TOWARDS SIMPLICITY IN DEEP REINFORCEMENT LEARNING: STREAMLINED OFF-POLICY LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The field of Deep Reinforcement Learning (DRL) has recently seen a surge in the popularity of maximum entropy reinforcement learning algorithms. Their popularity stems from the intuitive interpretation of the maximum entropy objective and their superior sample efficiency on standard benchmarks. In this paper, we seek to understand the primary contribution of the entropy term to the performance of maximum entropy algorithms. For the Mujoco benchmark, we demonstrate that the entropy term in Soft Actor Critic (SAC) principally addresses the bounded nature of the action spaces. With this insight, we show how streamlined algorithms without entropy maximization can match the performance of SAC. We also propose a simple non-uniform sampling method for selecting transitions from the replay buffer during training. We further show that the streamlined algorithm with the simple non-uniform sampling scheme outperforms SAC and achieves state-of-the-art performance on challenging continuous control tasks.

1 INTRODUCTION

Off-policy deep Reinforcement Learning (RL) algorithms aim to improve sample efficiency by reusing past experience. Recently a number of new off-policy Deep Reinforcement Learning algorithms have been proposed for control tasks with continuous state and action spaces, including Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) (Lillicrap et al., 2015; Fujimoto et al., 2018). TD3, in particular, has been shown to be significantly more sample efficient than popular on-policy methods for a wide range of Mujoco benchmarks.

The field of Deep Reinforcement Learning (DRL) has also recently seen a surge in the popularity of maximum entropy reinforcement learning algorithms. Their popularity stems from the intuitive interpretation of the maximum entropy objective and their superior sample efficiency on standard benchmarks. In particular, Soft Actor Critic (SAC), which combines off-policy learning with maximum-entropy RL, not only has many attractive theoretical properties, but can also give superior performance on a wide-range of Mujoco environments, including on the high-dimensional environment Humanoid for which both DDPG and TD3 perform poorly (Haarnoja et al., 2018a;b; Langlois et al., 2019). The TD3 and SAC algorithms share many common features, including an actor-critic structure, off-policy learning, and target critic networks. The newest version of SAC and TD3 both take the min of the two target critic network when computing update targets to reduce overestimation bias (Fujimoto et al., 2018). A primary difference between the two approaches is that SAC employs maximum entropy reinforcement learning whereas TD3 does not. TD3 also uses delayed policy update and policy network smoothing for improved robustness.

In this paper, we first seek to understand the primary contribution of the entropy term to the performance of maximum entropy algorithms. For the Mujoco benchmark, we demonstrate that when using the standard objective without entropy along with standard additive noise exploration, there is often insufficient exploration due to the bounded nature of the action spaces. Specifically, the outputs of the policy network are often way outside the bounds of the action space, so that they need to be squashed to fit within the action space. The squashing results in actions persistently taking on their maximal values, so that there is insufficient exploration. In contrast, the entropy term in the SAC objective forces the outputs to have sensible values, so that even with squashing, exploration is maintained. We conclude that the entropy term in the objective for Soft Actor Critic principally addresses the bounded nature of the action spaces in the Mujoco environments.

With this insight, we propose Streamlined Off Policy (SOP), a streamlined algorithm using the standard objective without the entropy term. SOP employs a simple normalization scheme to address the bounded nature of the action spaces, thereby allowing for satisfactory exploration throughout training. We also consider replacing the aforementioned normalization scheme with inverting gradients (IG) Hausknecht & Stone (2015). Our experimental results show that SOP and IG match the sample-efficiency and robustness performance of SAC, including on the more challenging Ant and Humanoid environments. This demonstrates a need to revisit the importance of entropy maximization in DRL.

Keeping with the theme of simplicity with the goal of meeting Occam's principle, we also propose a simple non-uniform sampling method for selecting transitions from the replay buffer during training. In vanilla SOP (as well as in DDPG, TD3, and SAC), samples from the replay buffer are chosen uniformly at random during training. Our method, called Emphasizing Recent Experience (ERE), samples more aggressively recent experience while not neglecting past experience. Unlike Priority Experience Replay (PER) (Schaul et al., 2015), a popular non-uniform sampling scheme for the Atari environments, ERE is only a few lines of code and does not rely on any sophisticated data structures. We show that SOP combined with ERE out-performs SAC and provides state of the art performance. For example, for Ant and Humanoid, it improves over SAC by 21% and 24%, respectively, with one million samples. Furthermore, we also investigate combining SOP with PER, and show SOP+ERE also out-performs the more complicated SOP+PER scheme.

The contributions of this paper are thus threefold. First, we uncover the primary contribution of the entropy term of maximum entropy RL algorithms when the environments have bounded action spaces. Second, we propose streamlined algorithms which do not employ entropy maximization but nevertheless matches the sampling efficiency and robustness performance of SAC for the Mujoco benchmarks. And third, we combine our streamlined algorithms with a simple non-uniform sampling scheme to achieve state-of-the art performance for the Mujoco benchmark. We provide anonymized code for reproducibility [1].

