LEARNING CONTROLLABLE ELEMENTS ORIENTED REPRESENTATIONS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep Reinforcement Learning (deep RL) has been successfully applied to solve various decision-making problems in recent years. However, the observations in many real-world tasks are often high dimensional and include much task-irrelevant information, limiting the applications of RL algorithms. To tackle this problem, we propose LCER, a representation learning method that aims to provide RL algorithms with compact and sufficient descriptions of the original observations. Specifically, LCER trains representations to retain the *controllable elements* of the environment, which can reflect the *action-related* environment dynamics and thus are likely to be task-relevant. We demonstrate the strength of LCER on the DMControl Suite, proving that it can achieve state-of-the-art performance. To the best of our knowledge, LCER is the first representation learning algorithm that enables the pixel-based SAC to outperform state-based SAC on the DMControl 100K benchmark, showing that the obtained representations can match the oracle descriptions ($i.e.$ the physical states) of the environment.

1 INTRODUCTION

Deep Reinforcement Learning (deep RL) has proven its ability to solve difficult sequential decision-making problems such as Dota2 (Berner et al., 2019) and StarCraft (Vinyals et al., 2019). However, it is still a challenge to apply deep RL to many real-world tasks, because the observations in these tasks are often high dimensional ($e.g.$ pixels) and include much task-irrelevant noise. To address these issues, representation learning is introduced in deep RL to provide information-dense descriptions of the original observations, and thus reduce the difficulty of solving RL problems.

Many representations learning algorithms are proposed with the design goal to make the representations be predictive of properties of future states. Such a goal is typically achieved by imposing the representations to reflect the dynamics of the environment (Mazoure et al.; Lee et al., 2020b; Schwarzer et al., 2020; Gelada et al., 2019). However, the dynamics of the environment are not always task-relevant($e.g.$ the noisy and varying background in control tasks). This can be problematic because those task-irrelevant dynamics will entice the representations into encoding task-irrelevant information($e.g.$ the noisy background). Since the capacity of representations is limited, the obtained representations can miss important task-relevant information (Gelada et al., 2019) or encode too much unimportant task-irrelevant information (Zhang et al., 2020). Therefore, it is essential to learn representations with more task-relevant environment dynamics.

We envision that action-related dynamics are likely to be more task-relevant because actions are the only feedback from the agent to the environment. For example, in control tasks and robotics particularly, the agent can complete its task only if it can realize which part of the environment can be influenced by its actions. For simplicity, we use *controllable elements* to denote those environment elements that can be influenced by the agent's actions and thus reflect the action-related dynamics. We argue that these controllable elements should be explicitly retained for better representations.

Following the intuition above, we develop an efficient and flexible framework named **L**earning **C**ontrollable **E**lements oriented **R**epresentations(LCER) for reinforcement learning to capture the controllable elements of the environment. Specifically, we formally define a metric that measures the amount of controllable elements that representations capture, and then propose a surrogate objective that can make the representations encode controllable elements as well as sufficient information.

We evaluate our method LCER on the DMControl Suite (Tassa et al., 2018), which has been widely used in recent years to benchmark sample efficiency (Kostrikov et al., 2020b; Hafner et al., 2019; Laskin et al.; Lee et al., 2020b). Our experiments show that LCER combined with SAC (Haarnoja et al., 2018) can outperform other representation learning methods on a majority of tasks, and even achieve better performance than the state-based SAC on the DMControl 100K benchmark. We also evaluate LCER on the Distracting Control Suite (Stone et al., 2021), showing LCER's ability to filter out distracting factors.

2 Related Work

In this section, we review the common approaches of representation learning in RL briefly.

Predictive representations in RL. Predictive representations are typically trained to predict the properties of future states. One commonly used approach is to predict the representation of future states given the current state's representation and actions. Such an approach is widely used in model-based RL algorithms. The representations are prone to collapse if without further constraints (Gelada et al., 2019), thus a reconstruction loss is usually applied to ensure non-trivial representations. However, this approach tries to encode everything, unable to filter out task-irrelevant information. SPR (Schwarzer et al., 2020) introduces a target encoder to avoid collapse and reconstruction, but its effect lacks theoretical guarantee. We perform an ablation comparing our method LCER to an SPR-like algorithm in the Appendix B, showing the latter's instability on several tasks. There are also other methods that train predictive representations from information-theoretic perspectives, such as (van den Oord et al., 2018; Hjelm et al., 2019; Anand et al., 2019; Lee et al., 2020b; Mazoure et al.). These methods do not give a bias explicitly towards what kind of environment's dynamics should the representations reflect, while our method LCER focuses more on the action-related dynamics which is more likely to be task-relevant. Our experiments on DMControl tasks in Section 5.1 shows that our method LCER outperforms PI-SAC (Lee et al., 2020b), which is one of the most representative methods among these approaches.

Bisimulation-based Representations in RL. Bisimulation-based representations group those states into a cluster which are indistinguishable w.r.t. future return, given any action sequence tested. The indistinguishability of two states is typically measured by the bisimulation metrics, which are firstly introduced in (Ferns & doina Precup, 2014). Recently, (Zhang et al., 2020) proposes DBC, a representation learning algorithm based on bisimulation that presents impressive results in distracting environments. However, these methods rely on a well-defined reward function, which is not always available in real-world tasks. Besides, they often fail to solve simple tasks efficiently such as ones from DMControl Suite (Zhang et al., 2020).

Prior-knowledge-based Representations in RL There are other methods that use prior knowledge to constraint the representations (Jonschkowski & Brock, 2015; Jonschkowski et al., 2017; Thomas et al., 2018). For example, (Jonschkowski & Brock, 2015) forces the representations to satisfy multiple priors such as proportionality and repeatability, which generally describe how the optimal representations should look like. However, such priors need to be specially designed for each environment, and much domain knowledge is required to form good representations. These drawbacks significantly limit the applications of these methods.

