CURL: Contrastive Unsupervised Representations for Reinforcement Learning

Aravind Srinivas[*1] **Michael Laskin**[*1] **Pieter Abbeel**[1]

Abstract

We present CURL: **C**ontrastive **U**nsupervised Representations for **R**einforcement **L**earning. CURL extracts high-level features from raw pixels using contrastive learning and performs off-policy control on top of the extracted features. CURL outperforms prior pixel-based methods, both model-based and model-free, on complex tasks in the DeepMind Control Suite and Atari Games showing 1.9x and 1.2x performance gains at the 100K environment and interaction steps benchmarks respectively. On the DeepMind Control Suite, CURL is the first image-based algorithm to nearly match the sample-efficiency of methods that use state-based features. Our code is open-sourced and available at https://www.github.com/MishaLaskin/curl.

1. Introduction

Developing agents that can perform complex control tasks from high dimensional observations such as pixels has been possible by combining the expressive power of deep neural networks with the long-term credit assignment power of reinforcement learning algorithms. Notable successes include learning to play a diverse set of video games from raw pixels (Mnih et al., 2015), continuous control tasks such as controlling a simulated car from a dashboard camera (Lillicrap et al., 2015) and subsequent algorithmic developments and applications to agents that successfully navigate mazes and solve complex tasks from first-person camera observations (Jaderberg et al., 2016; Espeholt et al., 2018; Jaderberg et al., 2019); and robots that successfully grasp objects in the real world (Kalashnikov et al., 2018).

However, it has been empirically observed that reinforcement learning from high dimensional observations such as raw pixels is sample-inefficient (Lake et al., 2017; Kaiser et al., 2019). Moreover, it is widely accepted that learning policies from physical state based features is significantly more sample-efficient than learning from pixels (Tassa et al., 2018). In principle, if the state information is present in the pixel data, then we should be able to learn representations that extract the relevant state information. For this reason, it may be possible to learn from pixels as fast as from state given the right representation.

From a practical standpoint, although high rendering speeds in simulated environments enable RL agents to solve complex tasks within reasonable wall clock time, learning in the real world means that agents are bound to work within the limitations of physics. Kalashnikov et al. (2018) needed a farm of robotic arms that collected large scale robot in-

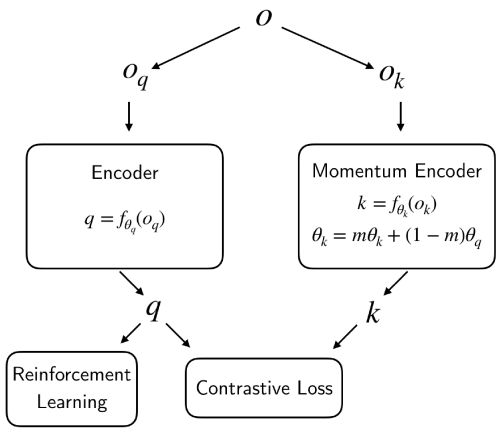

Figure 1. **C**ontrastive **U**nsupervised Representations for **R**einforcement **L**earning (CURL) combines instance contrastive learning and reinforcement learning. CURL trains a visual representation encoder by ensuring that the embeddings of data-augmented versions o_q and o_k of observation o match using a contrastive loss. The *query* observations o_q are treated as the anchor while the *key* observations o_k contain the positive and negatives, all constructed from the minibatch sampled for the RL update. The keys are encoded with a momentum averaged version of the query encoder. The RL policy and (or) value function are built on top of the query encoder which is jointly trained with the contrastive and reinforcement learning objectives. CURL is a generic framework that can be plugged into any RL algorithm that relies on learning representations from high dimensional images.

[*]Equal contribution [1]University of California, Berkeley, BAIR. Correspondence to: Aravind Srinivas, Michael Laskin <aravind_srinivas, mlaskin@berkeley.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

teraction data over several months to develop their robot grasp value functions and policies. The data-efficiency of the whole pipeline thus has significant room for improvement. Similarly, in simulated worlds which are limited by rendering speeds in the absence of GPU accelerators, data efficiency is extremely crucial to have a fast experimental turnover and iteration. Therefore, improving the sample efficiency of reinforcement learning (RL) methods that operate from high dimensional observations is of paramount importance to RL research both in simulation and the real world and allows for faster progress towards the broader goal of developing intelligent autonomous agents.

A number of approaches have been proposed in the literature to address the sample inefficiency of deep RL algorithms. Broadly, they can be classified into two streams of research, though not mutually exclusive: (i) Auxiliary tasks on the agent's sensory observations; (ii) World models that predict the future. While the former class of methods use auxiliary self-supervision tasks to accelerate the learning progress of model-free RL methods (Jaderberg et al., 2016; Mirowski et al., 2016), the latter class of methods build explicit predictive models of the world and use those models to plan through or collect fictitious rollouts for model-free methods to learn from (Sutton, 1990; Ha & Schmidhuber, 2018; Kaiser et al., 2019; Schrittwieser et al., 2019).

Our work falls into the first class of models, which use auxiliary tasks to improve sample efficiency. Our hypothesis is simple: *If an agent learns a useful semantic representation from high dimensional observations, control algorithms built on top of those representations should be significantly more data-efficient.* Self-supervised representation learning has seen dramatic progress in the last couple of years with huge advances in masked language modeling (Devlin et al., 2018) and contrastive learning (Hénaff et al., 2019; He et al., 2019a; Chen et al., 2020) for language and vision respectively. The representations uncovered by these objectives improve the performance of any supervised learning system especially in scenarios where the amount of labeled data available for the downstream task is really low.

We take inspiration from the contrastive pre-training successes in computer vision. However, there are a couple of key differences: (i) There is no giant unlabeled dataset of millions of images available beforehand - the dataset is collected online from the agent's interactions and changes dynamically with the agent's experience; (ii) The agent has to perform unsupervised and reinforcement learning simultaneously as opposed to fine-tuning a pre-trained network for a specific downstream task. These two differences introduce a different challenge: How can we use contrastive learning for improving agents that can learn to control effectively and efficiently from online interactions?

To address this challenge, we propose CURL - **C**ontrastive **U**unsupervised Representations for **R**einforcement **L**earning. CURL uses a form of contrastive learning that maximizes agreement between augmented versions of the same observation, where each observation is a stack of temporally sequential frames. We show that CURL significantly improves sample-efficiency over prior pixel-based methods by performing contrastive learning simultaneously with an off-policy RL algorithm. CURL coupled with the Soft-Actor-Critic (SAC) (Haarnoja et al., 2018) results in **1.9x** median higher performance over Dreamer, a prior state-of-the-art algorithm on DMControl environments, benchmarked at **100k** *environment steps* and *matches the performance of state-based SAC* on the majority of 16 environments tested, a **first** for pixel-based methods. In the Atari setting benchmarked at 100k *interaction steps*, we show that CURL coupled with a data-efficient version of Rainbow DQN (van Hasselt et al., 2019) results in **1.2x** median higher performance over prior methods such as SimPLe (Kaiser et al., 2019), improving upon Efficient Rainbow (van Hasselt et al., 2019) on *19 out of 26* Atari games, *surpassing human efficiency* on two games.

While contrastive learning in aid of model-free RL has been studied in the past by van den Oord et al. (2018) using Contrastive Predictive Coding (CPC), the results were mixed with marginal gains in a few DMLab (Espeholt et al., 2018) environments. CURL is the first model to show substantial data-efficiency gains from using a contrastive self-supervised learning objective for model-free RL agents across a multitude of pixel based continuous and discrete control tasks in DMControl and Atari.

We prioritize designing a simple and easily reproducible pipeline. While the promise of auxiliary tasks and learning world models for RL agents has been demonstrated in prior work, there's an added layer of complexity when introducing components like modeling the future in a latent space (van den Oord et al., 2018; Ha & Schmidhuber, 2018). CURL is designed to add minimal overhead in terms of architecture and model learning. The contrastive learning objective in CURL operates with the same latent space and architecture typically used for model-free RL and seamlessly integrates with the training pipeline without the need to introduce multiple additional hyperparameters.

Our paper makes the following **key contributions**: We present CURL, a simple framework that integrates contrastive learning with model-free RL with minimal changes to the architecture and training pipeline. Using 16 complex control tasks from the DeepMind control (DMControl) suite and 26 Atari games, we empirically show that contrastive learning combined with model-free RL outperforms the prior state-of-the-art by 1.9x on DMControl and 1.2x on Atari compared across leading prior pixel-based methods. CURL is also the first algorithm *across both model-based*

and model-free methods that operates purely from pixels, and nearly matches the performance and sample-efficiency of a SAC algorithm trained from the state based features on the DMControl suite. Finally, our design is simple and does not require any custom architectural choices or hyperparameters which is crucial for reproducible end-to-end training. Through these strong empirical results, we demonstrate that a contrastive objective is the preferred self-supervised auxiliary task for achieving sample-efficiency compared to reconstruction based methods, and enables *model-free methods to outperform state-of-the-art model-based methods in terms of data-efficiency*.

2. Related Work

Self-Supervised Learning: Self-Supervised Learning is aimed at learning rich representations of high dimensional unlabeled data to be useful for a wide variety of tasks. The fields of natural language processing and computer vision have seen dramatic advances in self-supervised methods such as BERT (Devlin et al., 2018), CPC, MoCo, SimCLR (Hénaff et al., 2019; He et al., 2019a; Chen et al., 2020).

