MASKED CONTRASTIVE REPRESENTATION LEARNING FOR REINFORCEMENT LEARNING

Jinhua Zhu[1], **Yingce Xia**[2], **Lijun Wu**[2],
Jiajun Deng[1],**Wengang Zhou**[1], **Tao Qin**[2], **Houqiang Li**[1]
[1]University of Science and Technology of China;
[2]Microsoft Research;
[1]{teslazhu,dengjj}@mail.ustc.edu.cn, {zhwg,lihq}@ustc.edu.cn
[2]{yingce.xia,lijuwu,taoqin}@microsoft.com

ABSTRACT

Improving sample efficiency is a key research problem in reinforcement learning (RL), and CURL, which uses contrastive learning to extract high-level features from raw pixels of individual video frames, is an efficient algorithm (Srinivas et al., 2020). We observe that consecutive video frames in a game are highly correlated but CURL deals with them independently. To further improve data efficiency, we propose a new algorithm, masked contrastive representation learning for RL, that takes the correlation among consecutive inputs into consideration. In addition to the CNN encoder and the policy network in CURL, our method introduces an auxiliary Transformer module to leverage the correlations among video frames. During training, we randomly mask the features of several frames, and use the CNN encoder and Transformer to reconstruct them based on the context frames. The CNN encoder and Transformer are jointly trained via contrastive learning where the reconstructed features should be similar to the ground-truth ones while dissimilar to others. During inference, the CNN encoder and the policy network are used to take actions, and the Transformer module is discarded. Our method achieves consistent improvements over CURL on 14 out of 16 environments from DMControl suite and 21 out of 26 environments from Atari 2600 Games. The code is available at https://github.com/teslacool/m-curl.

1 INTRODUCTION

Recently, reinforcement learning (RL) has achieved great success in game AI (Silver et al., 2018; Vinyals et al., 2019), robotics (Gu et al., 2017; Bousmalis et al., 2018), logistic (Li et al., 2019), etc. In many settings, the dimensions of the raw inputs are high, which is hard to directly work on and sample-inefficient (Oord et al., 2018; Srinivas et al., 2020). Alternatively, we can first learn an encoder, to map the high-dimensional input into low-dimensional representations, which will be further fed into the policy networks to get the actions. How to efficiently learn the encoder is an important problem for RL, and numerous methods have been proposed to address this problem. Among them, one line is introducing an auxiliary loss for learning better latent representations (Jaderberg et al., 2016; Oord et al., 2018; Lee et al., 2019; Yarats et al., 2019), and another line is using one world model to sample rollouts and plan in the learned pixel or latent space (Hafner et al., 2019b;a; Kaiser et al., 2019).

Recently, contrastive learning has attracted more and more attention in machine learning and computer vision communities (He et al., 2020; Hénaff et al., 2019). Contrastive learning is an instance-level pre-training technique to get an encoder. Taking image processing as an example, we first use the encoder to extract representations for a collection of images, which are saved into a dictionary. An encoded query should be similar to the matching one in the dictionary and dissimilar to others. Such an idea is adapted into the first line aforementioned (Oord et al., 2018; Srinivas et al., 2020) due to its efficiency. One of the representative methods is CURL (Srinivas et al., 2020), which is short for contrastive unsupervised representations for RL. CURL aims to learn the encoder in a

self-supervised manner, where an input x is augmented (e.g., random crop) into two variants x_q (query) and x_+ (matching key). The negative samples x_-'s are randomly selected from the replay buffer (Mnih et al., 2013), where x is excluded. After processed by the encoder, x_q should be similar to x_+ and dissimilar to the sampled x_-'s.

Although CURL makes great success in pixel-based RL, it overlooks the relevance among the inputs. After receiving an input from the environment, we take an action, get an instant reward, and observe another input. However, in pixel-based RL games, consecutive frames are highly correlated while this property is ignored. Intuitively, a good encoder should not only provide an accurate representation of the current input, but also remember the previous inputs and be predictable for future ones.

In this paper, we propose a new algorithm, masked contrastive representation learning for RL (M-CURL), to further improve data efficiency and performance of pixel-based RL. Apart from the convolutional neural network (CNN) as image encoder used in previous works (e.g., CURL), we introduce an auxiliary Transformer module (Vaswani et al., 2017), the state-of-the-art structure to model sequences, to capture the relevance. To achieve that, we first sample a set of consecutive transitions from the replay buffer and regard them as a sequence. We then use the CNN encoder to map each selected frames (i.e., images) into low-dimensional representations. Inspired by the success of masked pre-training in natural language processing like BERT (Devlin et al., 2018) and RoBERTa (Lample & Conneau, 2019), we randomly mask out several representations and let Transformer module reconstruct them. Finally, we use contrastive loss to jointly train the CNN encoder and Transformer, where the reconstructed features should be similar to the ground-truth ones while dissimilar to others.

The contribution of our work is summarized as follows: (1) We propose M-CURL, an effective representation learning method for RL leveraging the advantages of masked pre-training and contrastive learning; (2) Our method is a general one and can be combined with any off-policy algorithm with a replay buffer, such as SAC (Haarnoja et al., 2018) and rainbown DQN (Mnih et al., 2013) (we can also add one replay buffer for on-policy algorithms, such as A3C (Mnih et al., 2016)). (3) M-CURL surpasses CURL by a large margin on 14 out of 16 environments from DMControl suite and 21 out of 26 environments from Atari 2600 Games, which is the previously state-of-the-art algorithm leveraging instance discrimination only.

The remaining parts are organized as follows: Several related work and background are introduced in Section 2. We introduce our method in detail in Section 3, and present some experiment results and analyses in Section 4 and Section 5. Last, we conclude our work and propose some future works in Section 6.

2 Background and Related work

In this section, we introduce the background and related work of contrastive learning and sample-efficient RL.