3 Preliminaries

3.1 Notation

We assume the underlying environment is a Markov decision process (MDP), described by the tuple $M = (S, A, P, R, \gamma)$, where S is the state space, A the action space, $P : S_t \times A_t \times S_{t+1} \to [0, 1]$ the transition probability function which determines the distribution of next state given current state and action, and $\gamma \in [0, 1]$ a discount factor. Given the current state $s \in S$, an agent choose its action $a \in A$ according to a policy function $a \sim \pi(\cdot|s)$. This action will update the system state to a new state s' according to the transition function P, and then a reward $r = R(s, a, s') \in R$ is given to the agent. The goal of the agent is to maximize the expected cumulative rewards by learning a policy π.

We denote the state at time step t by s_t. We use the upper letter of s ($i.e.$ S) to refer to the random variable of state if there is no ambiguity. We use $s_{t:t+h}$ to present the state sequence from t to $t+h$, $i.e.$ $s_{t:t+h} := (s_t, s_{t+1}, ..., s_{t+h})$. These notations can be extended to other variables such as actions and rewards. We use $\phi : S \to Z$ to denote the embedding function of LCER, which maps $s \in S$ to a latent space Z.

3.2 Mutual Information

The mutual information(MI) of two random variables X, Y is defined as:

$$I(X;Y) = D_{KL}(P_{XY} \| P_X P_Y), \tag{1}$$

where D_{KL} is the Kullback–Leibler divergence that measures the difference of two distributions. Intuitively, $I(X;Y)$ measures the common information that X, Y shares.

The conditional mutual information(CMI) $I(X;Y|Z)$ is defined as the expected value of $I(X;Y)$ given the value of Z:

$$I(X;Y|Z) = E_{z \sim Z} I(X|z;Y|z). \tag{2}$$

The MI and CMI can be related by the chain rule for mutual information:

$$I(X;[Y,Z]) = I(X;Z) + I(X;Y|Z). \tag{3}$$

4 Learning Controllable Elements Oriented Representations

Given an embedding function: $\phi : S \to Z$, we argue that the number of controllable elements that ϕ contains can be measured by the conditional mutual information(CMI):

$$I_k(\phi) := I(\phi(S_{t+k}); A_{t:t+k-1} | \phi(S_t)) \tag{4}$$

Intuitively, I_k measures the information that $\phi(S_{t+k})$ and $A_{t:t+k-1}$ shares given $\phi(S_t)$. Note that the non-controllable elements captured by ϕ do not contribute to I_k because they are not related to the actions.

In this paper, we argue that the value of I_k should be an important metric to measure the quality of ϕ, because the agent can only interact with these controllable elements to complete its tasks, and thus these controllable elements (measured by I_k) are much likely to be task-relevant. However, in many real-world environments, there exist many distracting factors ($e.g.$ the noisy background) that hider the agent from finding out these controllable elements, making it much harder to solve the given tasks (see Section 5.2 for an example). Therefore, we want ϕ to encode these controllable elements automatically.

However, directly training ϕ to maximize I_k is problematic, because not all task-relevant elements are controllable. For example, in navigation tasks, the position of the goal is task-relevant but not controllable by the agent. In Section 4.1, we propose a surrogate objective that is closely related to I_k but allows to capture more task-relevant elements. In Section 4.2, we describe how to optimize this objective, which leads to our final algorithm LCER.

4.1 The Optimization Objective for LCER

By the chain rule of mutual information, we can break the I_k defined in Eq.(4) into the subtraction of two terms: $I_k(\phi) = I([\phi(S_t), A_{t:t+k-1}]; \phi(S_{t+k})) - I(\phi(S_t); \phi(S_{t+k}))$. The first term tries to capture predictive information, while the second term aims to filter out action-irrelevant information. To make ϕ capture the controllable elements and maintain efficient information at the same time, we introduce a hyper-parameter β to control the degree of filtering out action-irrelevant information:

$$I_k(\phi;\beta) = I([\phi(S_t), A_{t:t+k-1}]; \phi(S_{t+k})) - \beta I(\phi(S_t); \phi(S_{t+k})), \tag{5}$$

By choosing different β when training ϕ to maximize Eq.(5), we can control the "proportion" of controllable elements in ϕ.

However, directly maximizing $I_k(\phi;\beta)$ is intractable, because maximizing the first term and minimizing the second term in Eq.(5) at the same time have an antagonistic effect to each other, thus it

is difficult to tune in practice. In fact, we find this will often lead to unstable training process in our early experiments.

To tackle this problem, we substitute $I(\phi(S_t); \phi(S_{t+k}))$ with $I_k(\phi; \beta = 0) - I_k(\phi; \beta = 1)$:

$$\begin{aligned} I_k(\phi; \beta) &= I([\phi(S_t), A_{t:t+k-1}]; \phi(S_{t+k})) - \beta I(\phi(S_t); \phi(S_{t+k})) \\ &= I([\phi(S_t), A_{t:t+k-1}]; \phi(S_{t+k})) - \beta(I_k(\phi; \beta = 0) - I_k(\phi; \beta = 1)) \\ &= \underbrace{I([\phi(S_t), A_{t:t+k-1}]; \phi(S_{t+k}))}_{J_k^1} \cdot (1 - \beta) + \underbrace{I(\phi(S_{t+k}); A_{t:t+k-1} | \phi(S_t))}_{J_k^2} \cdot \beta. \end{aligned} \tag{6}$$

In Eq.(6), we maximize two terms (J_k^1 and J_k^2) at the same time, and their coefficients are $1 - \beta$ and β respectively. The first term J_k^1 is closely related to the predictive information encoded by ϕ. The second term J_k^2 is related to the controllable elements, which will play an important role in distracting environments as shown in Section 5.3.