Contrastive Learning: Contrastive Learning is a framework to learn representations that obey similarity constraints in a dataset typically organized by similar and dissimilar pairs. This is often best understood as performing a dictionary lookup task wherein the positive and negatives represent a set of keys with respect to a query (or an anchor). A simple instantiation of contrastive learning is Instance Discrimination (Wu et al., 2018) wherein a query and key are positive pairs if they are data-augmentations of the same instance (example, image) and negative otherwise. A key challenge in contrastive learning is the choice of negatives which can decide the quality of the underlying representations learned. The loss functions used to contrast could be among several choices such as InfoNCE (van den Oord et al., 2018), Triplet (Wang & Gupta, 2015), Siamese (Chopra et al., 2005) and so forth.

Self-Supervised Learning for RL: Auxiliary tasks such as predicting the future conditioned on the past observation(s) and action(s) (Jaderberg et al., 2016; Shelhamer et al., 2016; van den Oord et al., 2018; Schmidhuber, 1990) are a few representative examples of using auxiliary tasks to improve the sample-efficiency of model-free RL algorithms. The future prediction is either done in a pixel space (Jaderberg et al., 2016) or latent space (van den Oord et al., 2018). The sample-efficiency gains from reconstruction-based auxiliary losses have been benchmarked in Jaderberg et al. (2016); Higgins et al. (2017); Yarats et al. (2019). Contrastive learning has been used to extract reward signals in the latent space (Sermanet et al., 2018; Dwibedi et al., 2018; Warde-Farley et al., 2018); and study representation learning on Atari games by Anand et al. (2019).

World Models for sample-efficiency: While joint learning of an auxiliary unsupervised task with model-free RL is one way to improve the sample-efficiency of agents, there has also been another line of research that has tried to learn world models of the environment and use them to sample rollouts and plan. An early instantiation of the generic principle was put forth by Sutton (1990) in Dyna where fictitious samples rolled out from a learned world model are used in addition to the agent's experience for sample-efficient learning. Planning through a learned world model (Srinivas et al., 2018) is another way to improve sample-efficiency. While Jaderberg et al. (2016); van den Oord et al. (2018); Lee et al. (2019) also learn pixel and latent space forward models, the models are learned to shape the latent representations, and there is no explicit Dyna or planning. Planning through learned world models has been successfully demonstrated in Ha & Schmidhuber (2018); Hafner et al. (2018; 2019). Kaiser et al. (2019) introduce SimPLe which implements Dyna with expressive deep neural networks for the world model for sample-efficiency on Atari games.

Sample-efficient RL for image-based control: CURL encompasses the areas of self-supervision, contrastive learning and using auxiliary tasks for sample-efficient RL. We benchmark for sample-efficiency on the DMControl suite (Tassa et al., 2018) and Atari Games benchmarks (Bellemare et al., 2013). The DMControl suite has been used widely by Yarats et al. (2019), Hafner et al. (2018), Hafner et al. (2019) and Lee et al. (2019) for benchmarking sample-efficiency for image based continuous control methods. As for Atari, Kaiser et al. (2019) propose to use the 100k interaction steps benchmark for sample-efficiency which has been adopted in Kielak (2020); van Hasselt et al. (2019). The Rainbow DQN (Hessel et al., 2017) was originally proposed for maximum sample-efficiency on the Atari benchmark and in recent times has been adapted to a version known as Data-Efficient Rainbow (van Hasselt et al., 2019) with competitive performance to SimPLe without learning world models. We benchmark extensively against both model-based and model-free algorithms in our experiments. For the DMControl experiments, we compare our method to Dreamer, PlaNet, SLAC, SAC+AE whereas for Atari experiments we compare to SimPLe, Rainbow, and OverTrained Rainbow (OTRainbow) and Efficient Rainbow (Eff. Rainbow).

3. Background

CURL is a general framework for combining contrastive learning with RL. In principle, one could use any RL algorithm in the CURL pipeline, be it on-policy or off-policy. We use the widely adopted Soft Actor Critic (SAC) (Haarnoja et al., 2018) for continuous control benchmarks (DM Control) and Rainbow DQN (Hessel et al., 2017; van Hasselt et al., 2019) for discrete control benchmarks (Atari). Below, we review SAC, Rainbow DQN and Contrastive Learning.

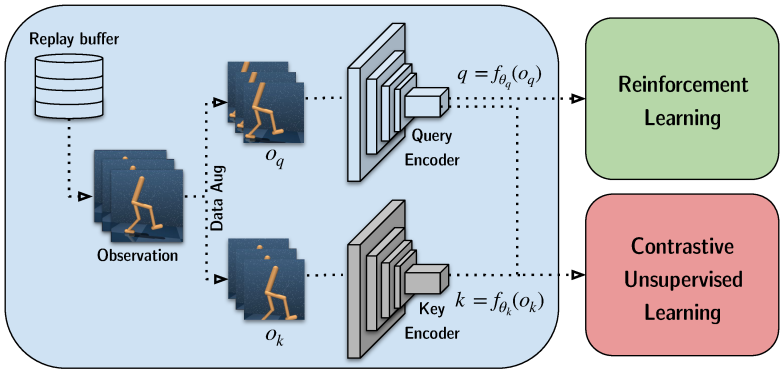

Figure 2. CURL Architecture: A batch of transitions is sampled from the replay buffer. Observations are then data-augmented twice to form *query* and *key* observations, which are then encoded with the query encoder and key encoders, respectively. The *queries* are passed to the RL algorithm while *query-key* pairs are passed to the contrastive learning objective. During the gradient update step, only the *query* encoder is updated. The *key* encoder weights are the moving average (EMA) of the query weights similar to MoCo (He et al., 2019a).

3.1. Soft Actor Critic

SAC is an off-policy RL algorithm that optimizes a stochastic policy for maximizing the expected trajectory returns. Like other state-of-the-art end-to-end RL algorithms, SAC is effective when solving tasks from state observations but fails to learn efficient policies from pixels. SAC is an actor-critic method that learns a policy π_ψ and critics Q_{ϕ_1} and Q_{ϕ_2}. The parameters ϕ_i are learned by minimizing the Bellman error:

$$\mathcal{L}(\phi_i, \mathcal{B}) = \mathbb{E}_{t\sim\mathcal{B}}\left[\left(Q_{\phi_i}(o,a) - (r + \gamma(1-d)\mathcal{T})\right)^2\right] \tag{1}$$

where $t = (o, a, o', r, d)$ is a tuple with observation o, action a, reward r and done signal d, $\mathcal{B}$ is the replay buffer, and $\mathcal{T}$ is the target, defined as:

$$\mathcal{T} = \left(\min_{i=1,2} Q^*_{\phi_i}(o', a') - \alpha \log \pi_\psi(a'|o')\right) \tag{2}$$

In the target equation (2), $Q^*_{\phi_i}$ denotes the exponential moving average (EMA) of the parameters of Q_{ϕ_i}. Using the EMA has empirically shown to improve training stability in off-policy RL algorithms. The parameter α is a positive entropy coefficient that determines the priority of the entropy maximization over value function optimization.

While the critic is given by Q_{ϕ_i}, the actor samples actions from policy π_ψ and is trained by maximizing the expected return of its actions as in:

$$\mathcal{L}(\psi) = \mathbb{E}_{a\sim\pi}\left[Q^\pi(o,a) - \alpha \log \pi_\psi(a|o)\right] \tag{3}$$

where actions are sampled stochastically from the policy $a_\psi(o, \xi) \sim \tanh\left(\mu_\psi(o) + \sigma_\psi(o) \odot \xi\right)$ and $\xi \sim \mathcal{N}(0, I)$ is a standard normalized noise vector.

3.2. Rainbow

Rainbow DQN (Hessel et al., 2017) is best summarized as multiple improvements on top of the original Nature DQN (Mnih et al., 2015) applied together. Specifically, Deep Q Network (DQN) (Mnih et al., 2015) combines the off-policy algorithm Q-Learning with a convolutional neural network as the function approximator to map raw pixels to action value functions. Since then, multiple improvements have been proposed such as Double Q Learning (Van Hasselt et al., 2016), Dueling Network Architectures (Wang et al., 2015), Prioritized Experience Replay (Schaul et al., 2015), and Noisy Networks (Fortunato et al., 2017). Additionally, distributional reinforcement learning (Bellemare et al., 2017) proposed the technique of predicting a distribution over possible value function bins through the C51 Algorithm. Rainbow DQN combines all of the above techniques into a single off-policy algorithm for state-of-the-art sample efficiency on Atari benchmarks. Additionally, Rainbow also makes use of multi-step returns (Sutton et al., 1998). van Hasselt et al. (2019) propose a data-efficient version of the Rainbow which can be summarized as an improved configuration of hyperparameters that is optimized for performance benchmarked at 100K interaction steps.

3.3. Contrastive Learning

A key component of CURL is the ability to learn rich representations of high dimensional data using contrastive unsupervised learning. Contrastive learning (Hadsell et al., 2006; LeCun et al., 2006; van den Oord et al., 2018; Wu et al., 2018; He et al., 2019a) can be understood as learning a differentiable dictionary look-up task. Given a query q and keys $\mathbb{K} = \{k_0, k_1, \dots\}$ and an explicitly known partition of $\mathbb{K}$ (with respect to q) $P(\mathbb{K}) = (\{k_+\}, \mathbb{K} \setminus \{k_+\})$, the goal of contrastive learning is to ensure that q matches with k_+ relatively more than any of the keys in $\mathbb{K} \setminus \{k_+\}$. $q, \mathbb{K}, k_+,$

and $\mathbb{K} \setminus \{k_+\}$ are also referred to as *anchor, targets, positive, negatives* respectively in the parlance of contrastive learning (van den Oord et al., 2018; He et al., 2019a). Similarities between the anchor and targets are best modeled with dot products ($q^T k$) (Wu et al., 2018; He et al., 2019a) or bilinear products ($q^T W k$) (van den Oord et al., 2018; Hénaff et al., 2019) though other forms like euclidean distances are also common (Schroff et al., 2015; Wang & Gupta, 2015). To learn embeddings that respect these similarity relations, van den Oord et al. (2018) propose the InfoNCE loss:

$$\mathcal{L}_q = \log \frac{\exp(q^T W k_+)}{\exp(q^T W k_+) + \sum_{i=0}^{K-1} \exp(q^T W k_i)} \quad (4)$$

The loss 4 can be interpreted as the log-loss of a K-way softmax classifier whose label is k_+.