Contrastive Learning: Contrastive learning is a self-supervised algorithm for representation learning, which does not require additional labeling on a dataset. Contrastive learning can be regarded as an instance classification problem, where one instance should be distinguished from others. Taking image processing as an example, denote $\mathcal{I} = \{I_1, I_2, \cdots, I_N\}$ as a collection of N images. f_q and f_k are CNN encoders, both of which can map images into hidden representations. To use contrastive learning, we first use f_k to build a look-up dictionary $\mathcal{K} = \{f_k(I_i)|I_i \in \mathcal{I}\}$. Then, given an query image $I_i, i \in [N]$, we use the query encoder f_q to encode I_i and get $q_i = f_q(I_i)$. q_i should be similar to k_i (i.e., the matching key in the dictionary) while dissimilar to those in $\mathcal{K}\backslash\{k_i\}$. Oord et al. (2018) proposed InfoNCE, which is an effective contrastive loss function. Mathematically,

$$\mathcal{L}_{\text{infoNCE}}(q_i, \mathcal{K}) = -\log \frac{\exp(q_i \cdot k_i/\tau)}{\sum_{k \in \mathcal{K}} \exp(q_i \cdot k/\tau)}, \tag{1}$$

where τ is a hyperparamter. f_q and f_k can share parameters (Wu et al., 2018) or not (He et al., 2020). Contrastive learning has achieved great success in image processing (Wu et al., 2018; Hénaff et al., 2019; He et al., 2020; Tian et al., 2019), and has been extended to graph neural networks (Sun et al., 2019; Hassani & Khasahmadi, 2020), speech translation (Khurana et al., 2020), etc. Specifically,

Srinivas et al. (2020) introduced contrastive loss into image based RL problems to enhance the image encoder, which has proven to be helpful for data-efficiency on DMControl and Atari benchmark.

Sample-efficient RL Sample-efficiency is one of the long-standing challenges in real-world applications like robotics and control. While learning a policy network from high-dimensional inputs with limited samples is much challenging, some works have taken meaningful steps (Hafner et al., 2019b;a; Srinivas et al., 2020; Laskin et al., 2020; Kostrikov et al., 2020). To effectively leverage the data, introducing auxiliary tasks to enhance the representations is an active topic in RL. Shelhamer et al. (2016); Yarats et al. (2019) use generative modeling to reconstruct the original input in pixel space according to the hidden representations. Shelhamer et al. (2016); Oord et al. (2018); Lee et al. (2019) propose to predict the future frames based on the representations of past observations and actions. CURL (Srinivas et al., 2020) first propose to leverage contrastive learning on individual frames to get one better encoder. Recently, some works introduce data augmentation into RL, and the policy should be invariant to different views (Laskin et al., 2020; Kostrikov et al., 2020). Our work is complementary to them and we will make a combination in the future. Another line in sample-efficiency RL is to sample rollouts and plan (Hafner et al., 2019b;a; Kaiser et al., 2019) according to a learnable world model. Later, we will discuss the relation of our method with these works.

3 Our Method

In this work, we focus on image based RL problems. We briefly review the preliminaries of this problem in Section 3.1, then introduce our method in detail in Section 3.2 and provide some discussions in Section 3.3.

3.1 Preliminaries

The image based RL tasks can be formulated as a partially observed Markov decision process (Kostrikov et al., 2020), which is described by the tuple $(\mathcal{O}, \mathcal{A}, p, r, \gamma)$, with each element defined as follows: (1) $\mathcal{O}$ represents observations, a collection of images rendered from the environment. (2) $\mathcal{A}$ is the action space. Let $o_t \in \mathcal{O}$ and $a_t \in \mathcal{A}$ denote the rendered image and action at time step t. (3) p denotes the transition dynamics, which represents the probability distribution over the next observation o'_t given $o_{\leq t}$ and a_t. Mathematically, $p = \Pr(o'_t \mid o_{\leq t}, a_t)$. (4) $r : \mathcal{O} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function that maps the observation $o_{\leq t}$ and action a_t to a reward value, i.e., $r_t = r(o_{\leq t}, a_t) \in \mathbb{R}$. (5) $\gamma \in (0, 1]$ is the discount factor to trade off immediate and future rewards.

In deep RL, we use neural networks to process the observations and make decisions. Following common practice (Mnih et al., 2013), we stack $K(> 1)$ consecutive observations as the input to capture more information. That is, $s_t = (o_{t-K+1}, o_{t-k+2}, \cdots, o_t)$, and let $\mathcal{S}$ denote the collection of all s_t's. Let f_θ denote the CNN encoder parameterized by θ, which maps $s \in \mathcal{S}$ into a d-dimension representation. After that, the encoded representation will be fed into a policy network π_ω parameterized by ω to get the action at time step t. That is, $a_t = \pi(f_\theta(s_t))$, $a_t \in \mathcal{A}$. The objective is to maximize the cumulative discounted return, which is defined as $\mathbb{E}_\pi \sum_{t=1}^{\infty} \gamma^t r_t$. The π_ω can be specialized as any model-free algorithm, like Soft Actor Critic (SAC) (Haarnoja et al., 2018), Rainbow DQN (Mnih et al., 2013), etc.

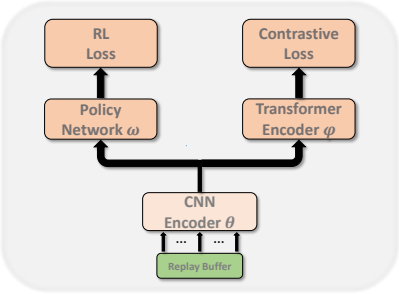


Figure 1: The training framework of M-CURL. The left part is the vanilla RL algorithm, such as SAC or DQN. The right part is a Transformer module to reconstruct the masked input.

Replay buffer $\mathcal{B}$: To stabilize the training, Mnih et al. (2013) introduce a replay buffer $\mathcal{B}$ to deep Q-learning algorithms. At time step t, after observing s_t, we take action a_t, get reward r_t and get a new observation s_{t+1}. We append the tuple $(s_t, a_t, s_{t+1}, r_t, d_t)$ to $\mathcal{B}$, where d_t indicates whether the episode terminates. The buffer has a maximum length, after which we will remove the oldest tuple from

$\mathcal{B}$. In Mnih et al. (2013), the parameters of the Q-value networks are updated by minimizing $\sum_{j=1}^{B}(r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta_Q^{'}) - Q(s_j, a_j; \theta_Q))^2$, where θ_Q is the learnable parameter, and we need to randomly sample B trajectories from $\mathcal{B}$. In CURL, the negative samples used to learn the CNN encoder are also randomly selected from $\mathcal{B}$.

3.2 ALGORITHM

Inspired by the pre-training in natural language processing (Devlin et al., 2018; Lample & Conneau, 2019), we leverage the masked pre-training technique and the Transformer architecture here. The overall framework is shown in Figure 1. A stack of K images $s_t \in \mathcal{S}$ is first processed by $f_\theta : \mathcal{S} \mapsto \mathbb{R}^d$ to get a d-dimension representation. After that, there are two parts: the left part is a policy network $\pi_\omega : \mathbb{R}^d \mapsto \mathcal{A}$, which outputs an action according to the current representation. The right part is an auxiliary model φ, which is a Transformer module and will be optimized via contrastive loss. f_θ and π_ω are introduced in Section 3.1. Next, we will introduce the auxiliary model φ in detail, including data preparation, model architecture and training objective function.