In practice, we sum over different choices of k, leading to the final objective to maximize:

$$I^K(\phi; \beta) := \sum_{k=1}^{K} I_k(\phi; \beta). \tag{7}$$

4.2 How to Optimize the Objective for LCER

To maximize $I^K(\phi, \beta)$, we propose to maximize its corresponding lower bound. In this section, we first provides the lower bound for J_k^1 and J_k^2 respectively that are used in this paper, and finally put them together to get our final loss for training ϕ.

Optimizing J_k^1. In order to maximize $J_k^1(\phi)$, we propose to maximize the InfoNCE (van den Oord et al., 2018) lower bound of $J_k^1(\phi)$:

$$\hat{J}_k^1(\phi) = -\frac{1}{B} \sum_{i=1}^{B} \log \frac{\exp(f(\phi(s_t^i), a_{t:t+k-1}^i; \phi(s_{t+k}^i)))}{\sum_{j=1}^{B} \exp(f(\phi(s_t^i), a_{t:t+k-1}^i; \phi(s_{t+k}^j)))}, \tag{8}$$

where f is any scalar function, B is the batch size, and $s_t^i, a_{t:t+k-1}^i, s_{t+k}^i$ is a transition sequence from s_t^i to s_{t+k}^i. There are other alternatives to InfoNCE such as NWJ (Poole et al., 2019) and MINE (Belghazi et al., 2018), however InfoNCE can often lead to better representations as shown in (Tschannen et al., 2020). In order to efficiently optimize $\hat{J}_k^1$ for all $1 \leq k \leq K$, we recursively define ψ_{t+k}^i to encode the information of $(\phi(s_t^i), a_{t:t+k-1}^i)$:

$$\psi_{t+k}^i = h(\psi_{t+k-1}^i, a_{t+k-1}^i), \psi_t^i = \phi(s_t^i), \tag{9}$$

and set f to be the bi-linear inner product of ψ_{t+k}^i and $\phi(s_{t+k}^i)$:

$$f(\phi(s_t^i), a_{t:t+k-1}^i; \phi(s_{t+k}^i)) := (\psi_{t+k}^i)^T W \phi(s_{t+k}^i). \tag{10}$$

Note that the ϕ_{t+k}^i is shared across all $\hat{J}_k$, thus we do not need to re-compute it. The h in Eq.(9) is quite similar to a "prediction" network, but is trained together with ϕ instead of prediction errors.

Optimizing J_k^2. In order to maximize $J_k^2(\phi)$, we turn to maximizing the corresponding Jensen-Shannon-divergence-based CMI $I^{JSD} := I^{JSD}(\phi(S_{t+k}); A_{t:t+k-1} | \phi(S_t))$ [1], because I^{JSD} is proved to be more stable in our experiments, which is also suggested in prior works (Hjelm et al., 2019; Sanchez et al., 2020). To maximize I^{JSD}, we introduce a statistics network $T_k : S \times A_{t:t+k-1} \times S \to R$ and maximize:

$$\hat{J}_k^2(\phi) := -\frac{1}{B} \sum_{i=1}^{B} [\log(1 + e^{-T_k(\phi_t^i, a_{t:t+k-1}^i, \phi_{t+k}^i)}) + \log(1 + e^{T_k(\phi_t^i, a_{t:t+k-1}^{j(i)}, \phi_{t+k}^i)})], \tag{11}$$

where $j(1), .., j(B)$ are obtained by randomly shuffling $1, .., B$. The derivation of Eq.(11) is provided in Appendix B. In practice, we choose $T_k(\phi_t, a_{t:t+k-1}, \phi_{t+k}) = T(\psi_{t+k}, \phi_{t+k})$, where ψ_{t+k} is a latent defined in Eq.(9) and $T : Z \times Z \to R$ is a network shared across different T_k's.

[1] $I^{JSD}(X; Y|Z) = E_{z \sim Z} D_{JS}(P_{XY|z} \| P_{X|z} P_{Y|z})$, where D_{JS} is the Jensen-Shannon divergence

Environment	LCER(ours)	PI-SAC	SLAC	CURL	PlaNet	StateSAC
cartpole-swingup	$\mathbf{790 \pm 40}$	772 ± 49	327 ± 44	582 ± 146	491 ± 66	860 ± 7
cheetah-run	$\mathbf{505 \pm 38}$	271 ± 57	413 ± 67	299 ± 48	304 ± 131	206 ± 28
walker-walk	$\mathbf{622 \pm 127}$	431 ± 80	528 ± 41	403 ± 24	350 ± 58	664 ± 111
reacher-easy	$\mathbf{704 \pm 160}$	687 ± 133	342 ± 96	538 ± 233	419 ± 465	774 ± 111
finger-spin	$\mathbf{961 \pm 56}$	942 ± 84	951 ± 32	767 ± 56	561 ± 283	749 ± 234
ball_in_cup-catch	$\mathbf{933 \pm 25}$	878 ± 117	917 ± 21	769 ± 43	459 ± 380	951 ± 18
Average Returns	**752**	663	579	559	430	700

Table 1: The DMControl 100K benchmarks, which report the performance at 100000 environment steps in the PlaNet benchmarks. Our method LCER achieves the best performance on **6 of 6** tasks, and is the only one that outperforms StateSAC w.r.t the average returns.

Final loss for training ϕ. Inspired by (He et al., 2020), we propose to utilize a *target encoder* ϕ_{targ} to stabilize the training process, whose weights are the exponential moving average(EMA[2]) of ϕ. Specifically, the embeddings of s^i_{t+k} $(k \geq 1)$ are calculated by ϕ_{targ}.