4. CURL Implementation

CURL minimally modifies a base RL algorithm by training the contrastive objective as an auxiliary loss during the batch update. In our experiments, we train CURL alongside two model-free RL algorithms — SAC for DMControl experiments and Rainbow DQN (data-efficient version) for Atari experiments. To specify a contrastive learning objective, we need to define (i) the discrimination objective (ii) the transformation for generating query-key observations (iii) the embedding procedure for transforming observations into queries and keys and (iv) the inner product used as a similarity measure between the query-key pairs in the contrastive loss. The exact specification these aspects largely determine the quality of the learned representations.

We first summarize the CURL architecture, and then cover each architectural choice in detail.

4.1. Architectural Overview

CURL uses instance discrimination with similarities to SimCLR (Chen et al., 2020), MoCo (He et al., 2019a) and CPC (Hénaff et al., 2019). Most Deep RL architectures operate with a stack of temporally consecutive frames as input (Hessel et al., 2017). Therefore, instance discrimination is performed across the frame stacks as opposed to single image instances. We use a momentum encoding procedure for targets similar to MoCo (He et al., 2019b) which we found to be better performing for RL. Finally, for the InfoNCE score function, we use a bi-linear inner product similar to CPC (van den Oord et al., 2018) which we found to work better than unit norm vector products used in MoCo and SimCLR. Ablations for both the encoder and the similarity measure choices are shown in Figure 5. The contrastive representation is trained jointly with the RL algorithm, and the latent code receives gradients from both the contrastive objective and the Q-function. An overview of the architecture is shown in in Figure 2.

4.2. Discrimination Objective

A key component of contrastive representation learning is the choice of positives and negative samples relative to an anchor (Bachman et al., 2019; Tian et al., 2019; Hénaff et al., 2019; He et al., 2019a; Chen et al., 2020). Contrastive Predictive Coding (CPC) based pipelines (Hénaff et al., 2019; van den Oord et al., 2018) use groups of image patches separated by a carefully chosen spatial offset for anchors and positives while the negatives come from other patches within the image and from other images.

While patches are a powerful way to incorporate spatial and instance discrimination together, they introduce extra hyperparameters and architectural design choices which may be hard to adapt for a new problem. SimCLR (Chen et al., 2020) and MoCo (He et al., 2019a) opt for a simpler design where there is no patch extraction.

Discriminating transformed image instances as opposed to image-patches within the same image optimizes a simpler instance discrimination objective (Wu et al., 2018) with the InfoNCE loss and requires minimal architectural adjustments (He et al., 2019b; Chen et al., 2020). It is preferable to pick a simpler discrimination objective in the RL setting for two reasons. First, considering the brittleness of reinforcement learning algorithms (Henderson et al., 2018), complex discrimination may destabilize the RL objective. Second, since RL algorithms are trained on dynamically generated datasets, a complex discrimination objective may significantly increase the wall-clock training time. CURL therefore uses instance discrimination rather than patch discrimination. One could view contrastive instance discrimination setups like SimCLR and MoCo as maximizing mutual information between an image and its augmented version. The reader is encouraged to refer to van den Oord et al. (2018); Hjelm et al. (2018); Tschannen et al. (2019) for connections between contrastive learning and mutual information.

4.3. Query-Key Pair Generation

Similar to instance discrimination in the image setting (He et al., 2019b; Chen et al., 2020), the anchor and positive observations are two different augmentations of the same image while negatives come from other images. CURL primarily relies on the random crop data augmentation, where a random square patch is cropped from the original rendering.

A significant difference between RL and computer vision settings is that an instance ingested by a model-free RL algorithm that operates from pixels is not just a single image but a stack of frames (Mnih et al., 2015). For example, one typically feeds in a stack of 4 frames in Atari experiments

and a stack of 3 frames in DMControl. This way, performing instance discrimination on frame stacks allows CURL to learn both spatial and temporal discriminative features. For details regarding the extent to which CURL captures temporal features, see Appendix E.

We apply the random augmentations across the batch but consistently across each stack of frames to retain information about the temporal structure of the observation. The augmentation procedure is shown in Figure 3. For more details, refer to Appendix A.

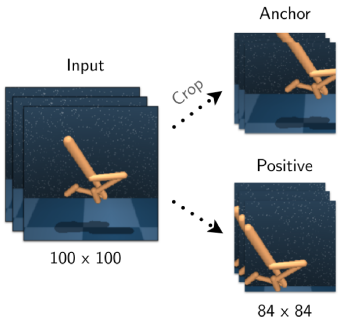

Figure 3. Visually illustrating the process of generating an anchor and its positive using stochastic random crops. Our aspect ratio for cropping is 0.84, i.e, we crop a 84×84 image from a 100×100 simulation-rendered image. Applying the same random crop coordinates across all frames in the stack ensures time-consistent spatial jittering.

4.4. Similarity Measure

Another determining factor in the discrimination objective is the inner product used to measure agreement between query-key pairs. CURL employs the bi-linear inner-product $\text{sim}(q, k) = q^T W k$, where W is a learned parameter matrix. We found this similarity measure to outperform the normalized dot-product (see Figure 5 in Appendix A) used in recent state-of-the-art contrastive learning methods in computer vision like MoCo and SimCLR.

4.5. Target Encoding with Momentum

The motivation for using contrastive learning in CURL is to train encoders that map from high dimensional pixels to more semantic latents. InfoNCE is an unsupervised loss that learns encoders f_q and f_k mapping the raw anchors (query) x_q and targets (keys) x_k into latents $q = f_q(x_q)$ and $k = f_k(x_k)$, on which we apply the similarity dot products. It is common to share the same encoder between the anchor and target mappings, that is, to have $f_q = f_k$ (van den Oord et al., 2018; Hénaff et al., 2019).

From the perspective of viewing contrastive learning as building differentiable dictionary lookups over high dimensional entities, increasing the size of the dictionary and enriching the set of negatives is helpful in learning rich representations. He et al. (2019a) propose momentum contrast (MoCo), which uses the exponentially moving average (momentum averaged) version of the query encoder f_q for encoding the keys in $\mathbb{K}$. Given f_q parametrized by θ_q and f_k parametrized by θ_k, MoCo performs the update $\theta_k = m\theta_k + (1-m)\theta_q$ and encodes any target x_k using $\text{SG}(f_k(x_k))$ [SG : Stop Gradient].

CURL couples frame-stack instance discrimination with momentum encoding for the targets during contrastive learning, and RL is performed on top of the encoder features.

4.6. Differences Between CURL and Prior Contrastive Methods in RL

van den Oord et al. (2018) use Contastive Predictive Coding (CPC) as an auxiliary task wherein an LSTM operates on a latent space of a convolutional encoder; and both the CPC and A2C (Mnih et al., 2015) objectives are jointly optimized. CURL avoids using pipelines that *predict the future* in a latent space such as van den Oord et al. (2018); Hafner et al. (2019). In CURL, we opt for a simple instance discrimination style contrastive auxiliary task.

4.7. CURL Contrastive Learning Pseudocode (PyTorch-like)

```
# f_q, f_k: encoder networks for anchor
# (query) and target (keys) respectively.
# loader: minibatch sampler from ReplayBuffer
# B-batch_size, C-channels, H,W-spatial_dims
# x : shape : [B, C, H, W]
# C = c * num_frames; c=3 (R/G/B) or 1 (gray)
# m: momentum, e.g. 0.95
# z_dim: latent dimension
f_k.params = f_q.params
W = rand(z_dim, z_dim) # bilinear product.
for x in loader: # load minibatch from buffer
 x_q = aug(x) # random augmentation
 x_k = aug(x) # different random augmentation
 z_q = f_q.forward(x_q)
 z_k = f_k.forward(x_k)
 z_k = z_k.detach() # stop gradient
 proj_k = matmul(W, z_k.T) # bilinear product
 logits = matmul(z_q, proj_k) # B x B
  # subtract max from logits for stability
 logits = logits - max(logits, axis=1)
 labels = arange(logits.shape[0])
 loss = CrossEntropyLoss(logits, labels)
 loss.backward()
 update(f_q.params) # Adam
 update(W) # Adam
 f_k.params = m*f_k.params+(1-m)*f_q.params
```

5. Experiments

5.1. Evaluation

We measure the data-efficiency and performance of our method and baselines at 100k and 500k *environment steps* on DMControl and 100k *interaction steps* (400k environment steps with action repeat of 4) on Atari, which we will henceforth refer to as **DMControl100k**, **DMControl500k** and **Atari100k** for clarity. While Atari100k benchmark has

been common practice when investigating data-efficiency on Atari (Kaiser et al., 2019; van Hasselt et al., 2019; Kielak, 2020), the DMControl benchmark was set at 500k environment steps because state-based RL approaches asymptotic performance on many environments at this point, and 100k steps to measure the speed of initial learning. A broader motivation is that while RL algorithms can achieve superhuman performance on Atari games, they are still far less efficient than a human learner. Training for 100-500k environment steps corresponds to a few hours of human time.

We evaluate (i) *sample-efficiency* by measuring how many steps it takes the best performing baselines to match CURL performance at a fixed T (100k or 500k) steps and (ii) *performance* by measuring the ratio of the episode returns achieved by CURL versus the best performing baseline at T steps. To be explicit, when we say data or sample-efficiency we're referring to (i) and when we say performance we're referring to (ii).