(1) **Data preparation**: We sample T consecutive observations from the replay buffer by chronological order. Denote the sampled observations as $S = (s_1, s_2, \cdots, s_T)$. To leverage the masked training technique, define $M = (M_1, M_2, \cdots, M_T)$, where for each $i \in [M]$, w.p. ϱ_m, $M_i = 1$; w.p. $1 - \varrho_m$, $M_i = 0$. $\varrho_m \in [0, 1]$ is a hyperparameter. If $M_i = 1$, following Devlin et al. (2018), s_i is modified as follows: w.p. 80%, the s_i is replaced with a zero vector; w.p. 10%, it is replaced by another s_j randomly sampled from $\mathcal{B}$; w.p. 10%, it remains unchanged. The modified observation is denoted as s_i'. Let $S' = (\bar{s}_1', \bar{s}_2', \cdots, \bar{s}_T')$ denote the refined S, where $\bar{s}_i' = s_i' M_i + s_i(1 - M_i)$. The S' is encoded by f_θ and we get $H^0 = (h_1, h_2, \cdots, h_T)$ where $h_i = f_\theta(\bar{s}_i')$, $\bar{s}_i' \in S'$.

(2) **Architectures of φ**: φ is used to reconstruct the masked input by leveraging the global input. The architecture of the Transformer encoder is leveraged. Specifically, φ consists of L identical blocks. Each block is stacked by two layers, including a self-attention layer $\mathrm{attn}(\cdots)$ and a feed-forward layer $\mathrm{ffn}(\cdots)$. Let h_i^l denote the output of i-th position of the l-th block, and $H^l = (h_1^l, h_2^l, \cdots, h_T^l)$. For any $l \in [L]$, we have that

$$\begin{aligned} \tilde{h}_i^l &= \mathrm{attn}(h_i^{l-1}, H^{l-1}) = \sum_{i=1}^{T} \alpha_i W_v h_i^{l-1}, \\ \alpha_i &= \exp\left((W_q q)^T (W_k k_i) \right) / Z, \\ Z &= \sum_{i=1}^{T} \exp\left((W_q q)^T (W_k k_i) \right). \end{aligned} \tag{2}$$

Then $\tilde{h}_i^l$ is transformed via ffn and eventually we get

$$h_i^l = \mathrm{ffn}(\tilde{h}_i^l) = W_2 \max\left(W_1 \tilde{h}_i^l + b_1, 0 \right) + b_2. \tag{3}$$

In equation 2 and equation 3, the W's and b's are the parameters to be learned. By enumerating l from 1 to L and alternatively using equation 2 and equation 3, we will eventually obtain H^L. Note that positional embedding, layer normalization (Ba et al., 2016) and residual connections are all used. We leave the details in Appendix A.1.4.

(3) **Training strategy**: Following Srinivas et al. (2020), we use momentum contrastive learning to optimize φ and f_θ. The workflow of our training strategy is in Figure 2. The query set Q is the H^L obtained in the previous step. The key set $K = (k_1, k_2, \cdots, K_T)$ is encoded from the non-masked set S: $k_i = f_{\theta_k}(s_i)$, $s_i \in S$. The architectures of query encoder f_θ and key encoder f_{θ_k} are the same but the parameters are different, where θ_k is iteratively updated as $\theta_k = m\mathtt{SG}(\theta) + (1-m)\theta_k$, m is the momentum value and $\mathtt{SG}$ means "stop gradient" following He et al. (2020).

The contrastive loss is defined as follows:

$$\mathcal{L}_{\mathrm{ct}} = \sum_{i=1}^{T} -M_i \log \frac{\exp(q_i \cdot k_i / \tau)}{\sum_{j=1}^{T} \exp(q_i \cdot k_j / \tau)}, \tag{4}$$

where τ is a hyperparameter. The intuition behind equation 4 is that for any masked position, the reconstructed h_i^L (i.e., q_i) should be similar to its original feature k_i, and dissimilar to the others. In

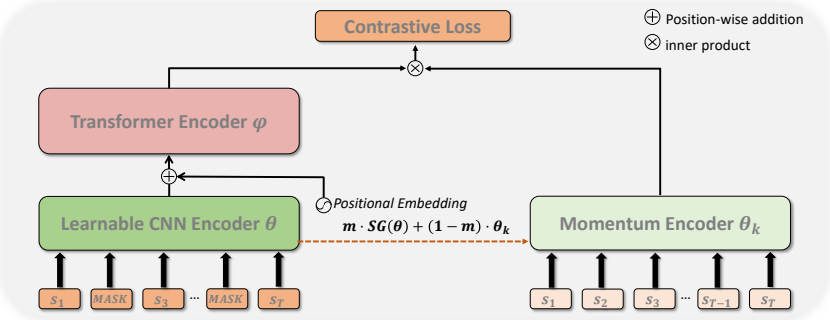


Figure 2: Training strategy of M-CURL.

this way, the H^0 output by f_θ should be relevant such that when we mask specific features in H^0, we can still reconstruct them by using the context features.

For the RL training objective function, we can use any one for off-policy RL algorithms. Generally, the loss for RL in our paper can be summarized as

$$\begin{aligned}\mathcal{L}_{\text{rl}} =& \mathbb{E}_{(s_t,a_t)\sim\mathcal{B}}\Bigg[\left(Q_{\omega_1}\left(f_\theta(s_t),a_t\right)-\hat{Q}\left(f_{\theta'}(s_t),a_t\right)\right)^2 \\ &+\alpha \mathrm{D}_{\mathrm{KL}}\left(\pi_{\omega_2}\left(\cdot \mid f_\theta(s_t)\right) \| \frac{\exp\left(Q_{\omega_1}\left(f_\theta(s_t),\cdot\right)\right)}{Z_\theta\left(s_t\right)}\right)\Bigg].\end{aligned} \tag{5}$$

where (s_t, a_t) is randomly sampled from $\mathcal{B}$, ω_1 and ω_2 are the parameters of the Q-function and the policy network, $\hat{Q}$ is the target value, $\mathrm{D}_{\mathrm{KL}}(\cdot\|\cdot)$ is KL-divergence between the policy function and Q-function (Haarnoja et al., 2018) and $Z_\theta\left(s_t\right)$ is the normalization term.