All pieces together, we obtain our loss for ϕ to minimize:

$$\begin{aligned} L^K(\phi; h, W, T) &= -\sum_{k=1}^{K} \hat{J}^1_k(\phi) \cdot (1-\beta) + \hat{J}^2_k(\phi) \cdot \beta \\ &= \frac{1}{B} \sum_{i=1}^{B} \sum_{k=1}^{K} [(1-\beta) \log \frac{\exp((\psi^i_{t+k})^T W \phi_{targ}(s^i_{t+k}))}{\sum_{j=1}^{B} \exp((\psi^i_{t+k})^T W \phi_{targ}(s^j_{t+k}))} \\ &+ \beta(\log(1 + \exp(-T(\psi^i_{t+k}, \phi_{targ}(s^i_{t+k}))) + \log(1 + \exp(T(\hat{\psi}^i_t, \phi_{targ}(s^i_{t+k})))], \end{aligned} \tag{12}$$

where B is the batch size, h, ψ^i_{t+k} are defined in Eq.(9), W is defined in Eq.(10), $\hat{\psi}^i_{t+k}$ is a latent defined in the same way with ψ^i_{t+k} but it encodes $\phi(s^i_t)$ and $a^{j(i)}_{t:t+k-1}$(instead of $a^i_{t:t+k-1}$, as discussed in Eq.(11)), T is a scalar function, and ϕ_{targ} is the target encoder. The parameters of ϕ, h, W, T are training together by minimizing L^K. In practice, Eq.(12) can be used as an *auxiliary loss* for any RL algorithms such as SAC (Haarnoja et al., 2018).

5 Experiments

In this section, we first compare LCER with other methods in Section 5.1 on the PlaNet benchmark. In Section 5.2, we then demonstrate LCER's strength to filter out distracting factors on the Distracting Control Suite, which is important for RL algorithms to solve real-world problems. In Section 5.3, we investigate the effect of K and β, which are the most important parameter in LCER. Finally, we demonstrate LCER can obtain representations of higher quality than the vanilla SAC, as well as its potential to transfer learning in Section 5.4.

In our experiments, LCER is equipped with SAC (Haarnoja et al., 2018). Most of the parameters in LCER are the same with PI-SAC (Lee et al., 2020b), including the batch size, learning rate, action repeat, and the network architecture of ϕ, see Appendix A for more details.

Following (Laskin et al.; Lee et al., 2020a;b), we report the performance using the true environment steps, which is invariant to the choice of action repeat. Unless specially specified, We set $K = 5, \beta = 0.1$ and run every experiment using 10 random seeds.

5.1 Evaluation on Sample Efficiency

In this section, We evaluate LCER on the PlaNet benchmark, which consists of six challenging control tasks from the DMControl Suite (Tassa et al., 2018). The PlaNet benchmark is first introduced in (Hafner et al., 2019) and later widely used to benchmark sample efficiency in (Kostrikov et al., 2020b; Laskin et al.; Lee et al., 2020b;a).

[2]EMA is a widely used trick in RL. For example, in SAC, the weights of target Q are the EMA of Q.

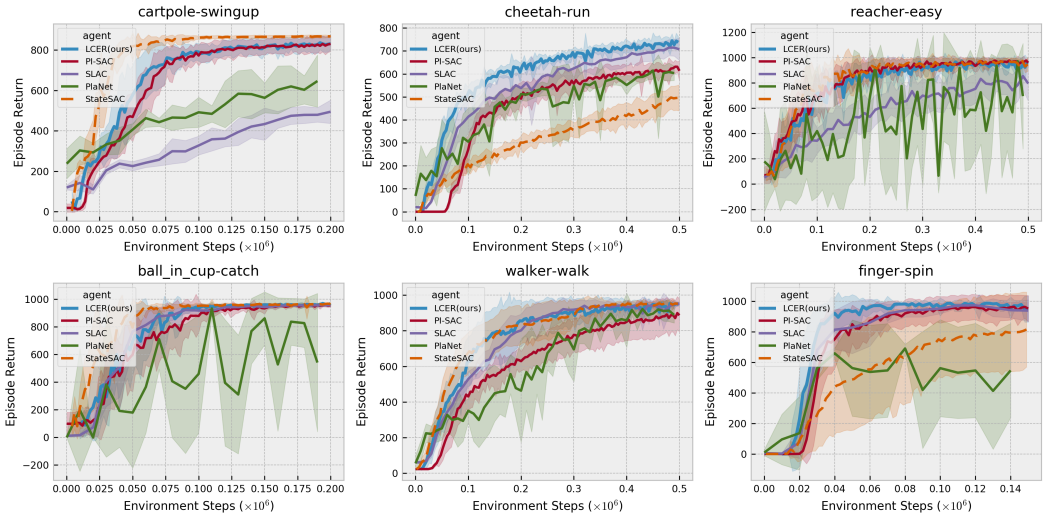


Figure 1: The PlaNet Benchmarks. The performance of StateSAC can be seen as an upper bound because it learns directly from internal physical states rather than pixels. Our algorithm LCER consistently achieves better or comparable performance than other methods(except StateSAC) on all environments

We equip LCER with SAC (Haarnoja et al., 2018) and compare LCER to 3 leading representation learning algorithms[3]: CURL (Laskin et al.),SLAC (Lee et al., 2020a),and PI-SAC (Lee et al., 2020b). CURL introduces contrastive learning into RL and actually trains representations to encode information that is invariant under data augmentation. SLAC efficiently combines both representation learning and control into a single objective via Bayesian inference, but it relies on a reconstruction loss. PI-SAC tries to encode predictive information (as described in the Related Works), while LCER focuses on encoding controllable elements. All these methods are also built on SAC (Haarnoja et al., 2018). Following (Laskin et al.; Kostrikov et al., 2020b), we use the version of PI-SAC that performs one gradient update per environment step to ensure a fair comparison. We also compare LCER with a model-based algorithm PlaNet (Hafner et al., 2019) in which the PlaNet benchmark is firstly introduced.