5.2. Environments

Our primary goal for CURL is sample-efficient control from pixels that is broadly applicable across a range of environments. We benchmark the performance of CURL for both discrete and continuous control environments. Specifically, we focus on DMControl suite for continuous control tasks and the Atari Games benchmark for discrete control tasks with inputs being raw pixels rendered by the environments.

DeepMind Control: Recently, there have been a number of papers that have benchmarked for sample efficiency on challenging visual continuous control tasks belonging to the DMControl suite (Tassa et al., 2018) where the agent operates purely from pixels. The reason for operating in these environments is multi fold: (i) they present a reasonably challenging and diverse set of tasks; (ii) sample-efficiency of pure model-free RL algorithms operating from pixels on these benchmarks is poor; (iii) multiple recent efforts to improve the sample efficiency of both model-free and model-based methods on these benchmarks thereby giving us sufficient baselines to compare against; (iv) performance on the DM control suite is relevant to robot learning in real world benchmarks.

We run experiments on sixteen environments from DMControl to examine the performance of CURL on pixels relative to SAC with access to the ground truth state, shown in Figure 7. For more extensive benchmarking, we compare CURL to five leading pixel-based methods across the the six environments presented in Yarats et al. (2019): ball-in-cup, finger-spin, reacher-easy, cheetah-run, walker-walk, cartpole-swingup for benchmarking.

Atari: Similar to DMControl sample-efficiency benchmarks, there have been a number of recent papers that have benchmarked for sample-efficiency on the Atari 2600 Games. Kaiser et al. (2019) proposed comparing various algorithms in terms of performance achieved within 100K timesteps (400K frames, frame skip of 4) of interaction with the environments (games). The method proposed by Kaiser et al. (2019) called SimPLe is a model-based RL algorithm. SimPLe is compared to a random agent, model-free Rainbow DQN (Hessel et al., 2017) and human performance for the same amount of interaction time. Recently, van Hasselt et al. (2019) and Kielak (2020) proposed data-efficient versions of Rainbow DQN which are competitive with SimPLe on the same benchmark. Given that the same benchmark has been established in multiple recent papers and that there is a human baseline to compare to, we benchmark CURL on all the 26 Atari Games (Table 2).

5.3. Baselines for benchmarking sample efficiency

DMControl baselines: We present a number of baselines for continuous control within the DMControl suite: (i) SAC-AE (Yarats et al., 2019) where the authors attempt to use a β-VAE (Higgins et al., 2017), VAE (Kingma & Welling, 2013) and a regualrized autoencoder Vincent et al. (2008); Ghosh et al. (2019) jointly with SAC; (ii) SLAC (Lee et al., 2019) which learns a latent space world model on top of VAE features Ha & Schmidhuber (2018) and builds value functions on top; (iii) PlaNet and (iv) Dreamer (Hafner et al., 2018; 2019) both of which learn a latent space world model and explicitly plan through it; (v) Pixel SAC: Vanilla SAC operating purely from pixels (Haarnoja et al., 2018). These baselines are competitive methods for benchmarking control from pixels. In addition to these, we also present the baseline State-SAC where the assumption is that the agent has access to low level state based features and does not operate from pixels. This baseline acts as an *oracle* in that it approximates the upper bound of how sample-efficient a pixel-based agent can get in these environments.

Atari baselines: For benchmarking performance on Atari, we compare CURL to (i) SimPLe (Kaiser et al., 2019), the top performing model-based method in terms of data-efficiency on Atari and (ii) Rainbow DQN (Hessel et al., 2017), a top-performing model-free baseline for Atari, (iii) OTRainbow (Kielak, 2020) which is an OverTrained version of Rainbow for data-efficiency, (iv) Efficient Rainbow (van Hasselt et al., 2019) which is a modification of Rainbow hyperparameters for data-efficiency, (v) Random Agent (Kaiser et al., 2019), (vi) Human Performance (Kaiser et al., 2019; van Hasselt et al., 2019). All the baselines and our method are evaluated for performance after 100K *interaction steps* (400K frames with a frame skip of 4) which corresponds to roughly two hours of gameplay. These benchmarks help us understand how the state-of-the-art pixel based RL algorithms compare in terms of sample efficiency and also to human efficiency. **Note:** Scores for SimPLe

Table 1. Scores achieved by CURL (mean & standard deviation for 10 seeds) and baselines on DMControl500k and 1DMControl100k. CURL achieves state-of-the-art performance on the majority (**5** out of **6**) environments benchmarked on DMControl500k. These environments were selected based on availability of data from baseline methods (we run CURL experiments on 16 environments in total and show results in Figure 7). The baselines are PlaNet (Hafner et al., 2018), Dreamer (Hafner et al., 2019), SAC+AE (Yarats et al., 2019), SLAC (Lee et al., 2019), pixel-based SAC and state-based SAC (Haarnoja et al., 2018). SLAC results were reported with one and three gradient updates per agent step, which we refer to as SLACv1 and SLACv2 respectively. We compare to SLACv1 since all other baselines and CURL only make one gradient update per agent step. We also ran CURL with three gradient updates per step and compare results to SLACv2 in Table 5.

500K STEP SCORES	CURL	PLANET	DREAMER	SAC+AE	SLACV1	PIXEL SAC	STATE SAC
FINGER, SPIN	**926 ± 45**	561 ± 284	796 ± 183	884 ± 128	673 ± 92	179 ± 166	923 ± 21
CARTPOLE, SWINGUP	**841 ± 45**	475 ± 71	762 ± 27	735 ± 63	-	419 ± 40	848 ± 15
REACHER, EASY	**929 ± 44**	210 ± 390	793 ± 164	627 ± 58	-	145 ± 30	923 ± 24
CHEETAH, RUN	518 ± 28	305 ± 131	570 ± 253	550 ± 34	**640 ± 19**	197 ± 15	795 ± 30
WALKER, WALK	**902 ± 43**	351 ± 58	897 ± 49	847 ± 48	842 ± 51	42 ± 12	948 ± 54
BALL IN CUP, CATCH	**959 ± 27**	460 ± 380	879 ± 87	794± 58	852 ± 71	312± 63	974 ± 33
100K STEP SCORES							
FINGER, SPIN	**767 ± 56**	136 ± 216	341 ± 70	740 ± 64	693 ± 141	179 ± 66	811±46
CARTPOLE, SWINGUP	**582±146**	297±39	326±27	311±11	-	419±40	835±22
REACHER, EASY	**538±233**	20±50	314±155	274±14	-	145±30	746±25
CHEETAH, RUN	299 ±48	138±88	235± 137	267±24	**319±56**	197±15	616±18
WALKER, WALK	**403±24**	224±48	277±12	394±22	361±73	42±12	891±82
BALL IN CUP, CATCH	**769 ± 43**	0 ± 0	246 ± 174	391± 82	512 ± 110	312± 63	746±91

Table 2. Scores achieved by CURL (coupled with Eff. Rainbow) and baselines on Atari benchmarked at 100k time-steps (Atari100k). CURL achieves state-of-the-art performance on **7** out of **26** environments. Our baselines are SimPLe (Kaiser et al., 2019), OverTrained Rainbow (OTRainbow) (Kielak, 2020), Data-Efficient Rainbow (Eff. Rainbow) (van Hasselt et al., 2019), Rainbow (Hessel et al., 2017), Random Agent and Human Performance (Human). We see that CURL implemented on top of Eff. Rainbow improves over Eff. Rainbow on **19** out of **26** games. We also run CURL with 20 random seeds given that this benchmark is susceptible to high variance across multiple runs. We also see that CURL achieves superhuman performance on JamesBond and Krull.

GAME	HUMAN	RANDOM	RAINBOW	SIMPLE	OTRAINBOW	EFF. RAINBOW	CURL
ALIEN	7127.7	227.8	318.7	616.9	**824.7**	739.9	558.2
AMIDAR	1719.5	5.8	32.5	88.0	82.8	**188.6**	142.1
ASSAULT	742.0	222.4	231	527.2	351.9	431.2	**600.6**
ASTERIX	8503.3	210.0	243.6	**1128.3**	628.5	470.8	734.5
BANK HEIST	753.1	14.2	15.55	34.2	**182.1**	51.0	131.6
BATTLE ZONE	37187.5	2360.0	2360.0	5184.4	4060.6	10124.6	**14870.0**
BOXING	12.1	0.1	-24.8	**9.1**	2.5	0.2	1.2
BREAKOUT	30.5	1.7	1.2	**16.4**	9.84	1.9	4.9
CHOPPER COMMAND	7387.8	811.0	120.0	**1246.9**	1033.33	861.8	1058.5
CRAZY_CLIMBER	35829.4	10780.5	2254.5	**62583.6**	21327.8	16185.3	12146.5
DEMON_ATTACK	1971.0	152.1	163.6	208.1	711.8	508.0	**817.6**
FREEWAY	29.6	0.0	0.0	20.3	25.0	**27.9**	26.7
FROSTBITE	4334.7	65.2	60.2	254.7	231.6	866.8	**1181.3**
GOPHER	2412.5	257.6	431.2	771.0	**778.0**	349.5	669.3
HERO	30826.4	1027.0	487	2656.6	6458.8	**6857.0**	6279.3
JAMESBOND	302.8	29.0	47.4	125.3	112.3	301.6	**471.0**
KANGAROO	3035.0	52.0	0.0	323.1	605.4	779.3	**872.5**
KRULL	2665.5	1598.0	1468	**4539.9**	3277.9	2851.5	4229.6
KUNG_FU_MASTER	22736.3	258.5	0.	**17257.2**	5722.2	14346.1	14307.8
MS_PACMAN	6951.6	307.3	67	**1480.0**	941.9	1204.1	1465.5
PONG	14.6	-20.7	-20.6	**12.8**	1.3	-19.3	-16.5
PRIVATE EYE	69571.3	24.9	0	58.3	100.0	97.8	**218.4**
QBERT	13455.0	163.9	123.46	**1288.8**	509.3	1152.9	1042.4
ROAD_RUNNER	7845.0	11.5	1588.46	5640.6	2696.7	**9600.0**	5661.0
SEAQUEST	42054.7	68.4	131.69	**683.3**	286.92	354.1	384.5
UP_N_DOWN	11693.2	533.4	504.6	**3350.3**	2847.6	2877.4	2955.2

and Human baselines have been reported differently in prior work (Kielak, 2020; van Hasselt et al., 2019). To be rigorous, we take the *best* reported score for each individual game reported in prior work.