The overall objective function of our approach is

$$\min_{\theta,\omega,\varphi} \mathcal{L}_{\text{all}} = \mathcal{L}_{\text{rl}} + \lambda \mathcal{L}_{\text{ct}}, \tag{6}$$

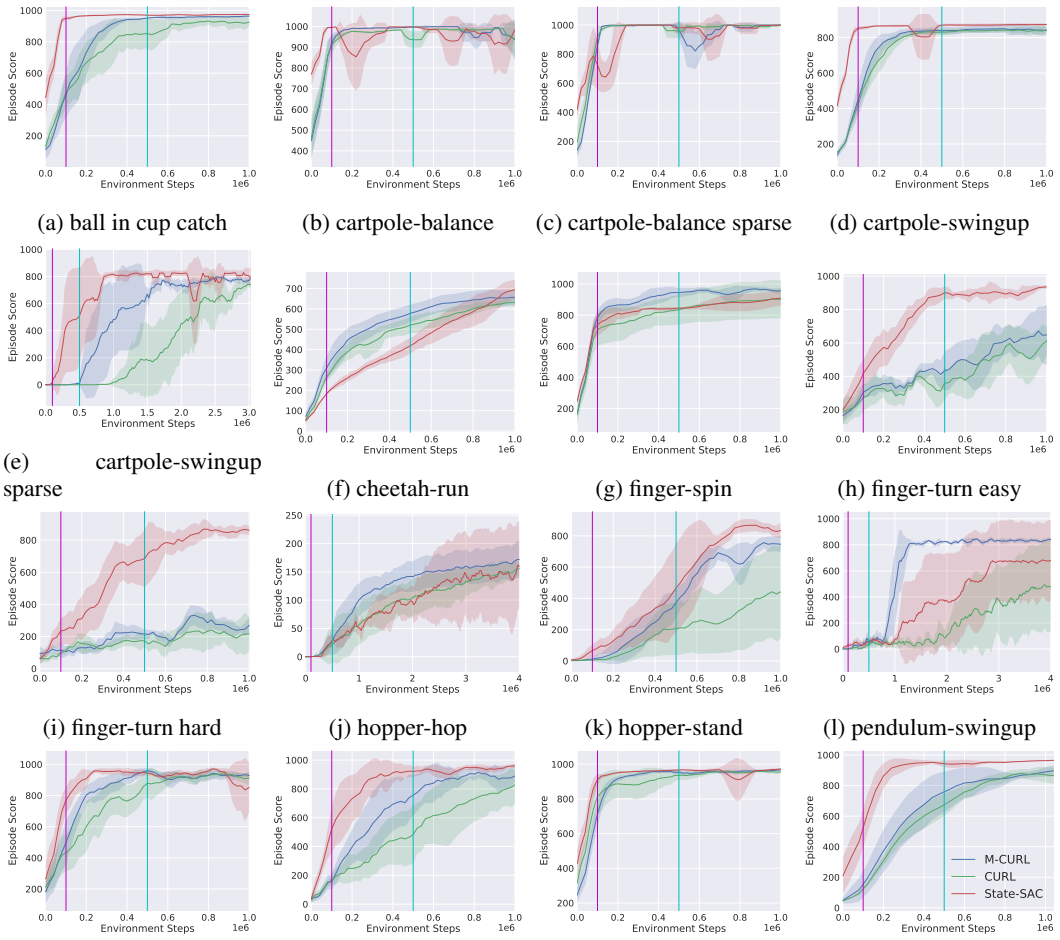
where θ is related to both terms in equation 6. Following Srinivas et al. (2020), we fix λ as 1. A detailed example of M-CURL coupling with SAC algorithm is described in Appendix C.

3.3 Discussion

Methods to improve sample efficiency in pixel-based RL can be divided into two categories, one is to sample rollouts and plan through a world model (Hafner et al., 2019b;a; Kaiser et al., 2019), and the other is to introduce auxiliary losses as discussed in previous section. While showing improved sample efficiency, the former class usually have difficulty on balancing various auxiliary loss, such as the dynamic loss, reconstruction loss, KL regularization term, reward prediction loss in addition to value function and policy function optimization, and are correspondingly brittle to hyper-parameters tuning and hard to reproduce. In contrast, in our method, we only introduce an additional loss and it is still widely effective to fix the balance coefficient as 1 across all our experiments. Our method belongs to the latter class, and has many advancements on it. First, different from reconstruction-based methods (Shelhamer et al., 2016; Yarats et al., 2019), which always optimize on a generative object on single observation, our method try to use one discriminative loss. Second, another related work is to use latent variable models to model the temporal structure in the MDPs (POMDPs) and get a compact and disentangled representations (Oord et al., 2018; Lee et al., 2019). However, these methods are all use the LSTMs (Hochreiter & Schmidhuber, 1997) as their basic arhcitecture which limits their method to be unidirectional. In our method, we use Transformer to model the context relation, and to the best of our knowledge, our method is the first one to incorporate Transformer into RL to model this bidirectional characteristic.

4 Experiments

In this section, we evaluate our method on two commonly used benchmarks, DMControl suite (Tassa et al., 2020) and Atari 2600 Games (Bellemare et al., 2013).



(a) ball in cup catch (b) cartpole-balance (c) cartpole-balance sparse (d) cartpole-swingup

(e) cartpole-swingup sparse (f) cheetah-run (g) finger-spin (h) finger-turn easy

(i) finger-turn hard (j) hopper-hop (k) hopper-stand (l) pendulum-swingup

(m) reacher-easy (n) reacher-hard (o) walker-stand (p) walker-walk

Figure 3: Results of CURL and our method on DMControl benchmark. Our method outperforms CURL on 14 out of 16 selected environments. The shaded region represents the standard deviation of the average return over 5 trials, and curves are smoothed uniformly for visual clarity.

4.1 Environments

(1) *DMcontrol suite* contains a set of continuous control tasks powered by the MuJoCo physics engine (Todorov et al., 2012). These tasks are rendered through raw pixels which is suitable to test data-efficiency of different methods. Following the setup of CURL (Srinivas et al., 2020), we evaluate the performance at $100k$ (low sample regime) and $500k$ (asymptotically optimal regime) environment steps. We run experiments on sixteen environments from DMControl to evaluate the performance. Following common practice, we randomly crop the images rendered by game simulator from 100×100 to 84×84 for data augmentation. Action repeat, which refers to the number of times an action is repeated when it's drawn from the agent's policy, is set to 4 by default except that 2 for Finger Spin and Walker Walk, 8 for Cartpole Swingup. For each state, we stack the latest 3 observations as input, i.e., the K in Section 3.1 is 3.