To further evaluate the quality of the representations obtained by LCER, we also consider the state-based SAC(which we called StateSAC) that directly learns from internal physical states provided by DMControl Suite. The performance of StateSAC can be seen as an upper bound since it utilizes the oracle representations of the environments. All parameters of StateSAC are the same with LCER except that it uses states as input instead of pixels.

In Figure 1, we report the learning curves of LCER, PI-SAC (Lee et al., 2020b), SLAC (Lee et al., 2020a), and PlaNet (Hafner et al., 2019). LCER consistently achieves better or comparable performance than other methods. In Table 1, We also report the performance of each algorithm at 100k environment steps. LCER is the only algorithm that enables pixel-based SAC to outperform state-based SAC w.r.t the average returns, showing that the obtained representations can match the oracle descriptions($i.e.$ the physical states) of the environment.

In practice, pixel-based observations are often equipped with data augmentation to serve as a regularizer (Kostrikov et al., 2020b; Lee et al., 2020b; Laskin et al.; 2020). In LCER, we also randomly shift the observations by $[-4, 4]$ pixels. However, a proper data augmentation method is not always available for general forms of observations. Therefore, We further test our method LCER in the situation where no data augmentation is available. As shown in Figure 2, LCER can still improve the performance to a great extent without the help of data augmentation. Note that some sample-efficient RL algorithms such as DrQ (Kostrikov et al., 2020b) and RAD (Laskin et al., 2020) that rely on data augmentation can not be extended to such a situation naturally.

[3]The performance data of SLAC, PlaNet, CURL is provided by the authors of the corresponding papers.

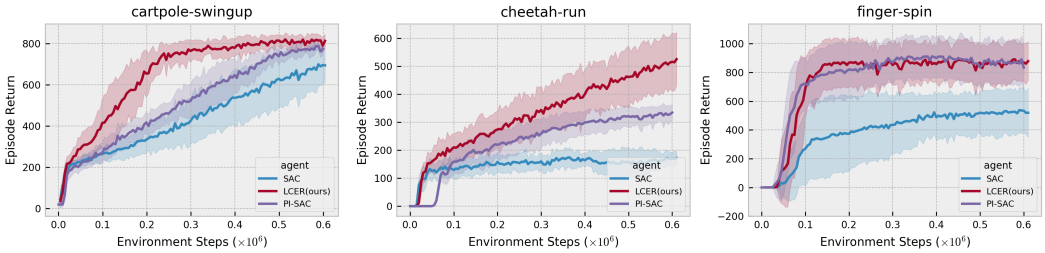


Figure 2: The learning curves when data augmentation is not available. LCER can still continuously improve the performance to a great extent without the help of data augmentation, showing LCER's ability to handle different data formats (beyond pixels) of observations.

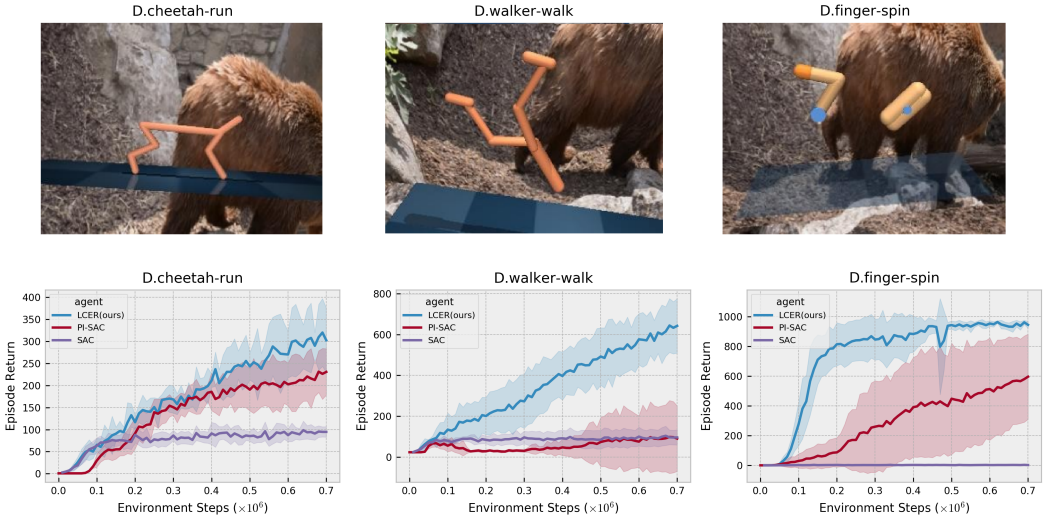


Figure 3: The Learning curves on Distracting Control Suite, a challenging benchmark that contains several kinds of visual distractions. Top row: the observations on each task. Bottom row: the learning curves on each task. Although these tasks are so difficult that SAC fails to solve, LCER can still improve the agent's performance gradually, showing LCER's robustness to distractions.

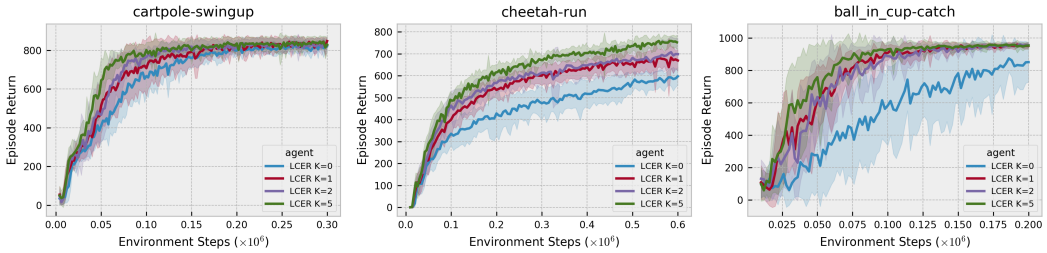
Our experiments on the PlaNet benchmark show that our proposed method LCER can obtain representations that help solve the tasks, and thus improve the sample efficiency.