2 PRELIMINARIES

We represent an environment as a Markov Decision Process (MDP) which is defined by the tuple $(\mathcal{S}, \mathcal{A}, r, p, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are continuous multi-dimensional state and action spaces, $r(s, a)$ is a bounded reward function, $p(s'|s, a)$ is a transition function, and γ is the discount factor. Let $s(t)$ and $a(t)$ respectively denote the state of the environment and the action chosen at time t. Let $\pi = \pi(a|s),\ s \in \mathcal{S}, a \in \mathcal{A}$ denote the policy. We further denote K for the dimension of the action space, and write a_k for the kth component of an action $a \in \mathcal{A}$, that is, $a = (a_1, \ldots, a_K)$.

The expected discounted return for policy π beginning in state s is given by:

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s(t), a(t))|s(0) = s] \tag{1}$$

Standard MDP and reinforcement learning problem formulations seek to maximize $V_\pi(s)$ over policies π. For finite state and action spaces, under suitable conditions for continuous state and action spaces, there exists an optimal policy that is deterministic (Puterman, 2014; Bertsekas & Tsitsiklis, 1996). In reinforcement learning with unknown environment, exploration is required to learn a suitable policy.

In DRL with continuous action spaces, typically the policy is modeled by a parameterized policy network which takes as input a state s and outputs a value $\mu(s; \theta)$, where θ represents the current parameters of the policy network (Schulman et al., 2015; 2017; Vuong et al., 2018; Lillicrap et al., 2015; Fujimoto et al., 2018). During training, typically additive random noise is added for exploration, so that the actual action taken when in state s takes the form $a = \mu(s; \theta) + \epsilon$ where ϵ is a K-dimensional Gaussian random vector with each component having zero mean and variance σ. During testing, ϵ is set to zero.

[1] https://anonymous.4open.science/r/e484a8c7-268a-4a66-a001-1e7676540237/

2.1 ENTROPY MAXIMIZATION RL

Maximum entropy reinforcement learning takes a different approach than (1) by optimizing policies to maximize both the expected return and the expected entropy of the policy (Ziebart et al., 2008; Ziebart, 2010; Todorov, 2008; Rawlik et al., 2013; Levine & Koltun, 2013; Levine et al., 2016; Nachum et al., 2017; Haarnoja et al., 2017; 2018a;b).

In particular, with maximization entropy RL, the objective is to maximize

$$V_\pi(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}_\pi[r(s(t), a(t)) + \lambda H(\pi(\cdot|s(t)))|s(0) = s] \tag{2}$$

where $H(\pi(\cdot|s))$ is the entropy of the policy when in state s, and the temperature parameter λ determines the relative importance of the entropy term against the reward.

For entropy maximization DRL, when given state s the policy network will typically output a K-dimensional vector $\sigma(s;\theta)$ in addition to the vector $\mu(s;\theta)$. The action selected when in state s is then modeled as $\mu(s;\theta) + \epsilon$ where $\epsilon \sim N(0, \sigma(s;\theta))$.

Maximum entropy RL has been touted to have a number of conceptual and practical advantages for DRL (Haarnoja et al., 2018a;b). For example, it has been argued that the policy is incentivized to explore more widely, while giving up on clearly unpromising avenues. It has also been argued that the policy can capture multiple modes of near-optimal behavior, that is, in problem settings where multiple actions seem equally attractive, the policy will commit equal probability mass to those actions. In this paper, we show for the Mujoco benchmarks that the standard additive noise exploration suffices and can achieve the same performance as maximum entropy RL.

3 THE SQUASHING EXPLORATION PROBLEM

3.1 BOUNDED ACTION SPACES

Continuous environments typically have bounded action spaces, that is, along each action dimension k there is a minimum possible action value $a_k^{\min}$ and a maximum possible action value $a_k^{\max}$. When selecting an action, the action needs to be selected within these bounds before the action can be taken. DRL algorithms often handle this by squashing the action so that it fits within the bounds. For example, if along any one dimension the value $\mu(s;\theta) + \epsilon$ exceeds $a_{\max}$, the action is set (clipped) to $a_{\max}$. Alternatively, a smooth form of squashing can be employed. For example, suppose $a_k^{\min} = -M$ and $a_k^{\max} = +M$ for some positive number M, then a smooth form of squashing could use $a = M \tanh(\mu(s;\theta)+\epsilon)$ in which $\tanh()$ is being applied to each component of the K-dimensional vector. DDPG (Hou et al., 2017) and TD3 (Fujimoto et al., 2018) use clipping, and SAC (Haarnoja et al., 2018a;b) uses smooth squashing with the $\tanh()$ function. For concreteness, henceforth we will assume that smooth squashing with the $\tanh()$ is employed.