(2) For discrete control tasks, we evaluate our method on *Atari 2600 games* from the arcade learning environment (Bellemare et al., 2013). We follow the training and evaluation procedures of Srinivas et al. (2020)[1], to compare different algorithms in term of performance within 100k interaction steps (i.e., 400k environment step). We choose Rainbow (Hessel et al., 2018) algorithm to train the policy network π_ω. We run experiments on 26 atari games. Following common practice, the action repeat is set to 4 and frame stack is 4 across all games.

[1] `https://github.com/aravindsrinivas/curl_rainbow`

GAME	HUMAN	SIMPLE	OTRAINBOW	EFF. RAINBOW	CURL	M-CURL
ALIEN	7127.7	616.9	824.7	739.9	558.2	**847.4**
AMIDAR	1719.5	88.0	82.8	**188.6**	142.1	168.0
ASSAULT	742.0	527.2	351.9	431.2	600.6	**606.8**
ASTERIX	8503.3	**1128.3**	628.5	470.8	734.5	629.0
BANK HEIST	753.1	34.2	**182.1**	51.0	131.6	176.6
BATTLE ZONE	37187.5	5184.4	4060.6	10124.6	14870.0	**21820.0**
BOXING	12.1	**9.1**	2.5	0.2	1.2	3.5
BREAKOUT	30.5	**16.4**	9.84	1.9	4.9	3.5
CHOPPER COMMAND	7387.8	**1246.9**	1033.33	861.8	1058.5	1096.0
CRAZY_CLIMBER	35829.4	**62583.6**	21327.8	16185.3	12146.5	19420.0
DEMON_ATTACK	1971.0	208.1	711.8	508.0	817.6	**860.8**
FREEWAY	29.6	20.3	25.0	27.9	26.7	**28.6**
FROSTBITE	4334.7	254.7	231.6	866.8	1181.3	**2154.8**
GOPHER	2412.5	771.0	**778.0**	349.5	669.3	426.4
HERO	30826.4	2656.6	6458.8	6857.0	6279.3	**6884.9**
JAMESBOND	302.8	125.3	112.3	301.6	**471.0**	368.0
KANGAROO	3035.0	323.1	605.4	779.3	872.5	**1410.0**
KRULL	2665.5	**4539.9**	3277.9	2851.5	4229.6	2928.9
KUNG_FU_MASTER	22736.3	**17257.2**	5722.2	14346.1	14307.8	15804.0
MS_PACMAN	6951.6	1480.0	941.9	1204.1	1465.5	**1687.2**
PONG	14.6	**12.8**	1.3	-19.3	-16.5	-16.8
PRIVATE EYE	69571.3	58.3	100.0	97.8	218.4	**293.4**
QBERT	13455.0	**1288.8**	509.3	1152.9	1042.4	1272.5
ROAD_RUNNER	7845.0	5640.6	2696.7	**9600.0**	5661.0	8890.0
SEAQUEST	42054.7	**683.3**	286.92	354.1	384.5	444.4
UP_N_DOWN	11693.2	3350.3	2847.6	2877.4	2955.2	**3475.6**
Median HNS	100.0%	14.4%	20.4%	16.1%	17.5%	**24.2%**

Table 1: Results of our method and baselines on 26 environments from Atari 2600 Games at 100k timesteps. Our method achieves state-of-the-art performance on 10 out of 26 environments. Our method is implemented on top of Eff. Rainbow (van Hasselt et al., 2019), improves its performance on 24 out of 26 environments, and outperforms CURL on 21 out of 26 environments. The scores are averaged over 5 different seeds, and we also can see our method achieves superhuman performance on JamesBond, Krull and Road_Runner.

4.2 BASELINES

For DMControl Suite, CURL (Srinivas et al., 2020) has demonstrated its superiority over previous methods designed for sample efficiency, such as PlaNet (Hafner et al., 2019b), Dreamer (Hafner et al., 2019a), SAC+AE (Yarats et al., 2019), SLACv1 (Lee et al., 2019). Due to computation limitation, we compare our method with CURL and state-SAC. With state-SAC, the agents can access to the low level states of the 16 selected DMControl environments, such as positions and velocities. State-SAC is usually implemented as a baseline for the algorithms about improving data efficiency, which can be used to measure the gap between the proposed method and the approximate upper-bound scores which pixel-based agents can achieve. Each algorithm is run with five different seeds.

For Atari 2600 Games, we compare our method with the following baselines: (i) Human performance (Kaiser et al., 2019; van Hasselt et al., 2019); (ii) SimPLe (Kaiser et al., 2019), the best model-based algorithm for Atari; (iii) OTRainbow (Kielak, 2020), the over-trained version of Rainbow, (iv) Efficient Rainbow (van Hasselt et al., 2019), which modifies the the hyper-parameters and architecture of Rainbow for data efficiency, (v) CURL (Srinivas et al., 2020), which first integrates contrastive learning in Rainbow for data efficiency. Following Srinivas et al. (2020), our method is implemented on top of Efficient Rainbow.

4.3 MODEL CONFIGURATION

The architecture of the CNN encoder f_θ is the same as that of CURL (Srinivas et al., 2020) for a fair comparison. The momentum coefficient is 0.05 for DMcontrol and 0.001 for Atari game. We use one 2-layer Transformer encoder with single-head attention for DMControl and Atari benchmark. The masked percentage in data preparation ϱ_m is 0.6 for Finger Spin and Walker Walk, 0.5 for

others. The other hyperparameters remain the same as (Srinivas et al., 2020)[2]. The two objects $\mathcal{L}_{rl}$ and $\mathcal{L}_{cl}$ are weighted equally, and updated jointly per interaction step. We use Adam (Kingma & Ba, 2014) optimizer for training. The detailed parameters are left in Appendix A.

4.4 RESULTS

The results of DMControl are shown in Figure 3. Generally, our method outperforms the standard CURL on 14 out of 16 experiments, and is slightly worse than CURL on the remaining two (Cartpole-Balance-Sparse and Walker-Stand), which demonstrates the effectiveness of our method. We have the following observations:

(1) From the perspective of sample efficiency, on most of these tasks, the slopes of the curves of our method are much larger than those of CURL. That is, with the same amount of interaction with environments, our method can achieve better performance. As shown in the figures, when the interaction steps are $100k$ (magenta vertical line) and $500k$ (cyan vertical line), our method achieves significantly better results than CURL. On some difficult tasks like Cartpole-Swingup-Sparse and Pendulum Swingup, our proposed M-CURL requires much fewer steps than CURL to get positive rewards. These results demonstrate that our method is more sample efficient.