5.2 Robustness to Distractions

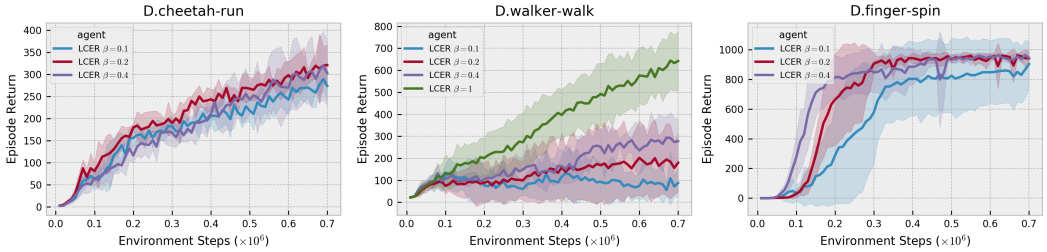
In this section, we test LCER on the Distracting Control Suite (Stone et al., 2021), a challenging benchmark that is implemented by introducing several kinds of visual distractions to the DMControl Suite, such as random changes to camera pose, object color, and background(see Figure 3). These distractions widely exist in the real world and can reduce the performance of RL algorithms to a great extent (Stone et al., 2021). We aim to show LCER's ability to filter out those distractions. Note that some representation learning methods such as SLAC(Lee et al., 2020a) rely on a reconstruction loss, which will lead to significant performance degradation as shown in (Zhang et al., 2020).

In Figure 3, we run experiments on three control tasks from the Distracting Control Suite: D.cheetah-run, D.walker-walk, D.finger-spin[4]. In all tasks, we set the difficulty to be 'easy' and use a dynamically varying background(refer to (Stone et al., 2021) for more details). We choose $\beta = 0.4$ for D.cheetah-run and D.finger-spin, and $\beta = 1$ for D.walker-walk. As shown in Figure 3, proper representation learning methods are crucial in these tasks, because it is difficult for the agent to filter out

[4]We use "D." to distinguish these tasks from the ones in the DMControl Suite.



(a) The effect of K on performance.



(b) The effect of β on performance.

Figure 4: The effect of K and β. Figure 4(a) shows that larger K generally leads to better performance. The effect of β is more complicated as shown in Figure 4(b). Larger β dose lead to better performance within a certain range, however when β goes beyond this range($i.e.\beta = 1$), it may hinder the performance. In practice, we can gradually increase the value of β to search for optimal performance.

task-irrelevant information by itself. LCER achieves much better performance than PI-SAC, which indicate that it can not lead to high-quality representations just by encoding predictive information, instead we have to attach more importance to the controllable elements. We argue that this is because the distractions are often not controllable by the agent, and thus can be filtered out by LCER. Although these distractions do have a negative effect on LCER's performance, LCER suffer less than other methods, showing LCER's robustness to these distractions.

5.3 The Effect of K and β

In this section, we try to specify the effect of K and β, which are the most important hyper-parameters in LCER.

To investigate the effect of K, we report the performance for $K = 0, 1, 2, 5$ in Figure 4(a). Note that $K = 0$ is a special case in LCER that does not take into account any actions, and exactly recovers the work of CURL (Laskin et al.). We add $K = 0$ to emphasize the importance of actions. As shown in Figure 4(a), larger K leads to better performance, especially when K changes from 0 to 1. We think this is because large K can provide more supervision signals thanks to our special design of Eq.(12).

To investigated the effect of β, we report the performance for $\beta = 0.1, 0.2, 0.4, 1$ in Figure 4(b). As shown in Figure 4(b), larger β dose lead to better performance within a certain range, however when β goes beyond this range($e.g.\beta = 1$), it may hinder the performance in some tasks. The result in Figure 4(b) indicates that the optimal β should be chosen according to the given environment. By introducing a hyper-parameter β, LCER is able to strike a balance on the proportion of controllable elements that ϕ captures, allowing for searching for better performance.

5.4 The Quality of Representations

In this section, we try to measure the quality of ϕ by generalizing ϕ to unseen tasks. Specifically, we train ϕ on a source task, froze ϕ, and then apply it to a target task within the same domain. Since

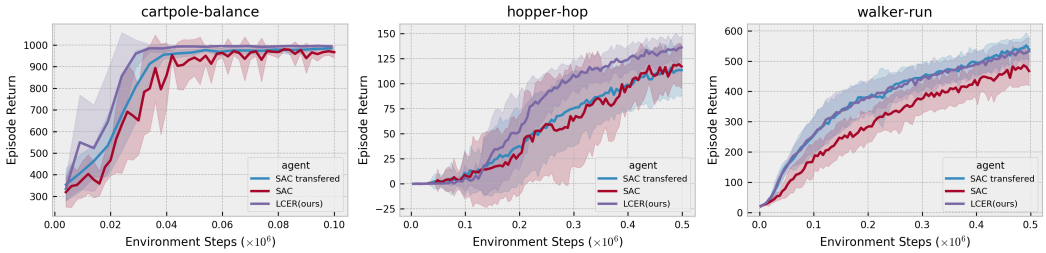


Figure 5: The learning curves of transferring obtained representations to target tasks. From left to right, the source tasks are cartpole-swingup, hopper-stand, walker-walk respectively. "LCER transferred" and "SAC transferred" refer to using the representations obtained by LCER and vanilla SAC respectively, and "SAC" refers to training from scratch using vanilla SAC on target tasks. The representation obtained by LCER can encode more useful information than vanilla SAC, leading to better performance on target tasks.

tasks within the same domain share the same environment dynamics, the performance on target tasks can measure how well ϕ can reflect the dynamics, and thus can be a metric of the representations' quality.

