Predictive Information Accelerates Learning in RL

Kuang-Huei Lee
Google Research
leekh@google.com

Ian Fischer
Google Research
iansf@google.com

Anthony Z. Liu
University of Michigan
anthliu@umich.edu

Yijie Guo
University of Michigan
guoyijie@umich.edu

Honglak Lee
Google Research
honglak@google.com

John Canny
Google Research
canny@google.com

Sergio Guadarrama
Google Research
sguada@google.com

Abstract

The *Predictive Information* is the mutual information between the past and the future, $I(X_{\text{past}}; X_{\text{future}})$. We hypothesize that capturing the predictive information is useful in RL, since the ability to model what will happen next is necessary for success on many tasks. To test our hypothesis, we train Soft Actor-Critic (SAC) agents from pixels with an auxiliary task that learns a compressed representation of the predictive information of the RL environment dynamics using a contrastive version of the Conditional Entropy Bottleneck (CEB) objective. We refer to these as Predictive Information SAC (PI-SAC) agents. We show that PI-SAC agents can substantially improve sample efficiency over challenging baselines on tasks from the DM Control suite of continuous control environments. We evaluate PI-SAC agents by comparing against uncompressed PI-SAC agents, other compressed and uncompressed agents, and SAC agents directly trained from pixels. Our implementation is given on GitHub.[1]

1 Introduction

Many Reinforcement Learning environments have specific dynamics and clear temporal structure: observations of the past allow us to predict what is likely to happen in the future. However, it is also commonly the case that not all information about the past is relevant for predicting the future. Indeed, there is a common Markov assumption in the modeling of RL tasks: given the full state at time t, the past and the future are independent of each other.

However, in general not all RL tasks are specified with a full state vector that can guarantee Markovity. Instead, the environment may be only partially observable, or the state may be represented in very high dimensions, such as an image. In such environments, the task of the agent may be described as finding a representation of the past that is most useful for predicting the future, upon which an optimal policy may more easily be learned.

In this work, we approach the problem of learning continuous control policies from pixel observations. We do this by first explicitly modeling the *Predictive Information*, the mutual information between the past and the future. In so doing, we are looking for a compressed representation of the past that the agent can use to select its next action, since most of the information about the past is irrelevant

[1] https://github.com/google-research/pisac

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

for predicting the future, as shown in [4]. This corresponds to learning a small state description that makes the environment more Markovian, rather than using the entire observed past as a state vector. This explicit requirement for a concise representation of the Predictive Information leads us to prefer objective functions that are *compressive*. Philosophically and technically, this is in contrast to other recent approaches that have been described in terms of the Predictive Information, such as Contrastive Predictive Coding (CPC) [30] and Deep InfoMax (DIM) [19], which do not explicitly compress.

Modeling the Predictive Information is, of course, insufficient to solve RL problems. We must also provide a mechanism for learning how to select actions. In purely model-based approaches, such as PlaNet [18], that can be achieved with a planner and a reward estimator that both use the model's state representation. Alternatively, one can use the learned state representation as an input to a model-free RL algorithm. That is the approach we explore in this paper. We train a standard Soft Actor-Critic (SAC) agent with an auxiliary model of the Predictive Information. Together, these pieces give us Predictive Information Soft Actor-Critic (PI-SAC).

The main contributions of this paper are:

- **PI-SAC:** A description of the core PI-SAC agent (Section 3).
- **Sample Efficiency:** We demonstrate strong gains in sample efficiency on nine tasks from the DM Control Suite [42] of continuous control tasks, compared to state-of-the-art baselines such as Dreamer [17] and DrQ [27] (Section 4.1).
- **Ablations:** Through careful ablations and analysis, we show that the benefit of PI-SAC is due substantially to the use of the Predictive Information and compression (Section 4.2).
- **Generalization:** We show that compressed representations outperform uncompressed representations in generalization to unseen tasks (Section 4.3).

2 Preliminaries

Predictive Information. The *Predictive Information* [4] is the mutual information between the past and the future, $I(\text{past}; \text{future})$. From here on, we will denote the past by X and the future by Y. [4] shows that the entropy of the past, $H(X)$, is a quantity that grows much faster than the Predictive Information, $I(X;Y)$, as the duration of past observations increases. Consequently, if we would like to represent only the information in X that is relevant for predicting Y, we should prefer a *compressed* representation of X. This is a sharp distinction with techniques such as Contrastive Predictive Coding (CPC) [30] and Deep InfoMax (DIM) [19] which explicitly attempt to maximize a lower bound on $I(X;Y)$ without respect to whether the learned representation has compressed away irrelevant information about X.

The Conditional Entropy Bottleneck. In order to learn a compressed representation of the Predictive Information, we will use the Conditional Entropy Bottleneck (CEB) [7] objective. CEB attempts to learn a representation Z of some observed variable X such that Z is as useful as possible for predicting a target variable Y, i.e. maximizing $I(Y;Z)$, while compressing away any information from X that is not also contained in Y, i.e. minimizing $I(X;Z|Y)$:

$$\begin{aligned}
CEB &\equiv \min_Z \beta I(X;Z|Y) - I(Y;Z) && (1)\\
&= \min_Z \beta(-H(Z|X) + H(Z|Y)) - I(Y;Z) && (2)\\
&= \min_Z \mathbb{E}_{x,y,z\sim p(x,y)e(z|x)}\, \beta \log \frac{e(z|x)}{p(z|y)} - I(Y;Z) && (3)\\
&\leq \min_Z \mathbb{E}_{x,y,z\sim p(x,y)e(z|x)}\, \beta \log \frac{e(z|x)}{b(z|y)} - I(Y;Z) && (4)
\end{aligned}$$

Here, $e(z|x)$ is the true *encoder* distribution where we sample z from; $b(z|y)$ is the variational *backwards encoder* distribution that approximates the unknown true distribution $p(z|y)$. Both can be parameterized by the outputs of neural networks. Compression increases as β goes from 0 to 1.

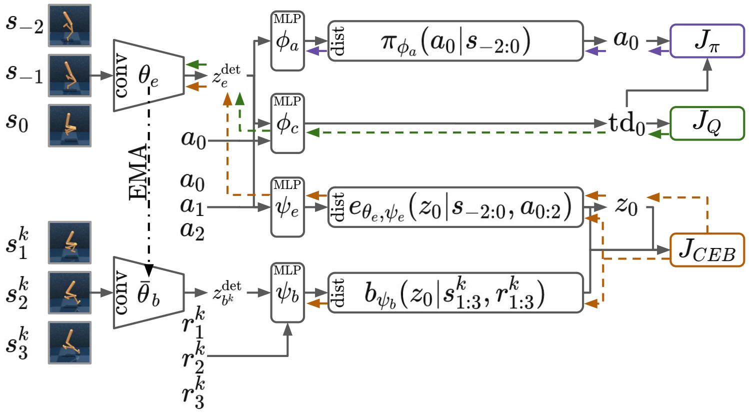

Figure 1: PI-SAC system diagram for a single minibatch example. To compute J_{CEB} requires K $b_{\psi_b}(\cdot)$ distributions from the minibatch, as described in Section 2. Colored edges show how gradients flow back to model weights.

To get a variational lower bound on the $I(Y;Z)$ term, we will use the *CatGen* formulation from [7], which is equivalent to the InfoNCE bound [30, 32] but reuses the backwards encoder:

$$I(Y;Z) \geq \mathbb{E}_{x,y,z\sim p(x,y)e(z|x)} \log \frac{b(z|y)}{\frac{1}{K}\sum_{k=1}^{K} b(z|y^k)} \tag{5}$$

We write the objective for a single example in a minibatch of size K to simplify notation. The K examples are sampled independently. Altogether, this gives us:

$$CEB \leq \min_{Z} \mathbb{E}_{x,y,z\sim p(x,y)e(z|x)}\, \beta \log \frac{e(z|x)}{b(z|y)} - \log \frac{b(z|y)}{\frac{1}{K}\sum_{k=1}^{K} b(z|y^k)} \tag{6}$$

Soft Actor-Critic. Soft Actor-Critic (SAC) [16] is an off-policy algorithm that learns a stochastic policy π_{ϕ_a}, a Q-value function Q_{ϕ_c}, and a temperature coefficient α to find an optimal control policy. It maximizes a γ-discounted return objective based on the Maximum Entropy Principle [45, 44, 15, 24]. SAC has objectives for the critic, the actor, and the temperature parameter, α. The critic minimizes:

$$J_Q(\phi_c) = \mathbb{E}_{s_t,a_t\sim\mathcal{D}}\, \frac{1}{2}\Big(Q_{\phi_c}(s_t,a_t) - \big(r(s_t,a_t) + \gamma\, \mathbb{E}_{s_{t+1}\sim p}\, V_{\bar{\phi}_c}(s_{t+1})\big)\Big)^2 \tag{7}$$

where $V_{\bar{\phi}_c}(s_t) \equiv \mathbb{E}_{a_t\sim\pi(a_t|s_t)}\, Q_{\bar{\phi}_c}(s_t,a_t) - \alpha \log \pi(a_t|s_t)$ is a value function that uses an exponential moving average of the ϕ_c parameters, $\bar{\phi}_c$. The actor minimizes:

$$J_\pi(\phi_a) = \mathbb{E}_{s_t\sim\mathcal{D}}\, \mathbb{E}_{a_t\sim\pi(a_t|s_t)}\, \alpha \log \pi(a_t|s_t) - Q_{\phi_c}(s_t,a_t) \tag{8}$$

Given $\mathcal{H}$, a target entropy for the policy distribution, the temperature α is learned by minimizing:

$$J_\alpha(\alpha) = \mathbb{E}_{a_t\sim\pi(a_t|s_t)} -\alpha \log \pi(a_t|s_t) - \alpha\mathcal{H} \tag{9}$$

SAC learns two critics $Q_{\phi_c^1}, Q_{\phi_c^2}$ and maintains two target critics $Q_{\bar{\phi}_c^1}, Q_{\bar{\phi}_c^2}$ for double Q-learning [9], but we omit it in our notation for simplicity and refer readers to the SAC paper [16] for details.

