiency in Walker-Walk, and more performance gains in Hopper-Stand. The whole integration achieves the best performance.

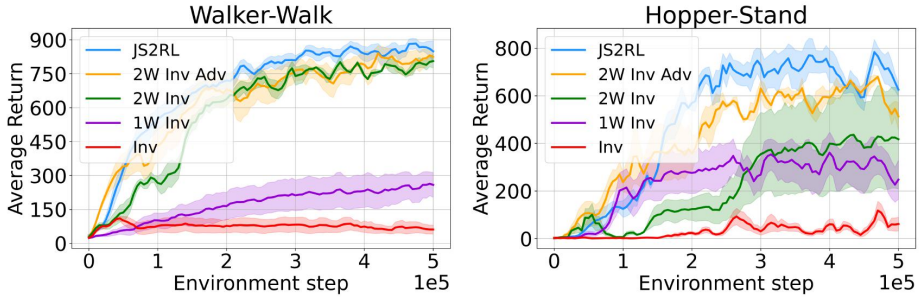


Figure 14: Ablation studies for JS2RL, where 2W stands for our two-way data augmentations and 1W stands for one-way augmentation (Random-shift only). We show generalization performance comparison of each ablation case on three different seeds with 1.0 standard error shaded.

C COMPARISON TO RL ALGORITHMS IMPROVING DATA EFFICIENCY

CURL and DrQ are well knwon RL algorithms that have highly improved Data Efficiency in the image based environment. As shown in Figure 15, these algorithms show very high performance in the Default background, but when distractors are added to the background, their performance gradually degrade, and in the Generalization experiment (trained on Simple Distractor and tested on Natural Video), both algorithms noticeably decrease. We note that the original CURL uses Random Crop which randomly crops an 84 x 84 image from a 100 x 100 simulation-rendered image during training. However, CURL uses a center crop of an 84 x 84 image from a 100 x 100 image for evaluation. In the Deepmind Control suit environments, robots are always located in the center of the simulation images. Therefore, the center crop easily removes background areas. To evaluate the performance on distracting backgrounds and unseen backgrounds, such a center crop is NOT fair. In our experiments, we replace Rrandom Crop (for training) & Center Crop (for evaluation) with Random-shift (for training) & No augmentation (for evaluation).

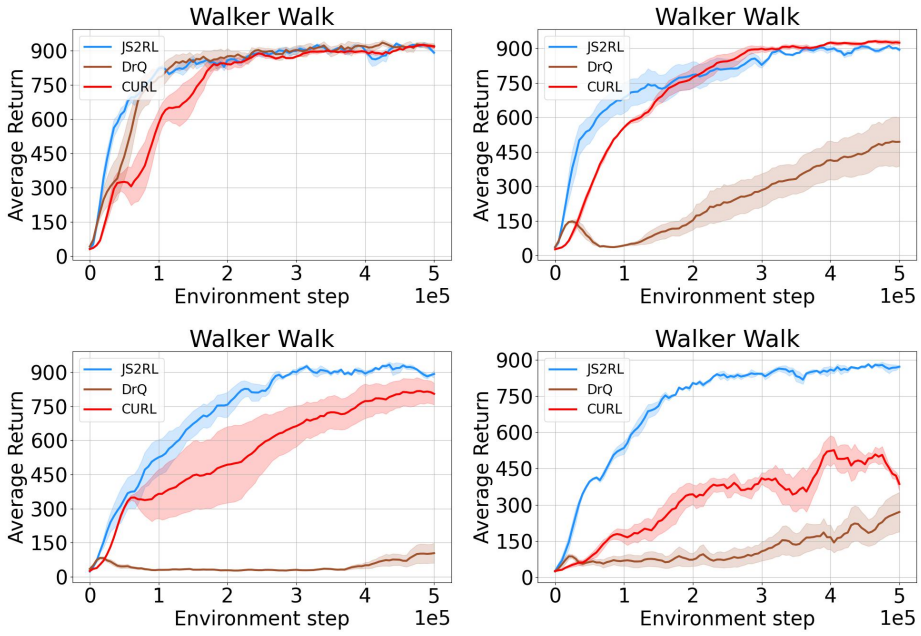


Figure 15: Data efficiency comparisons for each background on Walker Walk in the DeepMind Control suite; Default (top left), Simple Distractor (top right) and Natural Video (bottom left). Generalization performance (bottom right) on Walker Walk. We show the learning curves of each experiments on three different seeds with 1.0 standard error shaded.