6. Results

6.1. DMControl

Sample-efficiency results for DMControl experiments are shown in Table 1 and in Figures 4, 6, and 7. Below are the key findings:

(i) CURL is the **state-of-the-art image-based RL algorithm** on the majority (**5** out of **6**) DMControl environments that we benchmark on for sample-efficiency against existing pixel-based baselines. On DMControl100k, CURL achieves **1.9x** higher median performance than Dreamer (Hafner et al., 2019), a leading model-based method, and is **4.5x** more data-efficient shown in Figure 6.

(ii) CURL operating purely from pixels **nearly matches** (and sometimes surpasses) **the sample efficiency of SAC operating from state** on the majority of 16 DMControl environments tested shown in Figure 7 and matches the median state-based score on DMControl500k shown in Figure 4. This is a **first** for any image-based RL algorithm, be it model-based, model-free, with or without auxiliary tasks.

(iii) CURL solves (converges close to optimal score of 1000) on the majority of 16 DMControl experiments within **500k** steps. It also matches the state-based median score across the 6 extensively benchmarked environments in this regime.

6.2. Atari

Results for Atari100k are shown in Table 2. Below are the key findings:

(i) CURL achieves a median human-normalized score (HNS) of **17.5%** while SimPLe and Efficient Rainbow DQN achieve 14.4% and 16.1% respectively. The mean HNS is 38.1%, 44.3%, and 28.5% for CURL, SimPLe, and Efficient Rainbow DQN respectively.

(ii) CURL improves on top of Efficient Rainbow on **19** out of **26** Atari games. Averaged across 26 games, CURL improves on top of Efficient Rainbow by **1.3x**, while the median performance improvement over SimPLE and Efficient Rainbow are **1.2x** and **1.1x** respectively.

(iii) CURL **surpasses human performance** on two games JamesBond (1.6 HNS), Krull (2.5 HNS).

7. Ablation Studies

In Appendix E, we present the results of ablation studies carried out to answer the following questions: (i) Does CURL learn only visual features or does it also capture temporal dynamics of the environment? (ii) How well does the RL policy perform if CURL representations are learned solely with the contrastive objective and no signal from RL? (iii) Why does CURL match state-based RL performance on some DMControl environments but not on others?

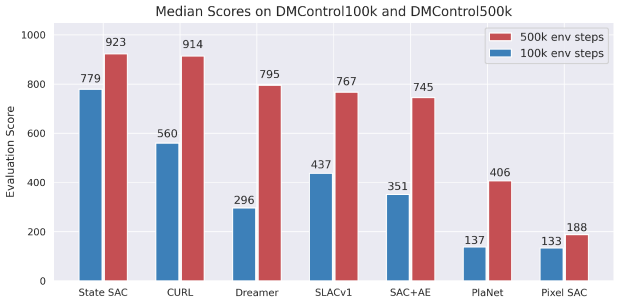

Figure 4. Performance of CURL coupled to SAC averaged across 10 seeds relative to SLACv1, PlaNet, Pixel SAC and State SAC baselines. At the 500k benchmark CURL matches the median score of state-based SAC. At 100k environment steps CURL achieves a 1.9x higher median score than Dreamer. For a direct comparison, we only compute the median across the 6 environments in 1 (4 for SLAC) and show learning curves for CURL across 16 DMControl experiments in 7.

8. Conclusion

In this work, we proposed CURL, a contrastive unsupervised representation learning method for RL, that achieves state-of-the-art data-efficiency on pixel-based RL tasks across a diverse set of benchmark environments. CURL is the first model-free RL pipeline accelerated by contrastive learning with minimal architectural changes to demonstrate state-of-the-art performance on complex tasks so far dominated by approaches that have relied on learning world models and (or) decoder-based objectives. We hope that progress like CURL enables avenues for real-world deployment of RL in areas like robotics where data-efficiency is paramount.

9. Acknowledgements

This research is supported in part by DARPA through the Learning with Less Labels (LwLL) Program and by ONR through PECASE N000141612723. We also thank Zak Stone and Google TFRC for cloud credits; Danijar Hafner, Alex Lee, and Denis Yarats for sharing data for baselines; and Lerrel Pinto, Adam Stooke, Will Whitney, and Ankesh Anand for insightful discussions.

References

Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A., and Hjelm, R. D. Unsupervised state representation learning in atari. In *Advances in Neural Information Processing Systems*, pp. 8766–8779, 2019.

Bachman, P., Hjelm, R. D., and Buchwalter, W. Learning representations by maximizing mutual information across views. In *Advances in Neural Information Processing Systems*, pp. 15509–15519, 2019.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 449–458. JMLR. org, 2017.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. A simple framework for contrastive learning of visual representations, 2020.

Chopra, S., Hadsell, R., and LeCun, Y. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pp. 539–546. IEEE, 2005.

Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. V. Randaugment: Practical automated data augmentation with a reduced search space, 2019.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Dwibedi, D., Tompson, J., Lynch, C., and Sermanet, P. Learning actionable representations from visual observations. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1577–1584. IEEE, 2018.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Ghosh, P., Sajjadi, M. S. M., Vergari, A., Black, M., and Schlkopf, B. From variational to deterministic autoencoders, 2019.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

Hadsell, R., Chopra, S., and LeCun, Y. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pp. 1735–1742. IEEE, 2006.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019a.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. Momentum contrast for unsupervised visual representation learning. 2019b.

Hénaff, O. J., Srinivas, A., De Fauw, J., Razavi, A., Doersch, C., Eslami, S., and van den Oord, A. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning, 2017.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. Darla: Improving zero-shot transfer in reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1480–1490. JMLR. org, 2017.

Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Kaiser, L., Babaeizadeh, M., Milos, P., Osinski, B., Campbell, R. H., Czechowski, K., Erhan, D., Finn, C., Kozakowski, P., Levine, S., et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

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

Kielak, K. Do recent advancements in model-based deep reinforcement learning really improve data efficiency?, 2020.

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

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

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

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

LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. 2006.

Lee, A. X., Nagabandi, A., Abbeel, P., and Levine, S. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

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

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Schmidhuber, J. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. *Technical Report FKI-126-90, TUM*, 1990.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Schroff, F., Kalenichenko, D., and Philbin, J. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823, 2015.

Sermanet, P., Lynch, C., Chebotar, Y., Hsu, J., Jang, E., Schaal, S., Levine, S., and Brain, G. Time-contrastive networks: Self-supervised learning from video. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1134–1141. IEEE, 2018.

Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.

Sutton, R. S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pp. 216–224. Elsevier, 1990.

Sutton, R. S. et al. *Introduction to reinforcement learning*, volume 135. 1998.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015. URL http://arxiv.org/abs/1409.4842.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Tian, Y., Krishnan, D., and Isola, P. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

Tschannen, M., Djolonga, J., Rubenstein, P. K., Gelly, S., and Lucic, M. On mutual information maximization for representation learning. *arXiv preprint arXiv:1907.13625*, 2019.

van den Oord, A., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems*, pp. 14322–14333, 2019.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103, 2008.

Wang, X. and Gupta, A. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2015.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

Wu, Z., Xiong, Y., Yu, S., and Lin, D. Unsupervised feature learning via non-parametric instance-level discrimination. *arXiv preprint arXiv:1805.01978*, 2018.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

A. Implementation Details

Below, we explain the implementation details for CURL in the DMControl setting. Specifically, we use the SAC algorithm as the RL objective coupled with CURL and build on top of the publicly released implementation from Yarats et al. (2019). We present in detail the hyperparameters for the architecture and optimization. We do not use any extra hyperparameter for balancing the contrastive loss and the reinforcement learning losses. Both the objectives are weighed equally in the gradient updates.

Table 3. Hyperparameters used for DMControl CURL experiments. Most hyperparameters values are unchanged across environments with the exception for action repeat, learning rate, and batch size.

Hyperparameter	**Value**
Random crop	True
Observation rendering	$(100, 100)$
Observation downsampling	$(84, 84)$
Replay buffer size	100000
Initial steps	1000
Stacked frames	3
Action repeat	2 finger, spin; walker, walk 8 cartpole, swingup 4 otherwise
Hidden units (MLP)	1024
Evaluation episodes	10
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$	$(.9, .999)$
$(\beta_1, \beta_2) \rightarrow (\alpha)$	$(.5, .999)$
Learning rate $(f_\theta, \pi_\psi, Q_\phi)$	$2e-4$ cheetah, run $1e-3$ otherwise
Learning rate (α)	$1e-4$
Batch Size	512 (cheetah), 128 (rest)
Q function EMA τ	0.01
Critic target update freq	2
Convolutional layers	4
Number of filters	32
Non-linearity	ReLU
Encoder EMA τ	0.05
Latent dimension	50
Discount γ	.99
Initial temperature	0.1

Architecture: We use an encoder architecture that is similar to (Yarats et al., 2019), which we sketch in PyTorch-like pseudocode below. The actor and critic both use the same encoder to embed image observations. A full list of hyperparameters is displayed in Table 3.