3 Predictive Information Soft Actor-Critic (PI-SAC)

A natural way to combine a stochastic latent variable model trained with CEB with a model-free RL algorithm like SAC is to use the latent representation at timestep t, z_t, as the state variable for the actor, the critic, or both. We will call this *Representation* PI-SAC and define it in Section H. However, any representation given to the actor cannot have a dependency on the next action, and any representation given to the critic can depend on at most the next action, since during training and evaluation, the actor must use the representation to decide what action to take, and the critic needs the representation to decide how good a particular state and action are. The CEB model's strength, on the

Algorithm 1 Training Algorithm for PI-SAC

```
Require: $E_{\text{step}}, \theta_e, \phi_c^1, \phi_c^2, \phi_a, \alpha, \psi_e, \psi_b$    ▷ Environment and initial parameters
  $\bar{\theta}_b \leftarrow \theta_e, \bar{\theta}_e \leftarrow \theta_e$    ▷ Copy initial forward conv weights to the backward and target critic conv encoder
  $\mathcal{D} \leftarrow \emptyset$    ▷ Initialize replay buffer
  for each initial collection step do    ▷ Initial collection with random policy
    $a_t \sim \pi_{\text{random}}(a_t)$    ▷ Sample action from a random policy
    $s_{t+1}, r_{t+1} \sim E_{\text{step}}(a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s_{t+1}, a_t, r_{t+1})$
  end for
  $s_1 \leftarrow E_{\text{step}}()$    ▷ Get initial environment step
  for t=1 to M do
    $a_t \sim \pi_{\phi_a}(a_t|s_t)$    ▷ Sample action from the policy
    $s_{t+1}, r_{t+1} \sim E_{\text{step}}(a_t)$    ▷ Sample next observation from environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup (s_{t+1}, a_t, r_{t+1})$
    for each gradient step do
      $\{\phi_c^i, \theta_e\} \leftarrow \{\phi_c^i, \theta_e\} - \lambda_Q \hat{\nabla}_{\{\phi_c^i, \theta_e\}} J_Q(\phi_c^i, \theta_e)$ for $i \in \{1, 2\}$    ▷ gradient step on critics
      $\phi_a \leftarrow \phi_a - \lambda_\pi \hat{\nabla}_{\phi_a} J_\pi(\phi_a)$    ▷ gradient step on actor
      $\alpha \leftarrow \alpha - \lambda_\alpha \hat{\nabla}_\alpha J_\alpha(\alpha)$    ▷ adjust temperature
      $\{\theta_e, \psi_e, \psi_b\} \leftarrow \{\theta_e, \psi_e, \psi_b\} - \lambda_{CEB} \hat{\nabla}_{\{\theta_e, \psi_e, \psi_b\}} J_{CEB}(\theta_e, \psi_e, \psi_b)$    ▷ CEB gradient step
      $\bar{\phi}_c^i \leftarrow \tau \phi_c^i - (1 - \tau)\bar{\phi}_c^i$ for $i \in \{1, 2\}$    ▷ Update target critic network weights
      $\bar{\theta}_e \leftarrow \tau \theta_e - (1 - \tau)\bar{\theta}_e$    ▷ Update target critic conv encoder weights
      $\bar{\theta}_b \leftarrow \tau_b \theta_e - (1 - \tau_b)\bar{\theta}_b$    ▷ Update backward conv encoder weights
    end for
  end for
```

other hand, lies in capturing a representation of the dynamics of the environment multiple steps into the future. We may therefore hypothesize that using CEB as an auxiliary loss can be more effective, since in that setting, the future prediction task can be conditioned on the actions taken at each future frame. Conditioning on multiple future actions in the forward encoder allows it to make more precise predictions about the future states, thereby allowing the forward encoder to more accurately model environment dynamics.[2] Consequently, PI-SAC agents are trained using CEB as an auxiliary task, as shown in Figure 1.

PI-SAC uses the same three objective functions from SAC, described above. The only additional piece to specify is the choice of X and Y for the CEB objective. In our setting, X consists of previous observations and future actions, and Y consists of future observations and future rewards. If we define the present as $t = 0$ and we limit ourselves to observations from $-T + 1$ to T, we have:

$$\begin{aligned} J_{CEB}(\theta_e, \psi_e, \psi_b) = \mathbb{E}_{s_{-T+1:T}, a_{0:T-1}, r_{1:T} \sim \mathcal{D}, z_0 \sim e(z_0|\cdot)} \log \frac{e_{\theta_e, \psi_e}(z_0 | s_{-T+1:0}, a_{0:T-1})}{b_{\psi_b}(z_0 | s_{1:T}, r_{1:T})} \\ + \log \frac{b_{\psi_b}(z_0 | s_{1:T}, r_{1:T})}{\frac{1}{K} \sum_{k=1}^{K} b_{\psi_b}(z_0 | s_{1:T}^k, r_{1:T}^k)} \end{aligned} \quad (10)$$

The training algorithm for PI-SAC is in Algorithm 1. E_{step} is the environment step function. θ_e is the weight vector of the convolutional encoder. $\bar{\theta}_b = \text{EMA}(\theta_e, \tau_b)$ is the weight vector of the convolutional backwards encoder, where EMA$(\cdot)$ is the exponential moving average function. ϕ_c^1, ϕ_c^2, and ϕ_a are the weight vectors for two critic networks and the actor network, respectively. We let the critic gradients back-propagate through the shared conv encoder e_{θ_e}, but stop gradients from actor. The target critics use a shared conv encoder parameterized by $\bar{\theta}_e$ which is an exponential moving average of e_{θ_e}, similar to updating the target critics. α is the SAC temperature parameter. ψ_e and ψ_b are the weight vectors of MLPs to parameterize the CEB forward and backwards encoders. τ and τ_b are exponents for EMA calls. λ_Q, λ_π, λ_α, and λ_{CEB} are learning rates for the four different objective functions. See Section A for implementation details.

[2] We give details and results for Representation PI-SAC agents in Section H. Representation PI-SAC agents are also very sample efficient on most tasks we consider, but they don't achieve as strong performance on tasks with more complicated environment dynamics, such as Cheetah, Hopper, and Walker.

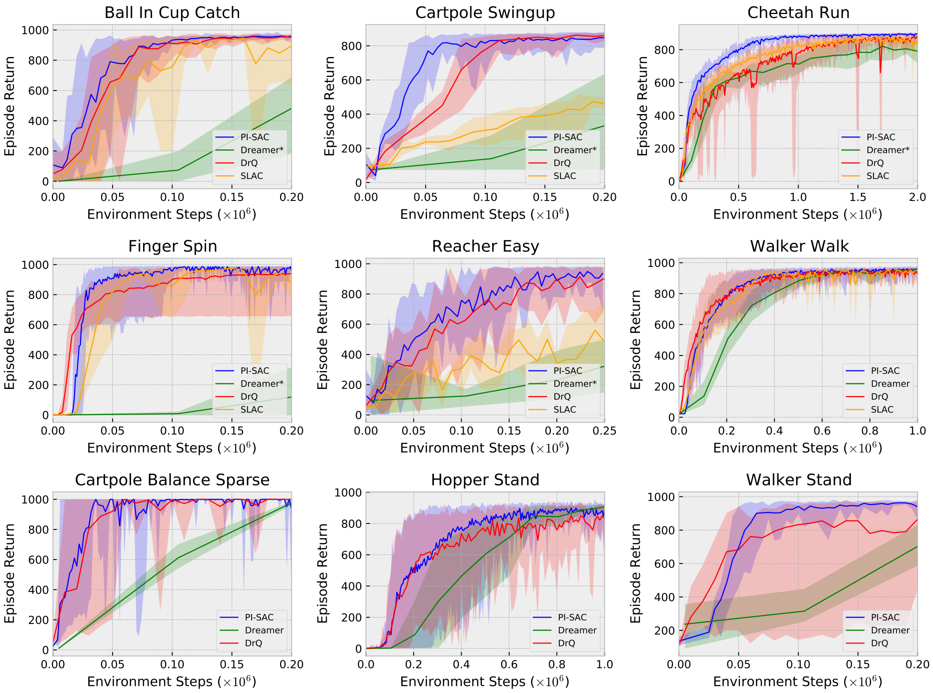

Figure 2: Performance comparison to existing methods on 9 tasks from DeepMind control suite. The upper 6 tasks are the PlaNet benchmark [18]. *Dreamer** indicates that the other agents do not use Dreamer's action repeat of 2. We additionally include the 3 lower tasks with a fixed action repeat of 2 to compare with Dreamer [17] and DrQ [27] results on the Dreamer benchmark. PI-SAC matches the state-of-the-art performance on all 9 tasks and is consistently the most sample efficient.