D CROSS-ENTERED STRUCTURE FROM INVERSE DYNAMICS TO FORWARD DYNAMICS

In this section, we empirically explain the justification of the cross-entered structure in JS2RL. The cross-entered version used the predicted action $\hat{a}_t^i$ and $\hat{a}_t^j$ from Inverse Dynamics are crossly fed into Forward Dynamics. We compare the generalization performance in three environments between the cross-entered version and the uncross-entered version. As shown in Figure 16, the cross-entered version has higher performance than the uncross-entered version for all three environments slightly.

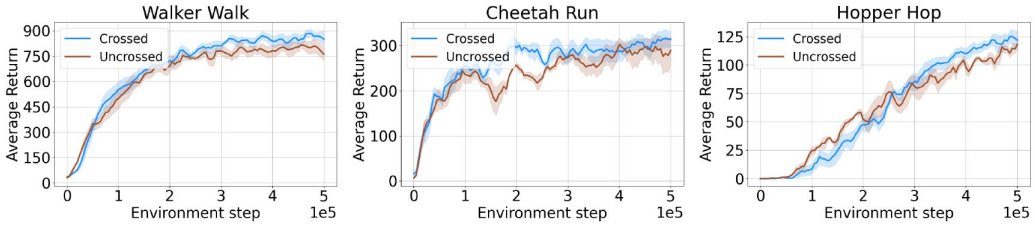


Figure 16: Comparison of cross-entered version (blue) and uncross-entered version (brown) on Walker Walk (left), Cheetah Run (center) and Hopper Hop (right). Cross-entering the predicted actions into Forward Dynamics has a positive effect on performance. We show results on three different seeds with 1.0 standard error shaded.

E APPLY JOINT SELF-SUPERVISED LEARNING METHOD TO OTHER RL ALGORITHMS

In this section, we show whether our joint self-supervised learning method improves the data efficiency and generalization performance using other RL algorithms. We have replaced SAC with a different off-policy algorithms, TD3 (Fujimoto et al., 2018), and one of the on-policy algorithms, PPO (Schulman et al., 2017). As shown in Figure 17, our framework is helpful to improve Data Efficiency and Generalization performance no matter what RL algorithms we apply.

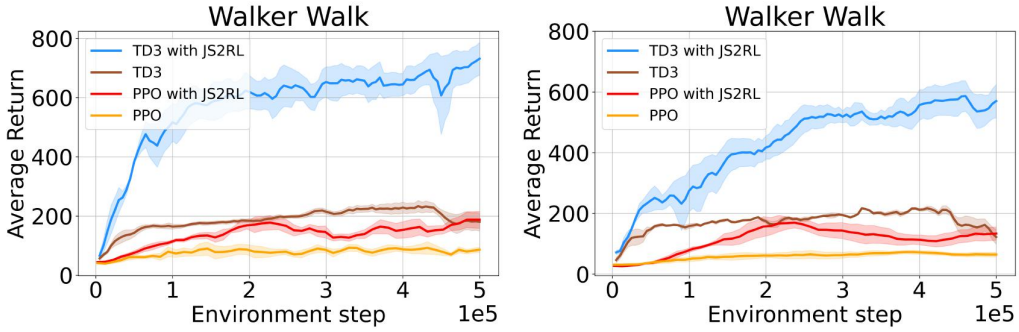


Figure 17: (left) Data Efficiency evaluation, (right) Generalization evaluation on the DeepMind Control suite (Walker Walk). TD3 with JS2RL (blue), standard TD3 (brown), PPO with JS2RL (red), standard PPO (orange). We show the learning curves of each experiments on three different seeds with 1.0 standard error shaded.