For contrastive learning, CURL utilizes momentum for the key encoder (He et al., 2019b) and a bi-linear inner product as the similarity measure (van den Oord et al., 2018). Performance curves ablating these two architectural choices are shown in Figure 5.

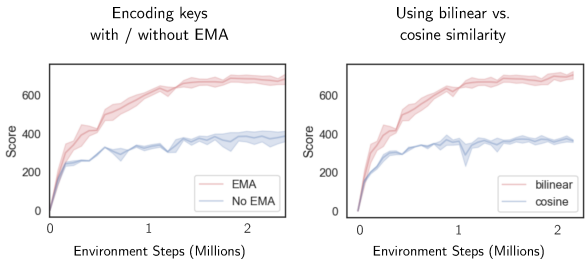

Figure 5. Performance on cheetah-run environment ablated two-ways: (left) using the query encoder or exponentially moving average of the query encoder for encoding keys (right) using the bi-linear inner product as in (van den Oord et al., 2018) or the cosine inner product as in He et al. (2019b); Chen et al. (2020)

Pseudo-code for the architecture is provided below:

```
def encode(x,z_dim):
    """
    ConvNet encoder
    args:
        B-batch_size, C-channels
        H,W-spatial_dims
        x : shape : [B, C, H, W]
        C = 3 * num_frames; 3 - R/G/B
        z_dim: latent dimension
    """

    x = x / 255.

    # c: channels, f: filters
    # k: kernel, s: stride

    z = Conv2d(c=x.shape[1], f=32, k=3, s=2)](
        x)
    z = ReLU(z)

    for _ in range(num_layers - 1):
        z = Conv2d((c=32, f=32, k=3, s=1))(z)
        z = ReLU(z)

    z = flatten(z)

    # in: input dim, out: output_dim, h:
        hiddens

    z = mlp(in=z.size(),out=z_dim,h=1024)
    z = LayerNorm(z)
    z = tanh(z)
```

Terminology: A common point of confusion is the meaning "training steps." We use the term *environment steps* to denote the amount of times the simulator environment is stepped through and *interaction steps* to denote the number of times the agent steps through its policy. The terms *action repeat* or *frame skip* refer to the number of times an action

is repeated when it's drawn from the agent's policy. For example, if action repeat is set to 4, then 100k interaction steps is equivalent to 400k environment steps.

Batch Updates: After initializing the replay buffer with observations extracted by a random agent, we sample a batch of observations, compute the CURL objectives, and step through the optimizer. Note that since queries and keys are generated by data-augmenting an observation, we can generate arbitrarily many keys to increase the contrastive batch size without sampling any additional observations.

Shared Representations: The objective of performing contrastive learning together with RL is to ensure that the shared encoder learns rich features that facilitate sample efficient control. There is a subtle coincidental connection between MoCo and off-policy RL. Both the frameworks adopt the usage of a momentum averaged (EMA) version of the underlying model. In MoCo, the EMA encoder is used for encoding the keys (targets) while in off-policy RL, the EMA version of the Q-networks are used as targets in the Bellman error (Mnih et al., 2015; Haarnoja et al., 2018). Thanks to this connection, CURL shares the convolutional encoder, momentum coefficient and EMA update between contrastive and reinforcement learning updates for the shared parameters. The MLP part of the critic that operates on top of these convolutional features has a separate momentum coefficient and update decoupled from the image encoder parameters.

Balancing Contrastive and RL Updates: While past work has learned hyperparameters to balance the auxiliary loss coefficient or learning rate relative to the RL objective (Jaderberg et al., 2016; Yarats et al., 2019), CURL does not need any such adjustments. We use both the contrastive and RL objectives together with equal weight and learning rate. This simplifies the training process compared to other methods, such as training a VAE jointly (Hafner et al., 2018; 2019; Lee et al., 2019), that require careful tuning of coefficients for representation learning.

Differences in Data Collection between Computer Vision and RL Settings: There are two key differences between contrastive learning in the computer vision and RL settings because of their different goals. Unsupervised feature learning methods built for downstream vision tasks like image classification assume a setting where there is a large static dataset of unlabeled images. On the other hand, in RL, the dataset changes over time to account for the agent's new experiences. Secondly, the size of the memory bank of labeled images and dataset of unlabeled ones in vision-based settings are 65K and 1M (or 1B) respectively. The goal in vision-based methods is to learn from millions of unlabeled images. On the other hand, the goal in CURL is to develop sample-efficient RL algorithms. For example, to be able to solve a task within 100K timesteps (approximately 2 hours in real-time), an agent can only ingest 100K image frames. Therefore, unlike MoCo, CURL does not use a memory bank for contrastive learning. Instead, the negatives are constructed on the fly for every minibatch sampled from the agent's replay buffer for an RL update similar to SimCLR. The exact implementation is provided as a PyTorch-like code snippet in 4.7.

Data Augmentation:

Random crop data augmentation has been crucial for the performance of deep learning based computer vision systems in object recognition, detection and segmentation (Krizhevsky et al., 2012; Szegedy et al., 2015; Cubuk et al., 2019; Chen et al., 2020). However, similar augmentation methods have not seen much adoption in the field of RL even though several benchmarks use raw pixels as inputs to the model.

CURL adopts the random crop data augmentation as the stochastic data augmentation applied to a frame stack. To make it easier for the model to correlate spatio-temporal patterns in the input, we apply the same random crop (in terms of box coordinates) across all four frames in the stack as opposed to extracting different random crop positions from each frame in the stack. Further, unlike in computer vision systems where the aspect ratio for random crop is allowed to be as low as 0.08, we preserve much of the spatial information as possible and use a constant aspect ratio of 0.84 between the original and cropped. In our experiments, data augmented samples for CURL are formed by cropping 84×84 frames from an input frame of 100×100.

DMControl: We render observations at 100×100 and randomly crop 84×84 frames. For evaluation, we render observations at 100×100 and center crop to 84×84 pixels. We found that implementing random crop efficiently was extremely important to the success of the algorithm. We provide pseudocode below:

```
from skimage import view_as_windows
import numpy as np

def random_crop(imgs, out):
    """
    Vectorized random crop
    args:
       imgs: shape (B,C,H,W)
       out: output size (e.g. 84)
    """

    # n: batch size.
    n = imgs.shape[0]
    img_size = imgs.shape[-1] # e.g. 100
    crop_max = img_size - out

    imgs = np.transpose(imgs, (0, 2, 3, 1))

    w1 = np.random.randint(0, crop_max, n)
    h1 = np.random.randint(0, crop_max, n)

    # creates all sliding window
    # combinations of size (out)

    windows = view_as_windows(
       imgs, (1, out, out, 1))[..., 0,:,:, 0]
```

```
    # selects a random window
    # for each batch element
    cropped = windows[np.arange(n), w1, h1]
    return cropped
```

B. Atari100k Implementation Details

The flexibility of CURL allows us to apply it to discrete control setting with minimal modifications. Similar to our rationale for picking SAC as the baseline RL algorithm to couple CURL with (for continuous control), we pick the data-efficient version of Rainbow DQN (Efficient Rainbow) (van Hasselt et al., 2019) for Atari100K which performs competitively with an older version of SimPLe (most recent version has improved numbers). In order to understand *specifically* what the gains from CURL are without any other changes, we adopt the *exact* same hyperparameters specified in the paper (van Hasselt et al., 2019) (including a modified convolutional encoder that uses larger kernel size and stride of 5). We present the details in Table 4. Similar to DMControl, the contrastive objective and the RL objective are weighted equally for learning (except for Pong, Freeway, Boxing and PrivateEye for which we used a coefficient of 0.05 for the momentum contastive loss. On a large majority (22 out of 26) of the games, we do not use this adjustment. While it is standard practice to use the same hyperparameters for all games in Atari, papers proposing auxiliary losses have adopted a different practice of using game specific coefficients (Jaderberg et al., 2016).). We use the Efficient Rainbow codebase from `https://github.com/Kaixhin/Rainbow` which has a reproduced version of van Hasselt et al. (2019). We evaluate with 20 random seeds and report the mean score for each game given the high variance nature of the Atari100k steps benchmark. We restrict ourselves to using grayscale renderings of image observations and use random crop of frame stack as data augmentation.

Table 4. Hyperparameters used for Atari100K CURL experiments. Hyperparameters are unchanged across games.

Hyperparameter	**Value**
Random crop	True
Image size	$(84, 84)$
Data Augmentation	Random Crop (Train)
Replay buffer size	100000
Training frames	400000
Training steps	100000
Frame skip	4
Stacked frames	4
Action repeat	4
Replay period every	1
Q network: channels	32, 64
Q network: filter size	$5 \times 5, 5 \times 5$
Q network: stride	5, 5
Q network: hidden units	256
Momentum (EMA for CURL) τ	0.001
Non-linearity	ReLU
Reward Clipping	$[-1, 1]$
Multi step return	20
Minimum replay size for sampling	1600
Max frames per episode	108K
Update	Distributional Double Q
Target Network Update Period	every 2000 updates
Support-of-Q-distribution	51 bins
Discount γ	0.99
Batch Size	32
Optimizer	Adam
Optimizer: learning rate	0.0001
Optimizer: $\beta 1$	0.9
Optimizer: $\beta 2$	0.999
Optimizer ϵ	0.000015
Max gradient norm	10
Exploration	Noisy Nets
Noisy nets parameter	0.1
Priority exponent	0.5
Priority correction	$0.4 \rightarrow 1$
Hardware	CPU

C. Benchmarking Data Efficiency

Tables 1 and 2 show the episode returns of DMControl100k, DMControl500k, and Atari100k across CURL and a number of pixel-based baselines. CURL outperforms all baseline pixel-based methods across experiments on both DMControl100k and DMControl500k. On Atari100k experiments, CURL coupled with Eff Rainbow outperforms the baseline on the majority of games tested (19 out of 26 games).