4 Experiments

We evaluate PI-SAC on the DeepMind control suite [42] and compare with leading model-free and model-based approaches for continuous control from pixels: SLAC [29], Dreamer [17], CURL [39] and DrQ [27]. Our benchmark includes the six tasks from the PlaNet benchmark [18] and three additional tasks: Cartpole Balance Sparse, Hopper Stand, and Walker Stand.

The PlaNet benchmark treats action repeat as a hyperparameter. On each PlaNet task, we evaluate PI-SAC with the action repeat at which SLAC performs the best, and compare with the best DrQ and CURL results. The choices of action repeat are listed in Section A.2. On Walker Walk (also in the PlaNet benchmark), Cartpole Balance Sparse, Hopper Stand, and Walker Stand, we evaluate PI-SAC with action repeat 2 and directly compare with Dreamer and DrQ results on the Dreamer benchmark. We report the performance using true environment steps to be invariant to action repeat. All figures show mean, minimum, and maximum episode returns of 10 runs unless specified otherwise.

Throughout these experiments we mostly use the standard SAC hyperparameters [16], including the sizes of the actor and critic networks, learning rates, and target critic update rate. Unless otherwise specified, we set CEB $\beta = 0.01$. We report our results with the best number of gradient updates per environment step in Section 4.1, and use one gradient update per environment step for the rest of the experiments. Full details of hyperparameters are listed in Section A.2. We use an encoder architecture similar to DrQ [27]; the details are described in Section A.1.

4.1 Sample Efficiency

Figure 2 and Table 1 compare PI-SAC with SLAC, Dreamer, DrQ, and CURL[3]. PI-SAC consistently achieves state-of-the-art performance and better sample efficiency across all benchmark tasks, better

[3] SLAC, Dreamer, and DrQ learning curves were provided to us by the authors.

100k step scores	PI-SAC	CURL	DrQ	SLAC
Ball in Cup Catch	**933±16**	769±43	913±53	756±314
Cartpole Swingup	**816±72**	582±146	759±92	305±66
Cheetah Run	**460±93**	299±48	344±67	344±69
Finger Spin	**957±45**	767±56	901±104	859±132
Reacher Easy	**758±167**	538±233	601±213	305±134
Walker Walk	514±89	403±24	**612±164**	541±98
Cartpole Balance Sparse	**1000±0**	-	**999±2**	-
Hopper Stand	**97±147**	-	87±152	-
Walker Stand	**942±21**	-	832±259	-
500k step scores	PI-SAC	CURL	DrQ	SLAC
Cheetah Run	**801±23**	518±28	660±96	715±24
Reacher Easy	**950±45**	929±44	942±71	688±135
Walker Walk	**946±18**	902±43	921±45	938±15
Hopper Stand	**821±166**	-	750±140	-

Table 1: Returns at 100k and 500k environment steps. We only show results at 500k steps for tasks on which PI-SAC is not close to convergence at 100k steps. We omit Dreamer's results using action repeat of 2 (most of the scores are significantly lower as shown in Figure 2).

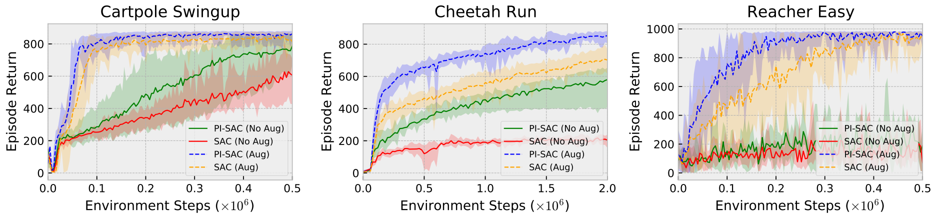

Figure 3: The predictive information improves performance on Cartpole Swingup and Cheetah Run without any data augmentation. With data augmentation, it continues showing strong improvements over the SAC baseline on all three tasks. We perform 5 runs for PI-SAC and SAC without augmentation. More results are presented in Section F.

than or at least comparable to all the baselines. We report our results on Reacher Easy with one gradient update per environment step, on Cheetah Run with four gradient updates, and the rest with two gradient updates. A comparison of PI-SAC agents with different numbers of gradient updates is available in Section B. The comparison in this section is system-to-system as all baseline methods have their own implementation advantages: SLAC uses much larger networks and 8 context frames, making its wall clock training time multiple times slower; DrQ and CURL's SAC differs substantially from the standard SAC [16], including having much larger actor and critic networks; Dreamer is a model-based method that uses RNNs and learns a policy in simulation.

4.2 Predictive Information

We test our hypothesis that predictive information is the source of the sample efficiency gains here.

Data Augmentation. We follow [27] to train our models with image sequences randomly shifted by $[-4, 4]$ pixels. Without this perturbation, Figure 3 shows that learning the predictive information by itself still greatly improves agents' performance on Cartpole and Cheetah but makes little difference on Reacher. Learning PI-SAC with data augmentation continues showing strong improvements over the SAC baseline with data augmentation and solves all benchmark tasks.

[27, 28] showed that input perturbation facilitates actor-critic learning, and we show that it also improves CEB learning. As described in Section 2, we use the contrastive CatGen formulation to get a variational lower bound on $I(Y; Z)$. Because of its contrastive nature, CatGen can ignore information that is not required for it to distinguish different samples and still saturate its bound. In our experiments without data augmentation, we found that it ignores essential information for solving

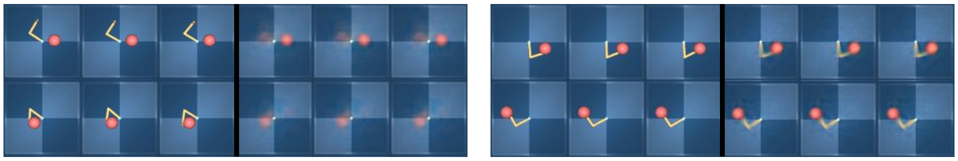

Figure 4: We learn a diagnostic deconvolutional decoder to predict future observations from CEB representations learned along with PI-SAC for Reacher. We show ground truth future observations and the predicted future observations from CEB representations. **Left:** CEB representations learned **without** data augmentation only capture positions of the target. **Right:** CEB representations learned **with** data augmentation capture both the target and the arm.

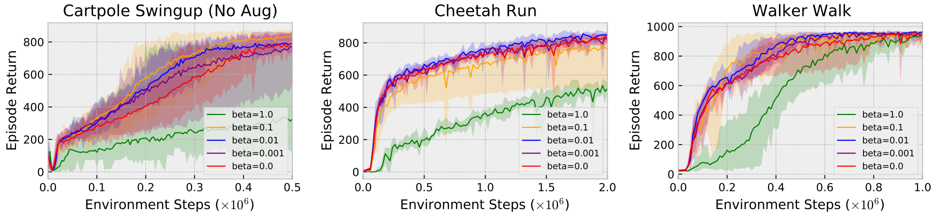

Figure 5: Compression improves agents' performance. We disable data augmentation for Cartpole Swingup to amplify the benefit of compression. At $\beta = 0$, compression is not part of the learning objective. We perform 5 runs for each curve shown in this figure.

Reacher. We train a deconvolutional decoder to diagnostically predict future frames from CEB representations (we stop gradients from the decoder). As shown in Figure 4, CEB representations learned without input perturbation completely fail to capture the arm's pose. This is because CatGen can perfectly distinguish frame sequences in a minibatch sampled from the replay buffer by only looking at the position of the target, since that is constant in each episode but varies between episodes. In contrast, CatGen representations learned with randomly shifted images successfully capture both the target and the arm. This observation suggests that appropriate data augmentation helps CatGen to capture meaningful information for control.

Compression. As described in Section 2, we compress the residual information $I(X; Z|Y)$ out to preserve the minimum necessary predictive information. Figure 5 studies the trade-off between strength of compression and agents' performance by sweeping β values. Some amount of compression improves sample efficiency and stability of the results, but overly strong compression can be harmful. The impact of β on the agent's performance confirms that, even though the CEB representation isn't being used directly by the agent, the auxiliary CEB objective is able to substantially change the agent's weights. Sweeping β allows us to explore the frontier of the agent's performance, as well as the Pareto-optimal frontier of the CEB objective as usual [7]. For example, for Cheetah Run, the residual information at the end of training ranges between ~ 0 nats for $\beta = 1$, to ~ 947 nats for $\beta = 0$. For the top performing agent, $\beta = 0.01$, the residual information was ~ 6 nats.

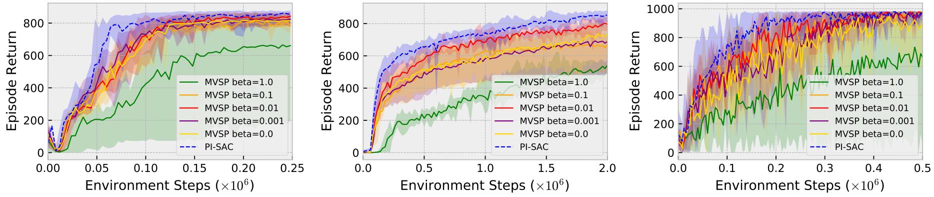

Figure 6: Learning the predictive information outperforms multiview self-prediction (MVSP) at all levels of compression. In this figure we show MVSP agents that predict future rewards conditioning on actions. The MVSP curves show results of 5 runs.