In Figure 5, we compare ϕ ("LCER transferred") with another representation function ψ ("SAC transferred") that is learned by the vanilla SAC. We first train ϕ or ψ on source tasks for 300k environment steps and then apply it to target tasks with frozen weights. As shown in Figure 5, ϕ generally achieves better performance than ψ, proving that LCER can learn better representation than the vanilla SAC. This is because ϕ can reflect the environment's dynamics which are shared across the tasks from the same domain, while ψ only encodes those elements that are helpful to solve the source task. As shown in Figure 5, ϕ can also accelerate the training process on target tasks, showing LCER's potential to transfer learning.

6 Conclusion

In this paper, we introduce the LCER, an efficient framework to train representations that can reflect the action-related dynamics of the environment. By focusing on the controllable elements of the environment, LCER can encode task-relevant information into representations, improving the performance of RL algorithms. Our experiments on DMControl 100K benchmark show that LCER can enable the pixel-based SAC to outperform state-based SAC, demonstrating improvement over prior works. We also evaluate LCER on the Distracting Control Suite, showing LCER's robustness to distractions.

Although LCER has proven to be effective by our experiments, there still remains some room to improve. For example, we have to manually tune β according to the given environment, which will cost lots of time in some scenario. We think some automatically hyper-parameter tuning technologies such as meta-gradient can help solve this problem.

References

Rishabh Agarwal, Marlos C. Machado, Pablo Samuel Castro, and Marc G. Bellemare. Contrastive behavioral similarity embeddings for generalization in reinforcement learning. *CoRR*, 2021.

Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R. Devon Hjelm. Unsupervised state representation learning in atari. *Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019.

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeswar, Sherjil Ozair, Yoshua Bengio, R. Devon Hjelm, and Aaron C. Courville. Mutual information neural estimation. *Proceedings of the 35th International Conference on Machine Learning, ICML*, 2018.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, 2019.

Pengyu Cheng, Weituo Hao, Shuyang Dai, Jiachang Liu, Zhe Gan, and Lawrence Carin. CLUB: A contrastive log-ratio upper bound of mutual information. *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2020.

Norman Ferns and doina Precup. Bisimulation metrics are optimal value functions. *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI*, 2014.

Carles Gelada, Saurabh Kumar, Jacob Buckman, Ofir Nachum, and Marc G. Bellemare. Deepmdp: Learning continuous latent space models for representation learning. *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Zhaohan Daniel Guo, Bernardo Ávila Pires, Bilal Piot, Jean-Bastien Grill, Florent Altché, Rémi Munos, and Mohammad Gheshlaghi Azar. Bootstrap latent-predictive representations for multitask reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2020.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *CoRR*, 2018.

Danijar Hafner, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020.

R. Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Philip Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *7th International Conference on Learning Representations, ICLR*, 2019.

Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Auton. Robots*, 2015.

Rico Jonschkowski, Roland Hafner, Jonathan Scholz, and Martin A. Riedmiller. Pves: Position-velocity encoders for unsupervised learning of structured state representations. *CoRR*, 2017.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *CoRR*, 2020a.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *CoRR*, 2020b.

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. CURL: contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*, 2020.

Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*, 2020a.

Kuang-Huei Lee, Ian Fischer, Anthony Liu, Yijie Guo, Honglak Lee, John Canny, and Sergio Guadarrama. Predictive information accelerates learning in RL. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS*, 2020b.

Bogdan Mazoure, Remi Tachet des Combes, Thang Doan, Philip Bachman, and R. Devon Hjelm. Deep reinforcement and infomax learning. *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, NeurIPS*, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS*, 2019.

Ben Poole, Sherjil Ozair, Aäron van den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. *Proceedings of the 36th International Conference on Machine Learning, ICML*, 2019.

Eduardo Hugo Sanchez, Mathieu Serrurier, and Mathias Ortner. Learning disentangled representations via mutual information estimation. 2020.

Max Schwarzer, Ankesh Anand, Rishab Goel, R. Devon Hjelm, Aaron C. Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *CoRR*, 2020.

Rui Shu, Tung Nguyen, Yinlam Chow, Tuan Pham, Khoat Than, Mohammad Ghavamzadeh, Stefano Ermon, and Hung H. Bui. Predictive coding for locally-linear control. *Proceedings of the 37th International Conference on Machine Learning, ICML*, 2020.

Alessandro Sordoni, Nouha Dziri, Hannes Schulz, Geoffrey J. Gordon, Philip Bachman, and Remi Tachet des Combes. Decomposed mutual information estimation for contrastive representation learning. In *Proceedings of the 38th International Conference on Machine Learning, ICML*, 2021.

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite - A challenging benchmark for reinforcement learning from pixels. *CoRR*, 2021.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy P. Lillicrap, and Martin A. Riedmiller. Deepmind control suite. *CoRR*, 2018.

Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, doina Precup, and Yoshua Bengio. Disentangling the independently controllable factors of variation by interacting with the world. *CoRR*, 2018.

Michael Tschannen, Josip Djolonga, Paul K. Rubenstein, Sylvain Gelly, and Mario Lucic. On mutual information maximization for representation learning. *8th International Conference on Learning Representations, ICLR*, 2020.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *CoRR*, 2018.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard,

David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Çaglar Gülçehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft II using multi-agent reinforcement learning. *Nat.*, 2019.

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

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

A IMPLEMENTATION

In this section, we explain the implementation in details.

Action Repeat Following (Lee et al., 2020b; Laskin et al.; Lee et al., 2020a; Hafner et al., 2019), we use different action repeat to different environments. The action repeat is a parameter that determine how frequently the agent is supposed to make a decision. For a fair comparison, we adopt the action repeat from (Lee et al., 2020b;a). Specifically, for cheetah-run, cartpole-swingup, reacher-easy, and ball_in_cup-catch, we use action repeat of 4. For walker-walk and finger-spin, we use 2 and 1 respectively. Note that the action repeat parameters can directly affect the performance.