D. Further Investigation of Data-Efficiency in Contrastive RL

To further benchmark CURL's sample-efficiency, we compare it to state-based SAC on a total of 16 DMControl environments. Shown in Figure 7, CURL matches state-based data-efficiency on most of the environments, but lags behind state-based SAC on more challenging environments.

E. Ablations

E.1. Learning Temporal Dynamics

To gain insight as to whether CURL learns temporal dynamics across the stacked frames, we also train a variant of CURL where the discriminants are individual frames as opposed to stacked ones. This can be done by sampling stacked frames from the replay buffer but only using the first frame to update the contrastive loss:

```
f_q = x_q[:,:3,...] # (B,C,H,W), C=9.
f_k = x_k[:,:3,...]
```

During the actor-critic update, frames in the batch are en-

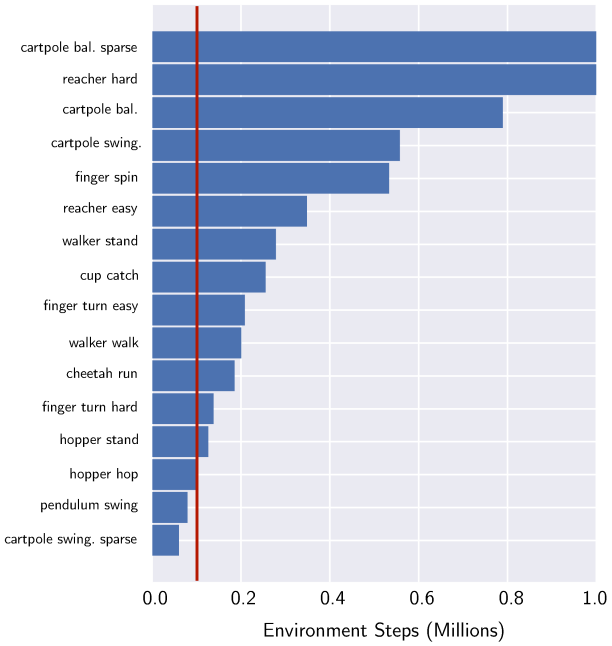

Figure 6. The number of steps it takes a prior leading pixel-based method, Dreamer, to achieve the same score that CURL achieves at 100k training steps (clipped at 1M steps). On average, CURL is 4.5x more data-efficient. We chose Dreamer because the authors (Hafner et al., 2019) report performance for all of the above environments while other baselines like SLAC and SAC+AE only benchmark on 4 and 6 environments, respectively. For further comparison of CURL with these methods, the reader is referred to Table 1 and Figure 4.

coded individually into latent codes, which are then concatenated before being passed to a dense network.

```
# x: (B,C,H,W), C=9.
z1 = encode(x[:,:3,...])
z2 = encode(x[:,3:6,...])
z3 = encode(x[:,6:9,...])
z = torch.cat([z1,z2,z3],-1)
```

Encoding each frame indiviudally ensures that the contrastive objective only has access to visual discriminants. Comparing the visual and spatiotemporal variants of CURL in Figure 8 shows that the variant trained on stacked frames outperforms the visual-only version in most environments. The only exceptions are reacher and ball-in-cup environments. Indeed, in those environments the visual signal is strong enough to solve the task optimally, whereas in other environments, such as walker and cheetah, where balance or coordination is required, visual information alone is insufficient.

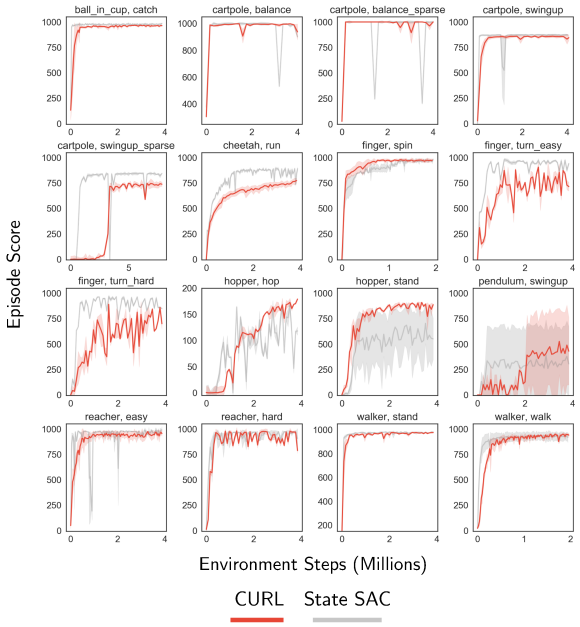

Figure 7. CURL compared to state-based SAC run for 3 seeds on each of 16 selected DMControl environments. For the 6 environments in 4, CURL performance is averaged over 10 seeds.

E.2. Increasing Gradient Updates per Agent Step

Although most baselines we benchmark against use one gradient update per agent step, it was recently empirically shown that increasing the ratio of gradients per step improves data-efficiency in RL (Kielak, 2020). This finding is also supported by SLAC (Lee et al., 2019), where results are shown with a ratio of 1:1 (SLACv1) and 3:1 (SLACv2). We

Table 5. Scores achieved by CURL and SLAC when run with a 3:1 ratio of gradient updates per agent step on DMControl500k and DMControl100k. CURL achieves state-of-the-art performance on the majority (**3** out of **4**) environments on DMControl500k. Performance of both algorithms is improved relative to the 1:1 ratio reported for all baselines in Table 1 but at the cost of significant compute and wall-clock time overhead.

DMControl500k	CURL	SLACv2
Finger, spin	**923** $\pm$ **50**	884 $\pm$ 98
Walker, walk	**911** $\pm$ **35**	891 $\pm$ 60
Cheetah, run	545 $\pm$ 39	**791** $\pm$ **37**
Ball in cup, catch	**948** $\pm$ **21**	885 $\pm$ 154
DMControl100k	CURL	SLACv2
Finger, spin	**741** $\pm$ **118**	728 $\pm$212
Walker, walk	428 $\pm$ 59	**513** $\pm$ **41**
Cheetah, run	314 $\pm$ 46	**438** $\pm$ **76**
Ball in cup, catch	**899** $\pm$ **47**	837 $\pm$ 147

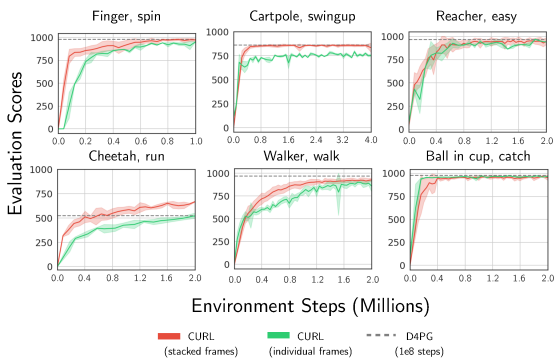

Figure 8. CURL with temporal and visual discrimination (red) compared to CURL with only visual discrimination (green). In most settings, the variant with temporal variant outperforms the purely visual variant of CURL. The two exceptions are reacher and ball in cup environments, suggesting that learning dynamics is not necessary for those two environments. Note that the walker environment was run with action repeat of 4, whereas walker walk in the main results Table 1 and Figure 7 was run with action repeat of 2.

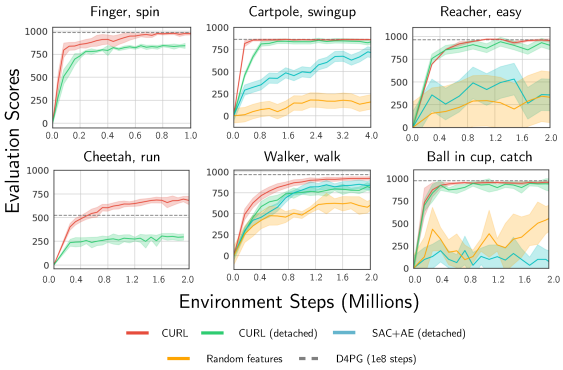

Figure 9. CURL where the CNN part of the encoder receives gradients from both the contrastive loss and critic (red) compared to CURL with the convolutional part of the encoder trained only with the contrastive objective (green). The detached encoder variant is able to learn representations that enable near-optimal learning on most environments, except for cheetah. As in Figure 8, the walker environment was run with action repeat of 4.

E.3. Decoupling Representation Learning from Reinforcement Learning

Typically, Deep RL representations depend almost entirely on the reward function specific to a task. However, handcrafted representations such as the proprioceptive state are independent of the reward function. It is much more desirable to learn reward-agnostic representations, so that the same representation can be re-used across different RL tasks. We test whether CURL can learn such representations by comparing CURL to a variant where the critic gradients are backpropagated through the critic and contrastive dense feedforward networks but stopped before reaching the convolutional neural network (CNN) part of the encoder.

Scores displayed in Figure 9 show that for many environments, the detached CNN representations are sufficient to learn an optimal policy. The major exception is the cheetah environment, where the detached representation significantly under-performs. Though promising, we leave further exploration of task-agnostic representations for future work.

E.4. Predicting State from Pixels

Despite improved sample-efficiency on most DMControl tasks, there is still a visible gap between the performance of SAC on state and SAC with CURL in some environments. Since CURL learns representations by performing instance discrimination across stacks of three frames, it's possible that the reason for degraded sample-efficiency on more challenging tasks is due to partial-observability of the ground truth state.