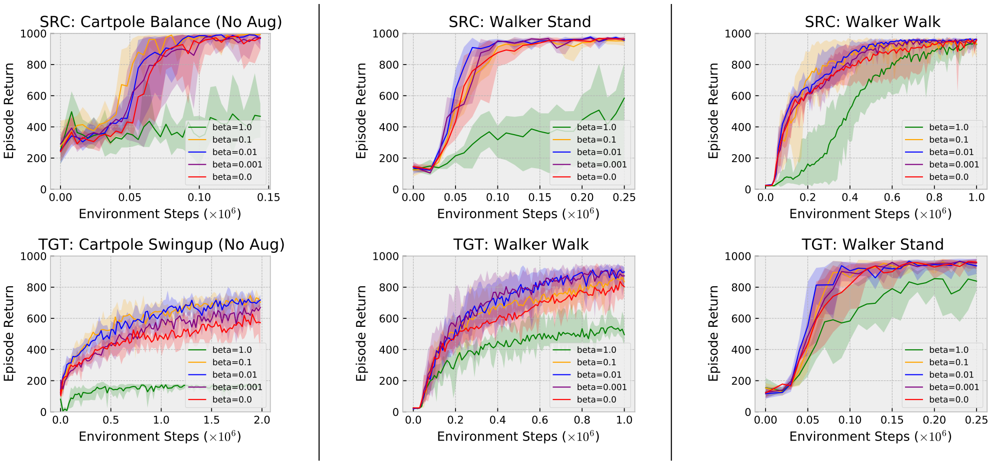

Figure 7: Compression improves task transfer. We train a PI-SAC agent on a source task (SRC), freeze the representation z_e^{det} (see Figure 1), and train a new agent on a target task (TGT). We show the results of normal PI-SAC agents with different β for the source tasks, and the transfer results using representations learned on the source tasks with different β for the target tasks. Compression substantially improves task transfer when the target task is intuitively more difficult than the source, i.e. Cartpole Balance to Swingup and Walker Stand to Walk. The difference is less significant when the target task is presumably easier, i.e. Walker Walk to Stand. We disable data augmentation for the Cartpole experiment to amplify the benefit of compression. All curves show 5 runs.

Comparison to Multiview Self-Prediction. Multiview Self-Prediction (MVSP) is an auxiliary objective used by CURL [39]. CURL uses the InfoNCE bound to capture the mutual information between two random crops $I(X_{\text{crop1}}; X_{\text{crop2}})$ as an auxiliary task for continuous control from pixels. This approach preserves information about the present, differing philosophically from PI-SAC which captures information about the future. By changing the CEB prediction target from the future (Y) to a random shift of the past observation, X', we can achieve the equivalent multiview self-prediction in our framework and fairly compare the two approaches. Figure 6 shows that PI-SAC agents outperform MVSP agents at all levels of compression. But similar to PI-SAC, compression also helps MVSP agents on tasks like Cheetah and Reacher. This empirical evidence suggests that, for RL agents, knowing what will happen in the future matters more than knowing what has happened in the past. Note that Figure 6 shows MVSP agents that predict future rewards conditioning on actions to enable a direct comparison to PI-SAC agents. More results of PI-SAC and MVSP agents with and without future reward prediction on all tasks can be found in Section D.

4.3 Generalization to Unseen Tasks

It is well-known that compressed representations can generalize better in many machine learning settings [36, 3, 8], including RL [20, 12, 11]. In addition to sample efficiency, for more testing of generalization, we explore transferring representations to an unseen task with the same environment dynamics. Specifically, we learn a PI-SAC agent on a source task, freeze the representation z_e^{det} (see Figure 1), and train a new agent for a target task using the frozen representation. Figure 7 shows that compressed representations generalize substantially better to unseen tasks than uncompressed representations. Especially when the target task is intuitively harder than the source task, i.e. Cartpole Balance to Swingup and Walker Stand to Walk, the performance differences between different levels of compression are more significant on the target task than on the original tasks. It is, however, less prominent when the target task is easier, i.e. Walker Walk to Stand. Our conjecture is that solving the original Walk task would require exploring a wider range of the environment dynamics that presumably includes much what the Stand task would need. On the other hand, transferring from Stand to Walk requires generalization to more unseen part of the environment dynamics. Note that in these settings it is still more sample efficient to train a full new PI-SAC agent on the target task. These experiments simply demonstrate that the more compressed predictive information models have representations that are more useful in a task transfer setting.

5 Related Work

Future Prediction in RL. Future prediction is commonly used in RL in a few different ways. Model-based RL algorithms build world model(s) to predict the future conditioned on the past and actions, and then act through planning [6, 14, 17, 18, 23, 25, 41]. Some of the curiosity learning approaches reward an agent based on future prediction error or uncertainty [5, 31, 40]. This work falls in another group, where future prediction is used as an auxiliary or representation learning method for model-free RL agents [1, 10, 14, 22, 29, 30, 33, 34, 35, 37]. We hypothesize that the success of these methods comes from the predictive information they capture. In contrast to prior work, our approach directly measures and compresses the predictive information, so that the representation avoids capturing the large amount of information in the past that is irrelevant to the future. As described in Section 2, the predictive information that we consider captures environment dynamics. This is different from some other approaches [1, 30] that use contrastive methods (e.g. InfoNCE) to capture temporal coherence of observations instead of dynamics (since their predictions are not action-conditioned) and thus have their limitations in off-policy learning. See Section E for discussion and results without conditioning on actions.

Other Types of Representation for RL. Besides future prediction, previous contributions have investigated other representation learning methods for RL. An example is inverse dynamics [5, 31] which learns representations through predicting actions given the current and next observed states. In contrast to future prediction, inverse dynamics only reflects what the agent can immediately affect, and thus may not be sufficient [5]. Another example is learning representations through retaining information about the past observations using reconstructive or contrastive approaches [39, 43], but that does not lean toward capturing information relevant for achieving the future and goal. We show that future prediction is empirically superior in Section 4.2 and Section D.

Continuous Control from Pixels. Recent successes in continuous control from visual observations can roughly be classified into two groups: model-based and model-free approaches. Prominent model-based approaches like PlaNet [18] and Dreamer [17], for example, learn latent dynamics and perform planning in latent space. Several recent works with the model-free approach [27, 39, 28] have demonstrated that image augmentation improves sample-efficiency and robustness of model-free continuous control from pixels. In this work, we also perform image augmentation and discover its value in learning the predictive information with a contrastive loss (Section 4.2). An example falling between model-based and model-free is SLAC [29] which captures latent dynamics for representation learning but learns the model-free SAC [16] with the representation. The approach is different from ours as PI-SAC does not model roll-outs in latent space.

6 Conclusion

We presented Predictive Information Soft Actor-Critic (PI-SAC), a continuous control algorithm that trains a SAC agent using an auxiliary objective that learns a compressed representation of the predictive information of the RL environment dynamics. We showed with extensive experiments that learning a compressed predictive information representation can substantially improve sample efficiency and training stability at no cost to final agent performance. Furthermore, we gave preliminary indications that compressed representations can generalize better than uncompressed representations at task transfer. Future work will explore variations of the PI-SAC architecture, such as using RNNs for environments that require long-term planning.

Broader Impact

This work attempts to expand the applicability of RL to visual inputs and expand the range of RL applications, especially in Robotics. In its current form this work is applied to simulated environments, but thanks to the improvements in sample efficiency and generalization it increases the possibility of training directly on real-world robots.

However, advancements in robotic automation likely have complex societal impacts. One potential risk is creating shifts in skill demand and thus structural unemployment. Improving RL autonomous agents' applicability to visual inputs is a potential threat to a range of employment types, for example, in the manufacturing industry. Public policy and regulation support will be necessary to reduce societal and economic friction as techniques like these are deployed in the physical world. On the other hand, potential positive outcomes from continued RL improvements include replacement of human workers at high-risk workspace, reduction of repetitive operations, and productivity increases.

We see opportunities that researchers and engineers would benefit from adding CEB and the Predictive Information to the list of tools in RL. We would like to note that designing RL tasks and reward functions can have potential biases if applied to real systems that interact with users. Therefore we encourage further research to understand the impacts, implications, and limitations of using this work in real-world scenarios.

Acknowledgments and Disclosure of Funding

We thank Justin Fu, Anoop Korattikara, and Ed Chi for valuable discussions. Our thanks also go to Toby Boyd for his help in making the code open source. Furthermore, we thank Danijar Hafner, Alex Lee, Ilya Kostrikov and Denis Yarats for sharing performance data for the Dreamer [17], SLAC [29], and DrQ [27] baselines.

John Canny is associated with both Google Research and University of California, Berkeley. Honglak Lee is associated with both Google Research and University of Michigan. Honglak would like to acknowledge NSF CAREER IIS-1453651 for partial support.

References

[1] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems*, pages 8766–8779, 2019.

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

[3] Raef Bassily, Shay Moran, Ido Nachum, Jonathan Shafer, and Amir Yehudayoff. Learners that use little information. In *Algorithmic Learning Theory*, pages 25–55, 2018.

[4] William Bialek and Naftali Tishby. Predictive information. *arXiv preprint cond-mat/9902341*, 1999.