(2) Our method further reduces the gap between CURL and state-SAC, which is trained from internal state rather than from pixel. State-SAC is expected to be an approximate upper bound of algorithms for improving data efficiency. We can even observe that in Cheetah-Run, Finger-Spin, Hopper-Hop and Pendulum-Swingup tasks, our method achieves higher scores than state-SAC, which again shows the effectiveness of our method.

We also conduct experiments on Atari with $100k$ interaction steps. The results are shown in Table 1. Again, our method achieves better score than CURL. Specifically,

(1) Our method M-CURL surpasses CURL on 21 out of 26 games, which shows that M-CURL can bring improvements to discrete control problems. Our method outperforms Efficient Rainbow DQN on 24 out of 26 Atari games, which is the algorithm that M-CURL built on top of. This shows the generality of our method. Besides, M-CURL achieves super-human performances on JamesBond, Krull and Road_Runner by 24%, 25% and 13% improvements, which shows the effectiveness of our method.

(2) We also provide the median human-normalized score (HNS) of all algorithms, which is calculated as follows: Given M games, we first get the human scores and the scores of a random algorithm, denoted as $S_{H,i}$ and $S_{R,i}$ respectively, $i \in [M]$. For any algorithm A, the score of the i-th game is $S_{A,i}$. Medium HNS is the medium value of $\{\frac{S_{A,i}-S_{R,i}}{S_{H,i}-S_{R,i}} | i \in [M]\}$. The median HNS of M-CURL is 24.2%, while the meadian HNS of CURL, SimPLe and Efficient Rainbow DQN are 17.5%, 14.4% and 16.1% respectively, which shows that in general, our method achieves the best performance across different tasks.

5 ABLATION STUDY

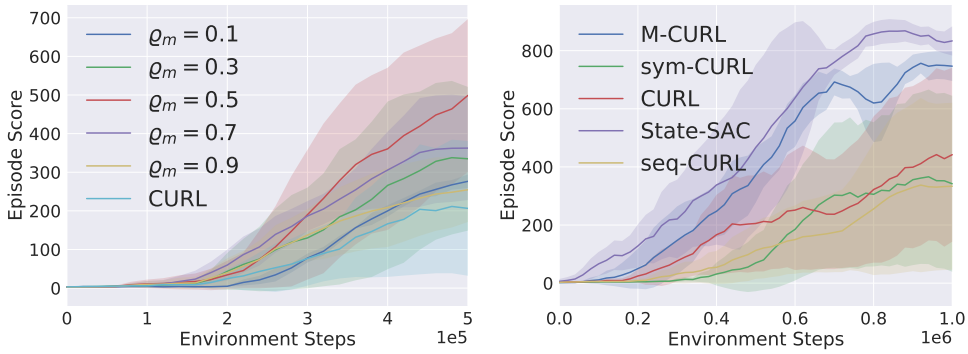
To better understand our framework, we provide several further study experiments and analyses.

5.1 STUDY ON MASKED PROBABILITY

We explore the effect of different masked probability ϱ_m in this section. We conduct experiments on Hopper-Stand environment with $\varrho_m \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The results are shown in Figure 4a. Our method achieves better results than standard CURL w.r.t. different ϱ_m, which demonstrates the effectiveness of using masked training.

As shown in Figure 4a, sample-efficiency indeed varies w.r.t. ϱ_m. (1) With $500k$ interactions, setting ϱ_m as 0.5 achieves the best performance. (2) Setting a too large ϱ_m (i.e., 0.9) will not lead to good results, since too much information is masked and we cannot reconstruct a reasonable feature. Also, setting a too small ϱ_m (i.e., 0.1) will hurt the performance too, because there are fewer positions to

[2] https://github.com/MishaLaskin/curl



(a) Scores on Hopper-Stand environment achieved by different masked probability ϱ_m.

(b) Comparison with different variants of CURL.

Figure 4: Results of ablation study.

provide the training signal (see equation 4), and leveraging more information will reduce the ability of remembering past information and predicting future information.

5.2 Study on Variants of CURL

(1) **CURL with consecutive inputs**

With our method, the input of the Transformer module is consecutive T observations from $\mathcal{S}$, which is then processed into query, key and negative samples for contrastive learning. However, in standard CURL, these T frames are randomly generated to simulate an i.i.d. data generation process (Mnih et al., 2013; Haarnoja et al., 2018). To investigate whether the improvements of our method are from the usage of consecutive inputs only, we implement a variant of CURL (denoted as seq-CURL), where the inputs are not randomly generated from replay buffer, but sequentially selected from the buffer (see Appendix D for pseudo-code). The results are in Figure 4b. Compared to standard CURL, the results become even worse. This shows the importance of using randomly sampling in CURL, and demonstrates that the improvement of our method is not simply due to the usage of consecutive inputs.

(2) **M-CURL with two Transformer modules**

In our method, the queries are encoded by a Transformer module while the keys are not. A natural baseline is that the keys should also be encoded using a Transformer module. We implement this variant (denoted as sym-CURL), where the Transformer module of keys are updated via momentum contrastive learning. The results are in Figure 4b. We can see that using a symmetric architecture hurts the performance too and the absence of Transformer module above the momentum encoder is crucial in our method. Our conjecture is that in M-CURL, the contrastive loss is directly applied to the features output by the CNN encoder, which is the inputs of policy network, and thus force it capture the correlation between consecutive observations like the outputs of Transformer.

Additional ablation studies are left in Appendix B due to space limitations, including the study on the consecutive length T and the study of Transformer module with different numbers of layers.

6 Conclusion and Future Work

In this work, we proposed M-CURL, which leverages masked training and Transformer module to improve sample efficiency for reinforcement learning. Experiments on DMControl and Atari benchmark demonstrate the effectiveness of our method and show the benefit of utilizing the correlation among the adjacent video frames. For future work, there are many directions to explore.

First, how to efficiently adapt our idea into on-policy RL algorithm is an interesting topic. Second, how to leverage both the states and actions (rather than only states in this work) saved in replay buffer into our method remains to be explored. Third, we will study how to use more advanced models and pre-training techniques to improve sample-efficiency for RL.