F IMPLEMENTATION DETAILS

In this section, we explain the implementation details for JS2RL in the DeepMind Control suite and the CARLA. We use Soft Actor Critic (SAC) (Haarnoja et al., 2018) which is modified by (Yarats et al., 2019) and same encoder architecture as in (Kostrikov et al., 2020) for DeepMind Control

suite and CARLA. For details on the SAC, the reader is referred to (Haarnoja et al., 2018). We use Pytorch implementation of SAC (Yarats & Kostrikov, 2020; Kostrikov et al., 2020) and build JS2RL on top of it. We augment with a shared encoder between the actor, critic, encoder ϕ, Inverse Dynamics I, Forward Dynamics F, Discriminator D.

F.1 NETWORK ARCHITECTURE DETAILS

Our encoder consists of four CNN layers with 3×3 kernels, 32 channels and set the stride to 1 for each layer, except set the first layer's stride to 2. And then we apply ReLU activation function to all CNN layers. Finally, output of CNN layers is fed into a fully-connected layer normalized by LayerNorm (Ba et al., 2016) and apply tanh activation to output of fully-connected layer. This output is 50 dimensional latent vector. The actor, critic, Inverse Dynamics, Forward Dynamics and Discriminator networks share encoder ϕ. The critic (Q-function) consists of three-layer MLP with applied ReLU to all layers except the last layer, and size of hidden layers is 256. The actor also consists of MLP architecture similar to critic and the final output is mean and covariance for the diagonal Gaussian, which represent the policy. Inverse Dynamics I, Forward Dynamics F and Discriminator D networks consist of two-layer MLP with applied ReLU to first layers and size of hidden layers is 256. Output of these networks apply tanh activation. When training in a CARLA environment, we modify encoder slightly. This is the same as used on DeepMind Control suite, except that stride is set to 2 all CNN layers.

F.2 EXPERIMENTAL SETUP

First, our agent collects 1000 observations of size $3\times84\times84$ in DeepMind Control suite and size $3\times420\times420$ using a random policy. After 1000 seed steps, agent is updated for each true environment step (when an episode length is 1000 steps, if action repeat is 2, true environment step is 500). All methods are trained for 500,000 steps (DeepMind Control suite) or 100,000 steps (CARLA). During training, the average return for 10 episodes is calculated to evaluate every 5000 true environment steps (DeepMind Control suite) or the average return for 20 episodes is calculated to evaluate every 10 episodes (CARLA).

F.3 IMAGE AUGMENTATIONS

In common, we use an image observation as an 3-stack of consecutive frames. And then we normalize it by dividing by 255. Image augmentations described in Section 3.2 is applied to the normalized image. We apply augmentation to images sampled from the buffer or a recent trajectory only during training procedure, not environment interaction procedure. In the DeepMind Control suite, when Random shift is applied, all sides are padded by 4 pixels, but in the CARLA environment, the top and bottom sides are padded by 4 pixels, and the left and right sides are padded by 20 pixels according to the image ratio.

F.4 HYPER-PARAMETERS

When applying JS2RL, we adopt hyper parameters used in (Yarats et al., 2019). We detail all hyper parameters used for DeepMind Control suite and CARLA environments in Table 3a and Table 3b.

hyper parameter	Value
Frame	3×84×84
Seed steps	1000
Stacked frames	3
Action repeat	2(walker, finger) 4(otherwise) 8(cartpole)
Batch size	128
Replay buffer size	100,000
Number of training steps	500,000
Discount factor	0.99
Optimizer	Adam
Episode length	1000
Actor, Critic Learning rate	10^{-3}
Inverse Dynamics learning rate	10^{-3}
Forward Dynamics learning rate	10^{-3}
Generator(Encoder) learning rate	10^{-3}
Discriminator learning rate	10^{-3}
Critic target update frequency	2
Critic Q-function soft-update rate	0.01
Actor update frequency	2
Actor log stddev bounds	[-10, 2]
Init temperature	0.1
λ_I, λ_F	0.1
λ_A	0.001