[5] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *International Conference on Learning Representations*, 2019.

[6] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[7] Ian Fischer. The conditional entropy bottleneck. *arXiv preprint arXiv:2002.05379*, 2020.

[8] Ian Fischer and Alexander A Alemi. CEB improves model robustness. *arXiv preprint arXiv:2002.05380*, 2020.

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

[10] 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*, pages 2170–2179, 2019.

[11] Anirudh Goyal, Yoshua Bengio, Matthew Botvinick, and Sergey Levine. The variational bandwidth bottleneck: Stochastic evaluation on an information budget. In *International Conference on Learning Representations*, 2020.

[12] Anirudh Goyal, Riashat Islam, Daniel Strouse, Zafarali Ahmed, Matthew Botvinick, Hugo Larochelle, Yoshua Bengio, and Sergey Levine. Infobot: Transfer and exploration via the information bottleneck. In *International Conference on Learning Representations*, 2019.

[13] Sergio Guadarrama, Anoop Korattikara, Oscar Ramirez, Pablo Castro, Ethan Holly, Sam Fishman, Ke Wang, Ekaterina Gonina, Neal Wu, Efi Kokiopoulou, Luciano Sbaiz, Jamie Smith, Gábor Bartók, Jesse Berent, Chris Harris, Vincent Vanhoucke, and Eugene Brevdo. TF-Agents: A library for reinforcement learning in tensorflow. `https://github.com/tensorflow/agents`, 2018. [Online; accessed 25-June-2019].

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

[15] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361, 2017.

[16] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[17] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2019.

[18] 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*, pages 2555–2565, 2019.

[19] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[20] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck. In *Advances in Neural Information Processing Systems*, pages 13978–13990, 2019.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[22] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*, 2017.

[23] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[24] Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

[25] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[27] 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.

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

[29] 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*, 2019.

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

[31] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[32] Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180, 2019.

[33] Jürgen Schmidhuber. Making the world differentiable: On using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical report, Institut für Informatik, Technische Universität München, 1990.

[34] Jürgen Schmidhuber. Reinforcement learning in markovian and non-markovian environments. In *Advances in Neural Information Processing Systems*, pages 500–506, 1991.

[35] Jürgen Schmidhuber. On learning to think: Algorithmic information theory for novel combinations of reinforcement learning controllers and recurrent neural world models. *arXiv preprint arXiv:1511.09249*, 2015.

[36] Ohad Shamir, Sivan Sabato, and Naftali Tishby. Learning and generalization with the information bottleneck. *Theoretical Computer Science*, 411(29-30):2696–2711, 2010.

[37] Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

[38] Saurabh Singh and Shankar Krishnan. Filter response normalization layer: Eliminating batch dependence in the training of deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11237–11246, 2020.

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

[40] Susanne Still and Doina Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

[41] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[42] 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.

[43] 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.

[44] Brian D Ziebart. *Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy*. PhD thesis, Carnegie Mellon University, 2010.

[45] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Appendices

A PI-SAC Implementation

Initial CEB training steps. After collecting initial experiences with a random policy (in Algorithm 1), we optionally pre-train with the predictive information CEB objective ($\theta_e, \bar{\theta}_e, \psi_e, \psi_b$ are updated). The amount of initial CEB steps are selected empirically for each task (listed in Table 2).

Observation Horizon and Frame Stacking. As described in Section 3, we limit our observations from $-T+1$ to T (following [39, 27], we set $T = 3$). We construct the observational input to the encoder and backward encoder as a T-stack of consecutive frames, where each frame is a $[84 \times 84 \times 3]$ RGB image rendered from the 0th DMControl camera. The pixel values range from $[0, 255]$, and we divide each pixel by 255.0.

Evaluation Setups. We evaluate our agent at every evaluation point by computing the average episode return over 10 evaluation episodes. At test time, our policy is deterministic and uses the mean of the policy distribution. For most of the experiments, we evaluate every 2500 environment steps after applying action repeat for Cheetah, Walker, and Hopper tasks. For Ball in Cup, Cartpole, Finger, and Reacher tasks, we evaluate every 1000 environment steps after applying action repeat.

SAC Implementation. Our SAC implementation is based off of TF-Agents [13]. It follows the standard SAC implementation [16]. The performance and sample-efficiency match with the benchmark results reported in [16].

A.1 Network Architecture

Encoder Networks. The convolutional encoder architecture consists of four convolution layers with 3×3 kernels, 32 channels, similar to the encoder architecture being used in [43, 39, 27]. We use stride 2 at the first convolution layer and 1 in the rest. Filter Response Normalization and Thresholded Linear Unit [38] are applied after each convolution layer. The output of the last convolution layer is fed into a fully-connected layer which projects to a 50-d feature vector and followed by Layer Normalization [2]. This gives us the 50-d z_e^{det}. As shown in Figure 1, we stop gradients from the actor network, but allow the critic optimizer and the CEB optimizer to update the convolutional encoder.

Actor and Critic Networks. Implementations of actor and critic follow the standard SAC [16]. Both actor and critic are parameterized by MLPs with two 256-d hidden layers. The actor network outputs mean and covariance for a parametric Gaussian distribution. And we use $\tanh$ as an invertible squashing function to enforce the action bounds as in [16]. Inputs to the critic network is a concatenation of z_e^{det} and action.

CEB Auxiliary Model. Our CEB forward and backward MLP encoders are parameterized by MLPs with two 128-d hidden layers. Each MLP outputs 50-d mean followed by Batch Normalization [21] for a multivariate Gaussian distribution, and we fix diagonal covariance at 1.0. Inputs to the forward MLP is a concatenation of z_e^{det} and T future actions. Inputs to the backward MLP is a concatenation of z_e^{det} and T future rewards.

A.2 Hyperparameters

Throughout these experiments we mostly use the standard SAC hyperparameters [16]. The hyperparameters that are fixed across all tasks are listed in Table 2. The size of the replay buffer is a smaller 10^5 due to high memory usage for storing image observations. The heuristic entropy target is set to $-dim(A)/2$, a default value used in TF-Agents SAC implementation [13], where $dim(A)$ is number of dimensions of action. In our experiments the results are similar to using $-dim(A)$, the default used in [16].

The amount of action repeat (described in Section 4), initial collection steps, and initial CEB steps for each task are listed in Table 2. The standard SAC takes 10,000 initial collection steps, but for some of the tasks we take fewer steps of 1,000 in favor of sample efficiency.

Table 2: **Left:** Global PI-SAC hyperparameters. **Right:** Per-task PI-SAC hyperparameters. PlaNet tasks are indicated with (P).

Parameter	Value
optimizer	Adam [26]
batch size	256
actor learning rate λ_π	3×10^{-4}
critic learning rate λ_Q	3×10^{-4}
alpha learning rate λ_α	3×10^{-4}
CEB learning rate λ_{CEB}	3×10^{-4}
discount (γ)	0.99
replay buffer size	10^5
entropy target $\mathcal{H}$	$-dim(A)/2$
target smoothing coefficient (τ)	0.005
target update interval	1
initial $log(\alpha)$	0.0
backward encoder EMA update rate	0.05
observation horizon T	3

Task	Action Repeat	Initial Collection Steps	Initial CEB steps
Cartpole Swingup (P)	4	1000	5000
Cartpole Balance Sparse	2	1000	5000
Reacher Easy (P)	4	1000	5000
Ball in Cup Catch (P)	4	1000	5000
Finger Spin (P)	1	10000	0
Cheetah Run (P)	4	10000	10000
Walker Walk (P)	2	10000	10000
Walker Stand	2	10000	10000
Hopper Stand	2	10000	10000

B PI-SAC and SAC at Different Numbers of Gradient Steps

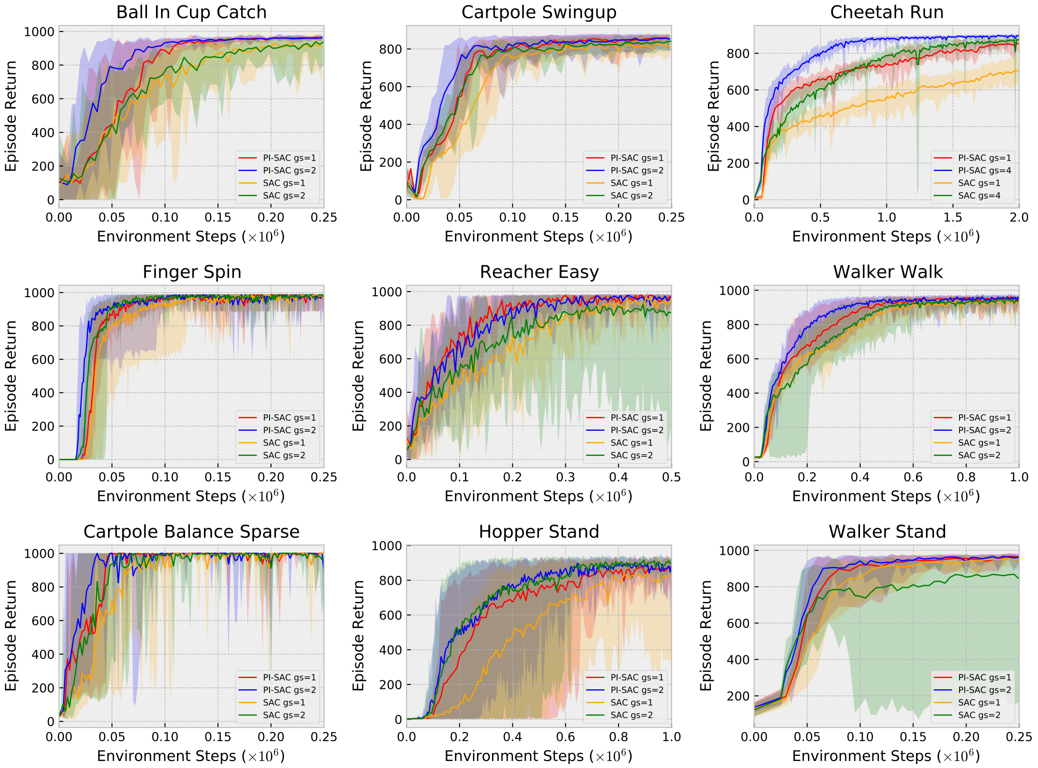

Figure 8: Comparison of PI-SAC and SAC with image augmentation at different numbers of gradient steps (gs) per environment step. We report results at 1 and 2 gradient steps, except that we show 1 and 4 gradient steps for Cheetah Run. PI-SAC consistently outperforms the SAC baseline.