REFERENCES

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

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

Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4243–4250. IEEE, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

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

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

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. *arXiv preprint arXiv:2006.05582*, 2020.

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

Olivier J Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

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

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. *arXiv preprint arXiv:1903.00374*, 2019.

Sameer Khurana, Antoine Laurent, and James Glass. Cstnet: Contrastive speech translation network for self-supervised speech representation learning. *arXiv preprint arXiv:2006.02814*, 2020.

Kacper Kielak. Do recent advancements in model-based deep reinforcement learning really improve data efficiency? *arXiv preprint arXiv:2003.10181*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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.

Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

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

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

Xihan Li, Jia Zhang, Jiang Bian, Yunhai Tong, and Tie-Yan Liu. A cooperative multi-agent reinforcement learning framework for resource balancing in complex logistics network. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 980–988, 2019.

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

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

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

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.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Zhirong Wu, Yuanjun Xiong, Stella Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *arXiv preprint arXiv:1805.01978*, 2018.

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.

A More Details about Experiment Setup

A.1 Network architecture

A.1.1 Encoder network

For DMControl, we use the encoder architecture from Yarats et al. (2019), which is the same as that in Srinivas et al. (2020). The encoder is stacked by four convolutional layers with 3×3 kernel size and 32 channels, and one fully-connected layer which outputs 50-dimension features. Each convolutional layer is followed by a `ReLU` activation and the output of fully-connected layer is normalized by `LayerNorm` (Ba et al., 2016). For Atari, we use the data-efficient version of Rainbow DQN (van Hasselt et al., 2019), and the encoder is stacked by two convolutional layers with 5×5 kernel size and 32, 64 channels respectively. These two convolutional layers are connected by `ReLU` activation.

A.1.2 Actor and Critic networks

For DMcontrol, we use the SAC (Haarnoja et al., 2018) algorithm, and there are a critic network and an actor network. The critic network consists of 3 fully connected layers. Each fully-connected layer (except the last one) is followed by a `ReLU` activation. The actor network also consists of 3 fully connected layers, which output the mean values and covariance for one diagonal Gaussian distribution to reparameterize the policy. For Atari, we use the efficient Rainbow DQN (van Hasselt et al., 2019), and there is only one Q-network consisting of a value branch and an advantage branch. Each branch is 2-layers MLP which intermediated by the `ReLU` activation, and we use Noisy Network (Fortunato et al., 2017) for exploration.

A.1.3 Positional Embedding

The outputs of the image encoder f_θ do not contain positional information, which cannot model the temporal information among different inputs. To overcome this difficulty, following Vaswani et al. (2017), we introduce positional embedding into the Transformer module, which is a d-dimensional vector and not to be learned. For each position k, denote the positional embedding as $p_k \in \mathbb{R}^d$. Note that in Section 3.2 (line 233), we also denote the image representation as a d-dimensional vector. Let $p_{(k,j)}$ denote the j-th element of p_k, $j \in [d]$. We have that

$$\begin{aligned} p_{(k,2i)} &= \sin\left(k/10000^{2i/d}\right), \\ p_{(k,2i+1)} &= \cos\left(k/10000^{2i/d}\right), \end{aligned} \tag{7}$$

where $2i$ and $2i+1$ are the dimension indicators. The augmented query representation of position k becomes

$$h_k = f_\theta(\vec{s}'_k) + p_k. \tag{8}$$

A.1.4 TRANSFORMER ENCODER BLOCK

We then process these query representations (obtained by using positional embedding) with several stacked Transformer encoder blocks. Each block consists of two layers, namely a self-attention layer $\mathrm{attn}(\cdots)$ and a feed-forward layer $\mathrm{ffn}(\cdots)$. Specifically, let h_i^l denote the output of i-th position of the l-th block, and $H^l = (h_1^l, h_2^l, \cdots, h_T^l)$. For any $l \in [L]$, we have that

$$
\begin{aligned}
\alpha_{ij} &= \frac{\exp\left(\left(W_q h_i^{l-1}\right)^T \left(W_k h_j^{l-1}\right)\right)}{Z}, \\
Z &= \sum_{j=1}^{|K|} \exp\left(\left(W_q h_i^{l-1}\right)^T \left(W_k h_j^{l-1}\right)\right), \\
\hat{h_i^l} &= \mathrm{attn}(h_i^{l-1}, H^{l-1}, H^{l-1}) = \sum_{j=1}^{|V|} \alpha_{ij} W_v h_j^{l-1}, \\
\tilde{h}_i^l &= \mathrm{LayerNorm}(h_i^{l-1} + \hat{h_i^l}), \\
\bar{h}_i^l &= \mathrm{ffn}(\tilde{h}^l) = W_2 \max\left(W_1 \tilde{h}_i^l + b_1, 0\right) + b_2, \\
h_i^l &= \mathrm{LayerNorm}(\tilde{h}_i^l + \bar{h}_i^l),
\end{aligned}
\tag{9}
$$

where the W_q, W_k, W_v, W_1, W_2 are learnable parameters. After L encoder block, we obtain H^L as the query set Q for contrastive learning.

A.2 LEARNING RATE SCHEDULER

Specifically, following Vaswani et al. (2017), the learning rate scheduler of the Transformer module is defined as follows:

$$
\eta = \eta_0 \times \min\left(\left(\frac{\mathrm{step}}{\mathrm{step}_w}\right)^{-0.5}, \frac{\mathrm{step}}{\mathrm{step}_w}\right), \tag{10}
$$

where η_0 is the base learning rate which is the same as that used in optimizing RL obeject, step is the current training step and step_w is the warmup step. We set step_w as 6000 in our experiments.