(a) DeepMind Control suite.

hyper parameter	Value
Frame	3×84×420
Seed steps	1000
Stacked frames	3
Action repeat	4
Batch size	128
Replay buffer size	100,000
Number of training steps	100,000
Discount factor	0.99
Optimizer	Adam
Episode length	1000
Actor, Critic Learning rate	10^{-3}
Inverse Dynamics learning rate	3×10^{-4}
Forward Dynamics learning rate	3×10^{-4}
Generator(Encoder) learning rate	3×10^{-4}
Discriminator learning rate	3×10^{-4}
Critic target update frequency	2
Critic Q-function soft-update rate	0.01
Actor update frequency	2
Actor log stddev bounds	[-10, 2]
Init temperature	0.1
λ_I, λ_F	0.1
λ_A	0.001

(b) CARLA.

Table 3: Overview of hyper parameters used for the experiments.

G HYPER-PARAMETER SENSITIVITY

For our hyper-parameter sensitivity testing, we select λ_I and λ_F from $\{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$ and λ_A from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}\}$. We showed that a sensitivity evaluation on Walker Walk in the DeepMind Control suite in Figure 18. We observe that JS2RL has good performance in a wide range of hyper parameter choices.

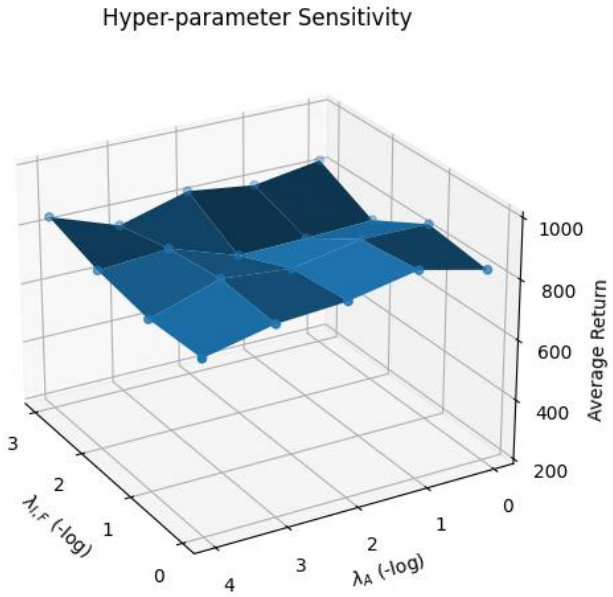


Figure 18: Hyper-parameter sensitivity of JS2RL's objective weights. The value corresponding to each point is the average return over three seeds.

H ASYMPTOTIC PERFORMANCE

In this section, we show learning curve on three tasks with Data efficiency and Generalization experiments when the number of the environment steps increases from 500K to 1M. For longer learning steps, the performance of JS2RL gradually increases slightly or converges well.

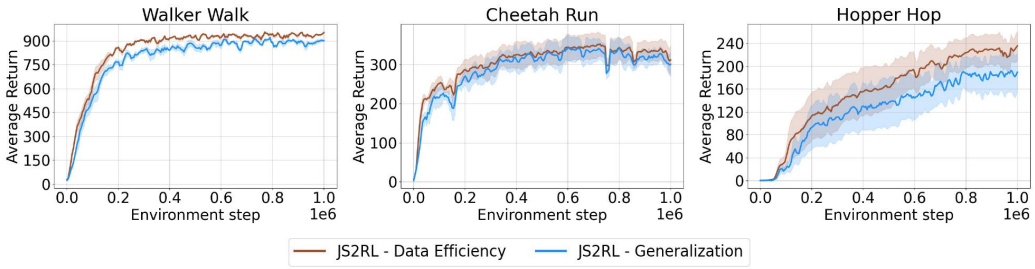


Figure 19: Learning curves with 1M environment steps (brown - Data Efficiency, blue - Generalization). We show the learning curves of each experiments on three different seeds with 1.0 standard error shaded.