Another way to improve sample efficiency in SAC and PI-SAC models is to increase the number of gradient steps taken per environment step collected. In Figure 8, we see that PI-SAC outperforms the SAC baseline while varying gradient steps, particularly on Ball In Cup Catch and Cheetah Run.

C Comparison to SAC from States

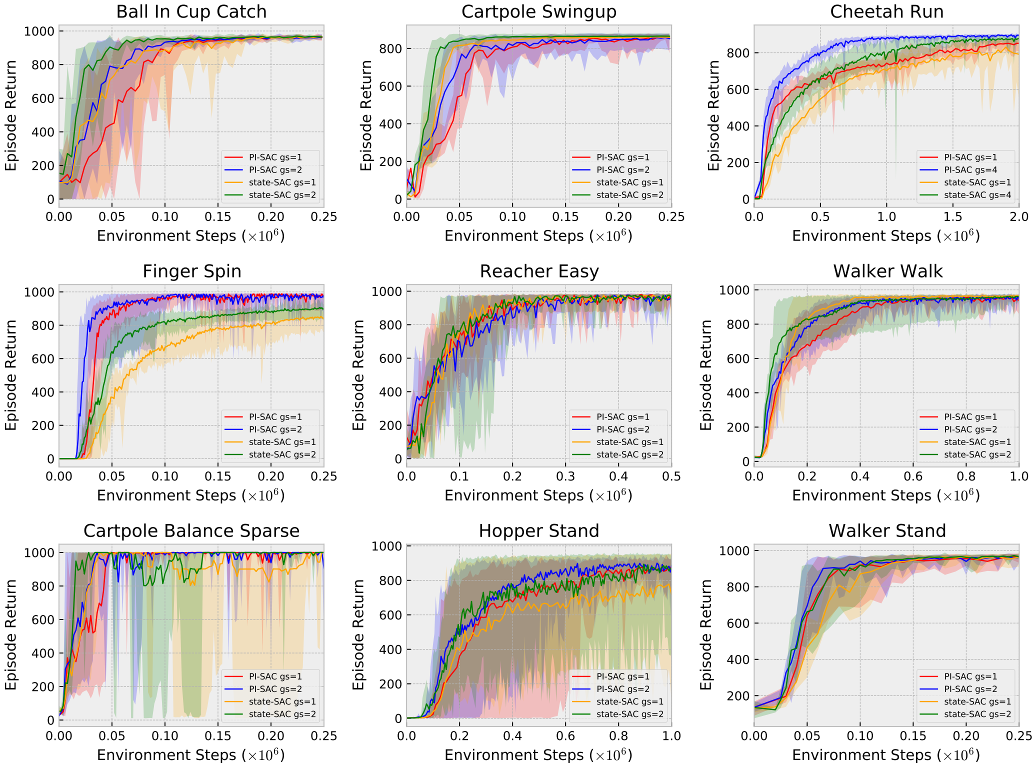

Figure 9: Comparison of PI-SAC (from pixels) to SAC from states (state-SAC) at different numbers of gradient steps (gs) per environment step. We report results at 1 and 2 gradient steps for all tasks except Cheetah Run, which uses 1 and 4 gradient steps. We use the action repeat values from Table 2 for state-SAC as well. PI-SAC performs comparably to state-SAC on most tasks.

It is interesting to compare PI-SAC agents, which are trained from pixels, to SAC agents that have been trained from states. Generally, training from pixels is considered to be more challenging than training from states. However, we find that PI-SAC performs comparably to SAC from states on most tasks. On Cheetah Run and Finger Spin, PI-SAC significantly outperforms state-SAC, indicating that those tasks benefit strongly from representations that model what will happen next, rather than simply needing a precise description of the current state. In contrast, state-SAC has a noticeable sample efficiency advantage over PI-SAC on Ball In Cup Catch, indicating that a precise description of the current state that is stable throughout training is more important than learning to model what will happen next.

D Comparison of PI-SAC and Multiview Self-Prediction Agents

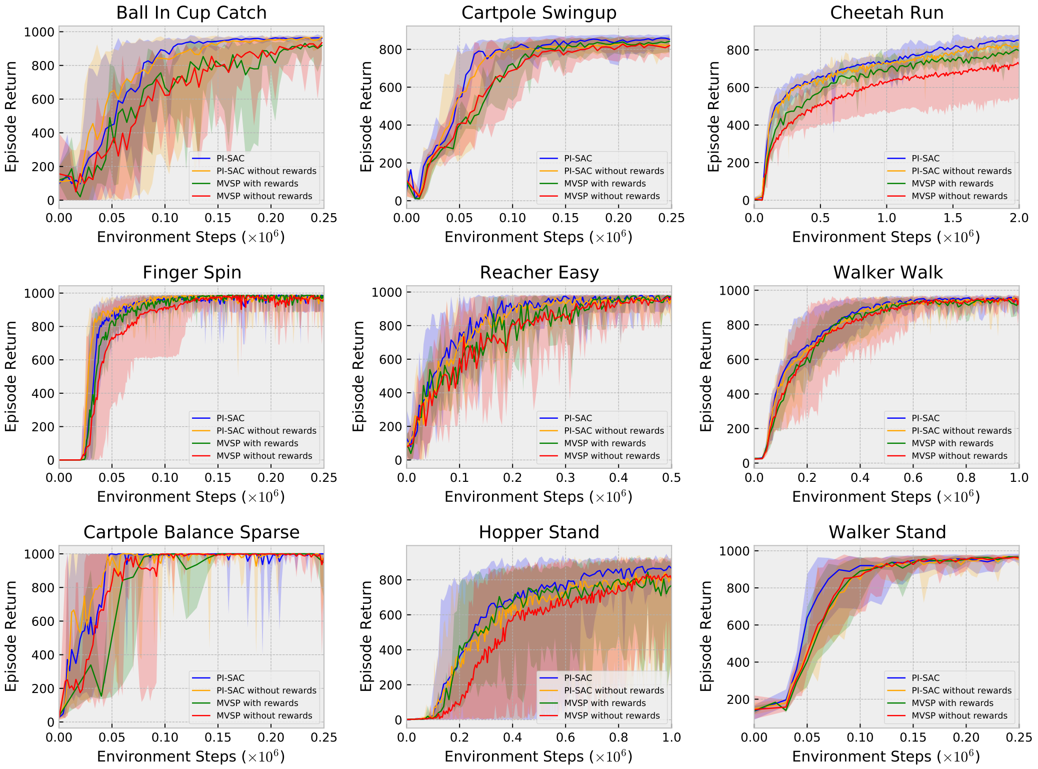

Figure 10: Learning the predictive information outperforms multiview self-prediction (MVSP), which is described in Section 4.2. We compare PI-SAC to using the MVSP auxiliary task on all 9 tasks. Specifically, we show PI-SAC and MVSP results with and without predicting future rewards in the auxiliary task. We use the default $\beta = 0.01$ for experiments in this figure.

In Figure 10 we present complete results of PI-SAC and Multiview Self-Prediction (MVSP) agents with and without predicting future rewards in the auxiliary task as an extension to Figure 6. For these experiments, we don't sweep β for the MVSP models, and instead use the default $\beta = 0.01$ for all models. In all cases, the PI-SAC models achieve equal or better performance. In general predicting future rewards leads to performance improvements.