A.3 OTHER HYPER-PARAMETERS

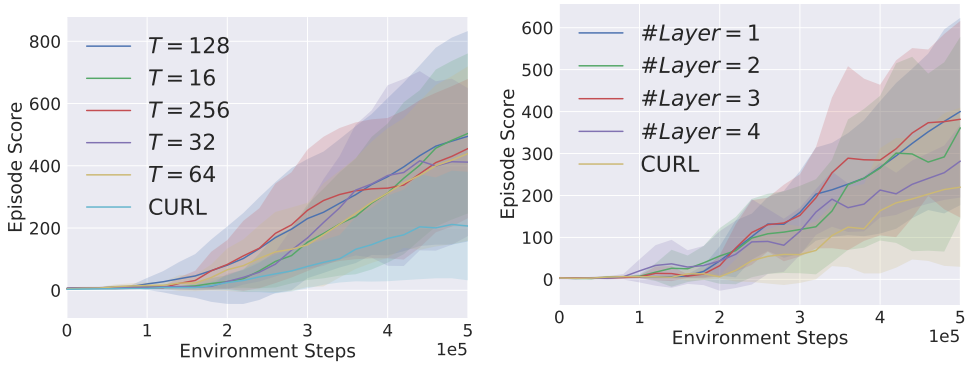
We mainly follow the setup used in Srinivas et al. (2020), and the other hyper-parameters are summarized below. For DMControl benchmark, the replay buffer size is $100k$, and the batchsize is 512 for cheetah and 128 for others. We use Adam (Kingma & Ba, 2014) to optimize all our parameters with $\beta_1 = 0.9, \beta_2 = 0.999$ for all our network parameters and $\beta_1 = 0.5, \beta_2 = 0.999$ for the temperature coefficient in SAC. The learning rate is $1e-4$ for all optimizer and we use `inverse_sqrt` learning rate scheduler described in A.2 for Transformer module. For Atari benchmark, the batchsize is 32 and the multi-step return length is 20. The optimizer settings and learning rates are the same as those used in DMControl.

B MORE ABLATION STUDY

B.1 STUDY ON SEQUENCE LENGTH

The role of Transformer module we used in our method is to capture temporal correlation in state sequences. The sequence length T is also another key factor for learning the representations. In this section, to investigate the effect of different sequence lengths, we conduct experiments on Hopper-Stand environment with different sequence length, $T \in \{16, 32, 64, 128, 256\}$, where we change the corresponding batch size B to ensure the same number of training samples $T \times B$ in each parameter update. The results are shown in Figure 5a.

As shown in the figure, we find that there is no much difference between the scores achieved by different sequence lengths, which shows the our method is not sensitive to the choice of T. The



(a) Scores on Hopper-Stand environment achieved by different sequence length.

(b) Scores on Hopper-Stand environment achieved by different layer number.

Figure 5: Results of ablation study.

result of length 16 is competitive with other results of longer lengths, which means that the effective context to reconstruct one state representation in RL settings is much smaller than that in NLP problems which is always hundreds of tokens.

B.2 Study on the Layer Number

We also vary the number of layers of the Transformer module in $\{1, 2, 3, 4\}$. The result is shown in Figure 5b. We notice that when the layer numbers are set as $\{1, 2, 3\}$, there is not significant difference. When the number of layer is increased to four, the performance slightly drops. We can also observe that the all the four settings outperform the standard CURL baseline, which demonstrates the effectiveness of our method.

C Algorithm

Algorithm 1 Masked Contrastive Representation Learning for Reinforcement Learning (M-CURL) coupling with SAC

Require: Environment E, initial parameters θ, ϕ, ω_1, ω_{21}, ω_{22} for the convoluational encoder, Transformer module, actor network π_{ω_1}, and two critics $Q_{\omega_{21}}, Q_{\omega_{22}}$, and an empty replay buffer $\mathcal{B}$.

$\bar{\theta} \leftarrow \theta$
$\bar{\omega}_{2i} \leftarrow \omega_{2i}$ for $i \in \{1, 2\}$
$s_1 \sim E_{\text{reset}}()$
for each iteration t **do**
  for each environment step **do**
    $a_t \sim \pi_{\omega_1}(\cdot|s_t)$
    $s_{t+1}, r_t, d_t \sim E$
    $\mathcal{B} \leftarrow \mathcal{B} \cup (s_t, a_t, s_{t+1}, r_t, d_t)$
  end for
  for each gradient step **do**
    $x_{1:T+1}, a_{1:T}, r_{1:T} \sim \mathcal{B}$ ▷ i.i.d samples
    $\omega_{2i} \leftarrow \omega_{2i} - \lambda_Q \nabla_{\omega_{2i}} J_Q(x_{1:T+1}, a_{1:T}, r_{1:T}; \omega_{2i})$, for $i \in \{1, 2\}$
    $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta J_Q(x_{1:T+1}, a_{1:T}, r_{1:T}; \theta)$, for $i \in \{1, 2\}$
    $\omega_1 \leftarrow \omega_1 - \lambda_\pi \nabla_{\omega_1} J_\pi(x_{1:T+1}, a_{1:T}, r_{1:T}; \omega_1)$
    $x_{1:T} \sim \mathcal{B}$ ▷ consecutive samples
    $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi J_{cl}(x_{1:T;\phi})$
    $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta J_{cl}(x_{1:T;\theta})$
    $\bar{\theta} \leftarrow \tau_1 \theta + (1 - \tau_1)\bar{\theta}$
    $\bar{\omega}_{2i} \leftarrow \tau_2 \omega_{2i} + (1 - \tau_2)\bar{\omega}_{2i}$, for $i \in \{1, 2\}$
  end for
end for

D PSEUDO-CODE FOR SEQ-CURL

```
def sample_seq_curl(self):

    idxs = np.random.randint(
        0, self.capacity if self.full else self.idx, size=self.batch_size
    )

    obses = self.obses[idxs]
    next_obses = self.next_obses[idxs]

    obses = random_crop(obses, self.image_size)
    next_obses = random_crop(next_obses, self.image_size)

    obses = torch.as_tensor(obses, device=self.device).float()
    next_obses = torch.as_tensor(
        next_obses, device=self.device
    ).float()
    actions = torch.as_tensor(self.actions[idxs], device=self.device)
    rewards = torch.as_tensor(self.rewards[idxs], device=self.device)
    not_dones = torch.as_tensor(self.not_dones[idxs], device=self.device)

    bsz = 8
    length = 32

    idxs = np.random.randint(
        0, self.capacity - length if self.full else self.idx - length,
        size=bsz
    )

    idxs = idxs.reshape(-1, 1)
    step = np.arange(length).reshape(1, -1)
    idxs = idxs + step
    obs_x = self.obses[idxs]
    obs_z = obs_x.copy()
    obs_x = obs_x.reshape(-1, *obs_x.shape[2:])
    obs_z = obs_z.reshape(-1, *obs_z.shape[2:])
    obs_x = random_crop(obs_x, self.image_size)
    obs_z = random_crop(obs_z, self.image_size)
    obs_x = torch.as_tensor(obs_x, device=self.device).float()
    obs_z = torch.as_tensor(obs_z, device=self.device).float()


    cpc_kwargs = dict(obs_anchor=obs_x, obs_pos=obs_z,
                                time_anchor=None, time_pos=None)

    return obses, actions, rewards, next_obses, not_dones, cpc_kwargs
```