Initial collecting step Following (Lee et al., 2020b;a), we pre-train our model using a small amount of random data. Specifically, we first interact with the environment using a random policy to collect initial experience, and then using the experience to pre-train our encoder ϕ. Note that the amount of initial random data has been taken into account in all experiments. For cheetah-run, walker-walk, reacher-easy, and ball_in_cup-catch, we collect random data for 10000 true environment steps and then train ϕ for 10000 iterations. For finger-spin, 10000 true environment steps and 50000 iterations respectively. For cartpole-swingup, 4000 true environment steps and 1000 iterations.

Network Architectures We use the encoder($i.e.$ ϕ) that shares the same architecture with (Laskin et al.; Lee et al., 2020b), which consists of 4 convolution layers(32 3×3 filters with stride 2 for the first layer, and 32 3×3 with stride 1 for the rest 3 layers) and a fully-connected layer that flattens the output of convolution layers and then maps it into a latent space with 50 dimensions. For the h in Eq.(9), we use MLPs with 2 128-d hidden layers, whose input and output are both of 50 dimensions. For the actor and critic, we both use MLPs with two 512-d hidden layers. We use tanh to squash the output of the actor to make sure it fits the action bound of the environment.

Other Hyperparameters We list the other hyperparamters in Table 2. Most of these paramters are the same with (Lee et al., 2020b).

Hyperparameter	Value
Batch size	256
Learning rate	3e-4
Replay buffer size	100000
Dimension of latent space	50
Stacked frames	3
Optimizer	Adam
(β_1, β_2) in Adam	(0.5, 0.999) for α, (0.9, 0.999) for others
target update interval	1
actor log std	[-10, 2]
Q EMA τ_i	0.005
Encoder EMA τ_ϕ	0.05
Initial temperature	0.1
Discount	0.99
Non-linearity	ReLU
Entropy Target	-dim(A)

Table 2: The hyperparameters of LCER.

B ESTIMATION OF CONDITIONAL MUTUAL INFORMATION

In this section, we describe how we optimize I^{JSD} defined in Section 4.2. In the followings, the $\phi(S_{t+k}), A_{t:t+k-1}, \phi(S_t)$ are denoted by X, Y, Z respectively.

We first borrow some theoretical results from (Nowozin et al., 2016):

$$\begin{aligned} I^{JSD}(X;Y|Z) &= E_{z\sim Z} D_{JS}(P_{XY|z} \| P_{X|z} P_{Y|z}) \\ &= D_{JSD}(P_{XYZ} \| P_{XY} P_{Y|Z}) \\ &\geq \sup_V E_{x,y,z\sim XYZ} V(x,y,z) - E_{x,z\sim XZ, y\sim Y|z} f^*_{JSD}(V(x,y,z)). \end{aligned} \tag{13}$$

Here, f_{JSD} is the generator function of Jensen-Shannon divergence: $f_{JSD}(u) = -(u+1)\log\frac{1+u}{2} + u\log u$, and f^*_{JSD} is its corresponding convex conjugate function: $f^*_{JSD}(t) = \sup_u\{ut - f_{JSD}(u)\} = -\log(2-\exp(t))$. We choose $V(x,y,z) = \log(2) - \log(1+\exp(-T_k(x,y,z)))$, then:

$$\begin{aligned} I^{JSD}(X;Y|Z) &\geq \sup_V E_{x,y,z\sim XYZ} V(x,y,z) - E_{x,z\sim XZ, y\sim Y|z} f^*_{JSD}(V(x,y,z)) \\ &= \sup_{T_k} -E_{x,y,z\sim XYZ} \log(1+e^{-T_k(x,y,z)}) - E_{x,z\sim XZ, y\sim Y|z} \log(1+e^{T_k(x,y,z)}) + 2\log 2 \\ &\approx \sup_{T_k} -E_{x,y,z\sim XYZ} \log(1+e^{-T_k(x,y,z)}) - E_{x,z\sim XZ, y\sim Y} \log(1+e^{T_k(x,y,z)}) + 2\log 2. \end{aligned} \tag{14}$$

In Eq.(14), we suppose $p(y|z) \approx p(y)$($i.e.$ $A_{t:t+k-1}$ is independent from $\phi(S_t)$) for simplicity. Such an assumption is appropriate in our case, because $(S_t, A_{t:t+k-1}, S_{t+k})$ is sampled from a large replay buffer($= 10^5$) whose action policy is quite non-stationary due to policy updates.

To estimate the RHS of Eq.(14), we first randomly sample B transitions $\{s^i_t, a^i_{t:t+k-1}, s^i_{t+k}\}_{i=1}^B$ from the replay buffer. Assume $(j(1), .., j(B))$ is a permutation of $(1, 2, .., B)$ obtained by random shuffling, then $E_{x,y,z\sim XYZ}(\cdot)$ can be approximated by $\frac{1}{B}\sum_{s^i_t, a^i_{t:t+k-1}, s^i_{t+k}}(\cdot)$ and $E_{x,z\sim XZ, y\sim Y}(\cdot)$ by $\frac{1}{B}\sum_{s^i_t, a^{j(i)}_{t:t+k-1}, s^i_{t+k}}(\cdot)$. Therefore, the RHS of Eq.(14) can be estimated by:

$$\hat{J}^2_k(\phi) := -\frac{1}{B}\sum_{i=1}^{B}[\log(1+e^{-T_k(\phi^i_t, a^i_{t:t+k-1}, \phi^i_{t+k})}) + \log(1+e^{T_k(\phi^i_t, a^{j(i)}_{t:t+k-1}, \phi^i_{t+k})})], \tag{15}$$

Here, we omit the "$2\log 2$" in Eq.(14) for simplicity.