E The Importance of Conditioning on Actions

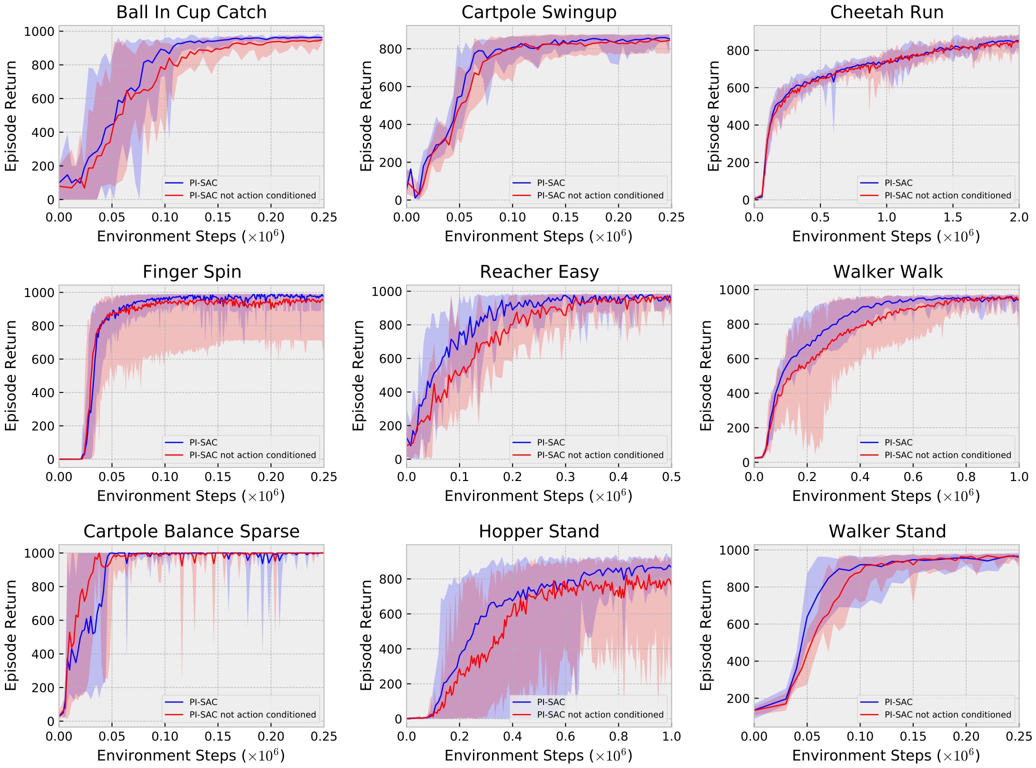

Figure 11: Standard PI-SAC that models environment dynamics $p(s'|s, a)$ outperforms the version that models $p(s'|s)$ without conditioning on actions on most tasks. We use the default $\beta = 0.01$ for experiments in this figure.

In Figure 11, we compare PI-SAC agents that predict the future conditioning on actions and those are not conditioning on actions to shed lights on the importance of modeling environment dynamics instead of temporal coherence. The former ones model environment dynamics $p(s'|s, a)$ which is independent of policy, whereas the latter ones instead capture temporal coherence of observations, i.e. the marginal $p(s'|s)$. Obviously, modeling $p(s'|s)$ has its limitation in off-policy learning as it fits to experience replay instead of the current policy [30]. As shown in Figure 11, conditioning on actions empirically leads to better sample efficiency and/or returns on most tasks, except for being slightly slower on Cartpole Balance Sparse. For the Cartpole Balance Sparse task, our conjecture is that, because the $p(s'|s)$ model is a simpler one without needing to consider actions and the Cartpole starts right at where the goal is in this task, it is still able to learn information as useful for achieving the goal as the $p(s'|s, a)$ model and even slightly faster.

F PI-SAC and SAC without Image Augmentation

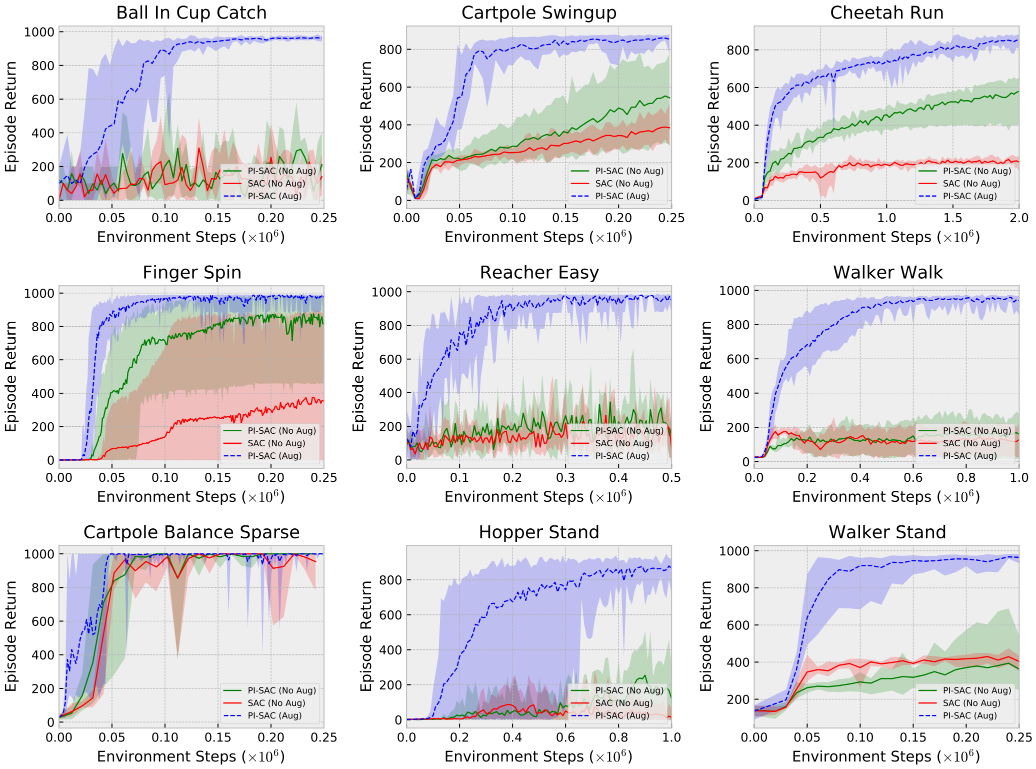

Figure 12: Comparison of PI-SAC and SAC both without image augmentation on all 9 tasks. We perform 5 runs for experiments without image augmentation. PI-SAC without image augmentation always matches or improves on the SAC baseline, but some tasks are only solved with the addition of image augmentation.

In Figure 12 we compare PI-SAC agents with SAC and PI-SAC agents trained without image augmentation on all nine tasks. This is the same setting as Figure 3. Learning the predictive information without image augmentation is sufficient to significantly improve SAC agents for some tasks, and is never detrimental compared to the SAC baseline. However, augmentation is essential to solving Reacher Easy, Walker Walk, and Hopper Stand. On all tasks, having both the predictive information and image augmentation performs the best.

G Comparison of Contrastive and Generative PI-SAC

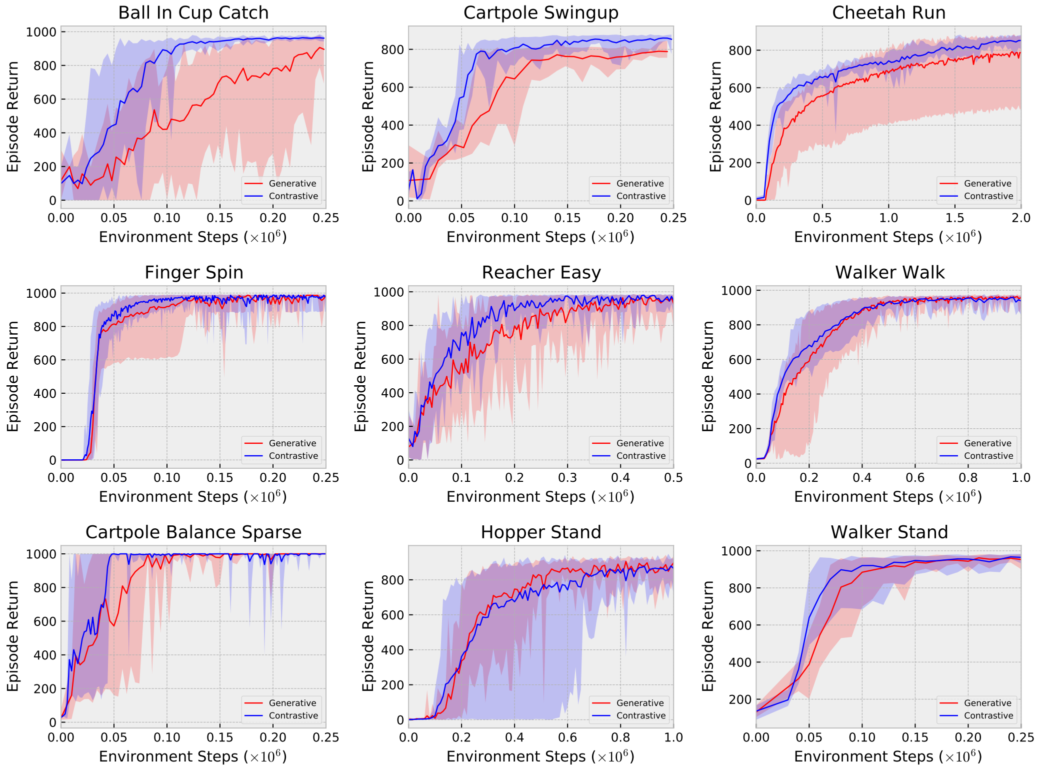

Figure 13: Comparison of the standard contrastive version and PI-SAC and the generative version of PI-SAC which directly predicts future observations and future rewards. The contrastive version shows better sample-efficiency and performance than the generative version every task except Hopper Stand, where the two approaches are essentially indistinguishable. The wall time per gradient step of the contrastive models is about 30% faster than the generative models, even with the small number of frames being predicted (3 future frames).