To test this hypothesis, we perform supervised regression (X, Y) from pixels X to the proprioceptive state Y, where each data point $x \in X$ is a stack of three consecutive frames and $y \in Y$ is the corresponding state extracted from the simulator. We find that the error in predicting the state from pixels correlates with the policy performance of pixel-based methods. Test-time error rates displayed in Figure 10 show that environments that CURL solves as efficiently as state-based SAC have low error-rates in predicting the state from stacks of pixels. The prediction error increases for more challenging environments, such as cheetah-run and walker-walk. Finally, the error is highest for environments where current pixel-based methods, CURL included, make no progress at all (Tassa et al., 2018), such as humanoid and swimmer.

This investigation suggests that degraded policy performance on challenging tasks may result from the lack of requisite information about the underlying state in the pixel data used for learning representations. We leave further investigation for future work.

E.5. CURL + Efficient Rainbow Atari runs

We report the scores (Tables 6 and 7) for 20 seeds across the 26 Atari games in the Atari100k benchmark for CURL cou-

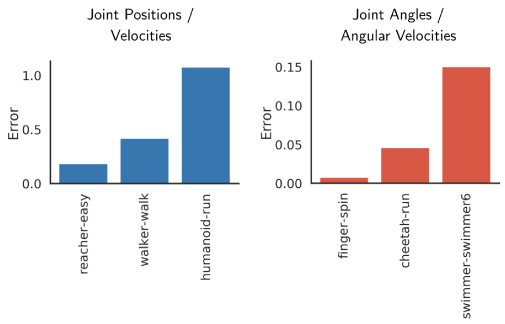

Figure 10. Test-time mean squared error for predicting the proprioceptive state from pixels on a number of DMControl environments. In DMControl, environments fall into two groups - where the state corresponds to either (a) positions and velocities of the robot joints or (b) the joint angles and angular velocities.

pled with Efficient Rainbow. The variance across multiple seeds is considerably high in this benchmark. Therefore, we report the scores for each of the seeds along with the mean and standard deviation for each game.

F. Document changelog

This document tracks the progress and changes of CURL. In order to help readers be aware of and understand the changes, here is a brief summary:

v1 Initial version.

v2 Minor changes to DMControl to account for frame skip factor when evaluating data-efficiency of CURL and baselines. Changed action repeat for the Walker-walk task from 4 to 2 to match baseline implementations.

v3 ICML 2020 Camera Ready. For our Atari experiments, we moved to the `https://github.com/Kaixhin/Rainbow` codebase for easy and clean benchmarking that directly builds on top of Efficient Rainbow without other changes. We also run 20 seeds as opposed to 3 seeds earlier given the high variance nature of the benchmark.

G. Connection to work on data augmentations

Recently, there have been two papers published on using data augmentations for reinforcement learning, RAD (Laskin et al., 2020) and DrQ (Kostrikov et al., 2020). These two papers present the version of CURL without an auxiliary contrastive loss but rather directly feeding in the augmented views of the image observations to the underlying value / policy network(s). Both RAD and DrQ present results on both continuous and discrete control environments, surpassing the results presented in CURL on both the DMControl and Atari benchmarks. Plenty of researchers have opined in public forums whether the results in RAD and DrQ make CURL irrelevant if the objective is to use data augmentations for data-efficient reinforcement learning. We believe that answering this question needs more nuance and present our opinions below:

1. If one has access to a rich stream of rewards from the underlying environment and is interested in optimzing the performance in terms of average reward, RAD and DrQ are likely to work better than CURL. The reason for this is simply that RAD and DrQ directly optimize for the objective one cares about, while CURL introduces an additional auxiliary consistency objective.

2. If one *does not* have access to a rich stream of rewards and is interested in learning good latent spaces in a *task agnostic manner* that can allow for data-efficient controllers across multiple tasks, CURL is the only option since the contrastive objective in CURL is reward independent. Our ablation on the detached encoder with the CURL objective present evidence that one could build simple MLPs on top of the CURL features without fine-tuning the underlying encoder and still be data-efficient on many of the DMControl tasks.

3. Future work in data-efficient reinforcement learning, particularly for real world settings, is likely to require approaches that *do not rely on reward functions*. In such scenarios, CURL is likely to be the more preferred approach. Further, one could potentially use CURL in a scenario where unsupervised pre-training without reward functions is initially performed before fine-tuning to the RL objective across multiple tasks.

Given the above reasons, there isn't a straightforward answer as to which is the better algorithm and the answer really depends on what the researcher / practioner wants to solve. We also emphasize that CURL was the first approach that used data augmentations effectively to significantly improve the data-efficiency of model-free reinforcement learning methods with very simple changes and showed improvement over relatively more complex model-based methods. The augmentations and results in CURL inspired future work in the form of RAD and DrQ. We hope that the analysis and results presented in CURL encourage researchers to employ data augmentations, contrastive losses and unsupervised pre-training for future reinforcement learning research.

Pacman	Frostbite	Asterix	KungFuMaster	Kangaroo	Gopher	RoadRunner	JamesBond	BattleZone	Seaquest	Assault	Krull	Qbert
1287	2292	850	8470	600	1036	2820	305	18100	322	634.2	3404.3	1020
1608	1046	525	10870	2280	574	3190	265	18200	236	696.8	2443.5	650
1466	1209	655	10920	1940	540	7840	335	26800	352	655.2	6791.4	830
1430	255	565	7730	1140	618	12060	145	21300	386	443	3022.5	902.5
1114	426	715	17525	520	534	8340	565	7900	458	546	3892.2	3957.5
1083	2280	715	3560	600	596	6920	565	8100	224	564.9	3505.5	772.5
2301	259	770	10940	600	502	2230	350	12000	282	514.4	2564.1	782.5
1128	335	980	23420	900	998	4250	365	16500	339	516.6	4079.7	727.5
1184	1409	665	15160	600	950	1570	140	23900	526	661.5	2376.4	705
1510	258	610	15370	730	544	6300	425	19900	436	664.5	4161.8	757.5
2343	335	905	22260	600	796	3100	315	10000	272	529	3311.1	647.5
1063	1062	800	17320	880	522	1060	335	11200	428	445.2	2517.3	562.5
2040	1542	675	31820	220	392	6050	735	9700	358	573.3	3764.7	2425
1195	1102	795	23360	920	780	11810	950	23500	533	531.3	10150.2	1112.5
1343	2461	585	27460	600	792	4630	520	10500	968	663.6	2883.6	527.5
1354	257	865	7770	2300	454	2530	755	18100	314	795.3	5123.7	472.5
1925	513	730	8820	320	564	6840	750	9000	378	633	3652.5	610
1228	1826	680	2980	600	522	6580	795	8900	168	674.1	2376.4	697.5
1099	1889	965	10100	600	496	10720	450	10700	242	604.8	11745	1847.5
1608	2869	640	10300	500	1176	4380	355	13100	467	665.7	2826	840
1465.5	1181.3	734.5	14307.8	872.5	669.3	5661	471	14870	384.5	600.6	4229.6	1042.4
397.5	856.2	129.8	7919.3	600.1	220.6	3289.3	226.2	5964.3	170.2	89.5	2540.6	828.4

Table 6. CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.

UpNDown	Hero	CrazyClimber	ChopperComm.	DemonAttack	Amidar	Alien	BankHeist	Breakout	Freeway	Pong	PrivateEye	Boxing
3529	8747.5	19090	560	611.5	150.9	616	95	3.6	29.2	-19.3	100	-0.5
772	3026	8290	1530	707.5	131.2	923	184	5	25.4	-16.9	100	-11.4
5972	7146	12160	1390	843.5	141.5	467	75	3.2	27.6	-12	100	4
2793	7686	8920	1100	330.5	133.7	441	232	5.1	28.6	-19.6	100	3.6
3546	7335	11360	500	759	157.1	716	187	2.9	22.8	-17.8	1357.4	6.2
4552	7325	4110	990	940	125.4	453	367	6.3	29.6	-18.9	100	5
2972	7275.5	9460	780	1136	183.2	273	186	5.9	23.3	-15.9	0	-1.7
2865	3115	20630	1180	758	153.6	540	68	2.6	27.6	-15.2	100	0.1
3098	7424	6780	1380	772.5	127.8	499	60	5.9	26.1	-18.7	100	3.5
1953	7475	13570	970	820	149.4	475	123	4.3	28.3	-13.3	100	-0.5
1467	3135	11890	1200	784	125.7	553	72	3.2	21.8	-17.2	1510	-22.1
2912	5060.5	9160	1130	1080	130.4	446	53	4.8	21.8	-20.1	100	-1.8
4123	4409	10960	1380	847	133	533	68	6.3	28.9	-16.5	100	1.6
2334	6979	17360	1230	771.5	140.5	968	36	7.3	28.2	-14.9	100	3.6
2605	4159	8930	1350	907.5	133.8	499	53	4.8	28.3	-19.3	100	-17.6
2432	7560	11510	1080	1095.5	191.8	523	105	3.7	26.8	-15.6	0	21.7
3826	8587	22690	1210	700	115.5	616	276	6.6	27.5	-21	100	2
3052	4683.5	8120	840	803.5	164	475	69	5.5	26.5	-10.5	0	5.9
3131	7317	13500	730	818	131.7	525	50	4.3	26.8	-13.3	100	18.7
1169	7141	14440	640	866	122.4	622	273	6.2	28.6	-13.1	100	3.7
2955.2	6279.3	12146.5	1058.5	817.6	142.1	558.2	131.6	4.9	26.7	-16.5	218.4	1.2
1181.1	1871.5	4765.6	299.1	176.6	20.0	160.3	94.4	1.4	2.4	2.9	417.9	10.0

Table 7. CURL implemented on top of Efficient Rainbow - Scores reported for 20 random seeds for each of the above games, with the last two rows being the mean and standard deviation across the runs.