As shown in [7], under an assumption of a uniform distribution over the training examples, the contrastive CatGen formulation (eq. (5)) approximates the decoder distribution:

$$\frac{b(z|y)}{\sum_{k=1}^{K} b(z|y^k)} \approx p(y|z) \tag{11}$$

Instead of using CatGen in CEB, we can alternatively learn to predict y, the future observations and future rewards. This gives a generative variant of PI-SAC. To predict the future observations for the generative PI-SAC, we use a decoder network consisting of four transposed convolution layers with features of $(12, 64, 32, 3)$, kernel widths of $(3 \times 3, 3 \times 3, 11 \times 11, 3 \times 3)$, and strides of $(2, 2, 1, 2)$. To predict the future rewards, we use an MLP with 50-d first hidden layer and 25-d second hidden layer.

Figure 13 compares the standard contrastive version of PI-SAC to the generative variant. It shows that the contrastive version is generally more sample-efficient and gives better performance. Additionally, the generative version is slower to train in terms of wall time. These observations lead us to prefer the contrastive CatGen formulation for PI-SAC.

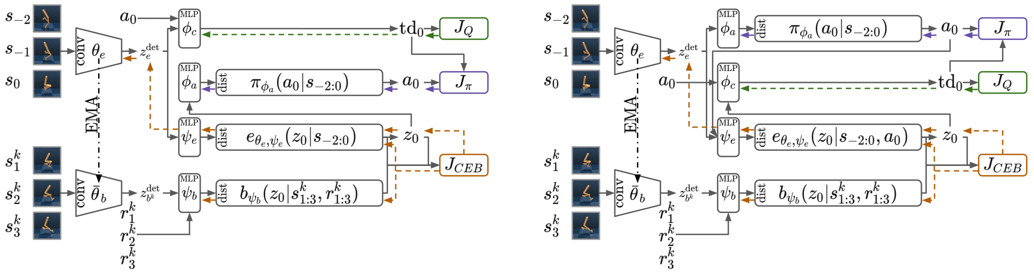

Figure 14: **Left:** Actor Representation PI-SAC system diagram. **Right:** Critic Representation PI-SAC system diagram.

H Representation PI-SAC

The CEB representation, Z, can be used directly by either the actor, the critic, or both. Figure 14 show the system diagrams for actor and critic Representation PI-SAC models. For models that use CEB representations for both actor and critic, it suffices to combine those systems so that there are two separate CEB objectives, but with both objectives updating the same convolutional encoder parameters (θ_e). In this setting, neither the actor nor the critic pass gradients back through the convolutional encoder parameters. This an important difference from PI-SAC, where both the CEB objective and the SAC critic objective contribute gradient information to the convolutional encoder parameters. As shown in Figure 14, the actor or the critic use the mean of the forward encoder's Z distribution (z_e^{det} in the figures). It is also possible to train using samples from the distribution (z_0 in the figures). Empirically, we found that the results were qualitatively the same, but often with higher variance in evaluation performance early in training, so we only present results using the mean of the representation here.

Figure 15 shows results comparing Representation PI-SAC models to Dreamer on the same tasks from the paper. In these experiments, no hyperparameters are changed between the different tasks. All Representation PI-SAC models in Figure 15 are trained using the hyperparameters in Table 2, action repeat of 2, 1000 initial collection steps, and 0 initial CEB steps. This makes the results directly comparable to Dreamer, which also uses action repeat of 2 for all tasks and has no task-specific hyperparameters.

Using a CEB representation for the actor, the critic, or both still gives much better sample efficiency than Dreamer at 6 of the 9 tasks, but the lack of the full set of future actions as input to the forward encoder in these models appears to make it more difficult for the agents to solve locomotion tasks like Cheetah Run and Walker Walk. We also tested a critic Representation PI-SAC model variant that allowed critic gradients to flow to the convolutional encoder parameters (as in PI-SAC), and found that doing so improved performance on Cartpole Swingup and Reacher Easy, but substantially worse on Ball In Cup Catch, Cartpole Balance Sparse, and Cheetah Run, and comparable on the remaining four tasks. This indicates that the difficulty on the locomotion tasks we see here is not due to the lack of critic gradients.

On most tasks, we found that actor Representation PI-SAC had the strongest performance and best stability of the three Representation PI-SAC variants. In particular, its stability on Cartpole Balance Sparse was remarkable: from 80,000 environment steps through 480,000 environment steps, all five agents got perfect scores of 1,000 on all 10 evaluation trials that occurred every 10,000 environment steps, for a total of 2,000 perfect evaluations in a row. In contrast, the other two variants (and Dreamer) had substantial deviations from perfect scores throughout training. More exploration is necessary to understand why actor Representation PI-SAC has this stability advantage.

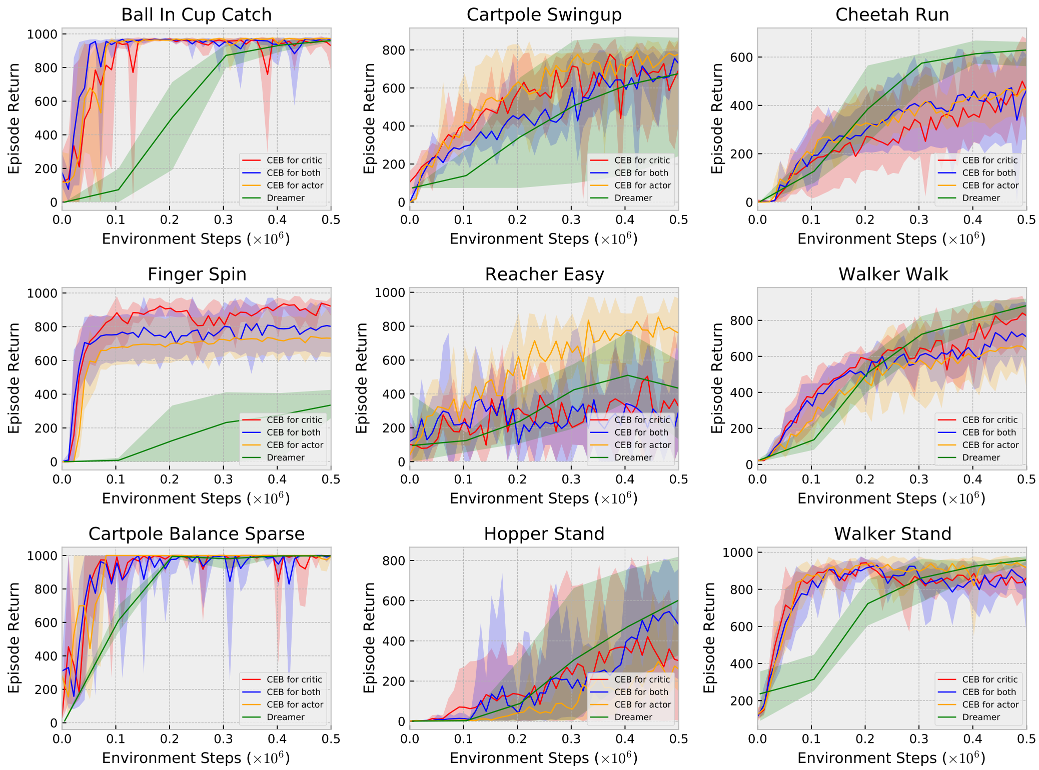

Figure 15: Representation PI-SAC models with action repeat of 2 and no task-dependent hyperparameter changes, making the experiments directly comparable to the Dreamer results. All curves are 5 runs.

I Discussion of Compression

As discussed in Section 4.3, compression is known to improve generalization [36, 3, 8]. For example, in [36], the authors show that every additional bit in a representation Z requires four times as much training data to achieve the same generalization, which may explain PI-SAC sample efficiency gains:

$$|I(Y;Z) - I(\hat{Y};Z)| \leq \mathcal{O}\left(\frac{|Y|2^{I(\hat{X};Z)}}{\sqrt{N}}\right) \tag{12}$$

Here, $\hat{X}$ and $\hat{Y}$ are the training observations and targets, Z is the learned representation, N is the number of training examples, and X and Y are the observations and targets in the full distribution the training examples are sampled from. Since we are trying to learn Z that maximizes $I(Y;Z)$ while only observing $\hat{X}$ and $\hat{Y}$, we want $I(\hat{Y};Z)$ to be as close to $I(Y;Z)$ as possible. This bound makes it clear that we can make the two close by either increasing N or decreasing $I(\hat{X};Z)$.

Of course, one way to make that bound tight is to make Z independent of $\hat{X}$ and $\hat{Y}$. In that case, all of $I(\hat{X};Z)$, $I(\hat{Y};Z)$, and $I(Y;Z)$ converge to 0. To avoid this, we would like to learn the *Minimum Necessary Information* (MNI), introduced in [7]. The MNI is defined as the equality:

$$I(X;Y) = I(X;Z) = I(Y;Z) \tag{13}$$

Intuitively, this says that we are trying to find a representation Z that captures exactly the information that is shared between X and Y. Learning such a Z corresponds to finding the maximum of $I(Y;Z)$ while also having $I(X;Z|Y) = 0$.[4] This is basis of the CEB objective function which we apply here to the problem of bounding the Predictive Information, as described in Section 2.

[4] And also having $I(Y;Z|X) = 0$, but that can be enforced by having the Markov chain $Z \leftarrow X \rightarrow Y$, which means that Z is a stochastic function of X only.