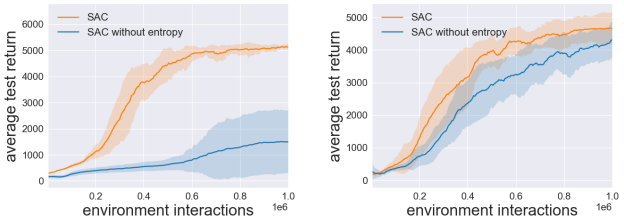
We note that an environment may actually allow the agent to input actions that are outside the bounds. In this case, the environment will typically first clip the actions internally before passing them on to the "actual" environment (Fujita & Maeda, 2018).

We now make a simple but crucial observation: squashing actions so that they fit into a bounded action space can have a disastrous effect on additive-noise exploration strategies. To see this, let the output of the policy network be denoted by $\mu(s) = (\mu_1(s), \ldots, \mu_K(s))$. Consider an action taken along one dimension k, and suppose $\mu_k(s) >> 1$ and $|\epsilon_k|$ is relatively small compared to $\mu_k(s)$. Then the action $a_k = M \tanh(\mu_k(s) + \epsilon_k)$ will be very close (essentially equal) to M. If the condition $\mu_k(s) >> 1$ persists over many consecutive states, then a_k will remain close to 1 for all these states, and consequently there will be essentially no exploration along the kth dimension. We will refer to this problem as the *squashing exploration problem*. A similar observation was made in Hausknecht & Stone (2015). We will argue that algorithms such as DDPG and TD3 based on the standard objective (1) with additive noise exploration can be greatly impaired by squashing exploration.

3.2 WHAT DOES ENTROPY MAXIMIZATION BRING TO SAC FOR THE MUJUCO ENVIRONMENTS?

SAC is a maximum-entropy based off-policy DRL algorithm which provides good performance across all of the Mujuco benchmark environments. To the best of our knowledge, it currently provides state of the art performance for the Mujoco benchmark. In this section, we argue that the principal contribution of the entropy term in the SAC objective is to resolve the squashing exploration problem, thereby maintaining sufficient exploration when facing bounded action spaces. To argue this, we consider two DRL algorithms: SAC with adaptive temperature (Haarnoja et al., 2018b), and SAC with entropy removed altogether (temperature set to zero) but everything else the same. We refer to them as *SAC* and as *SAC without entropy*. For SAC without entropy, for exploration we use additive zero-mean Gaussian noise with σ fixed at 0.3. Both algorithms use $\tanh$ squashing. We compare these two algorithms on two Mujoco environments: Humanoid-v2 and Walker-v2.

Figure 1 shows the performance of the two algorithms with 10 seeds. We see that for Humanoid, SAC with entropy maximization performs much better than SAC without entropy maximization. However, for Walker, SAC without entropy performs nearly as well as SAC, implying maximum entropy RL is not as critical for this environment.



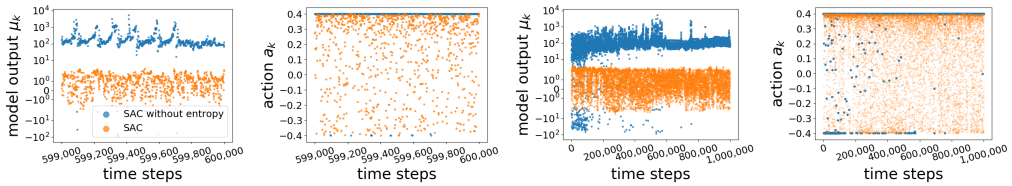
(a) Humanoid-v2 (b) Walker2d-v2

Figure 1: SAC performance with and without entropy maximization

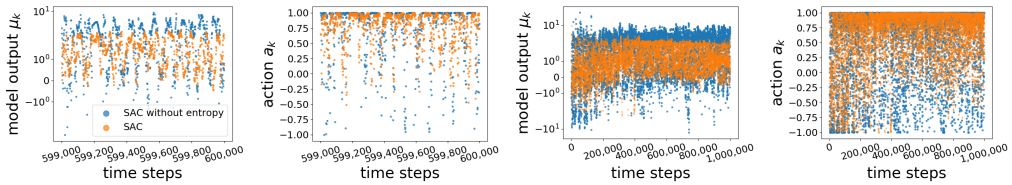
To understand why entropy maximization is important for one environment but less so for another, we examine the actions selected when training these two algorithms. Humanoid and Walker have action dimensions $K = 17$ and $K = 6$, respectively. Here we show representative results for one dimension for both environments, and provide the results for all the dimensions in the Appendix. The top and bottom rows of Figure 2 shows results for Humanoid and Walker, respectively. The first column shows the μ_k values for an interval of 1,000 consecutive time steps, namely, for time steps 599,000 to 600,000. The second column shows the actual action values passed to the environment again for time steps 599,000 to 600,000. The third and fourth columns show a concatenation of 10 such intervals of 1000 time steps, with each interval coming from a larger interval of 100,000 time steps. The first and third columns use a log scale on the y-axis.

The top and bottom rows of Figure 2 are strikingly different. For Humanoid using SAC (which uses entropy maximization), the $|\mu_k|$ values are small, mostly in the range [-1.5,1.5], and fluctuate significantly. This allows the action values to also fluctuate significantly, providing exploration in the action space. On the other hand, for SAC without entropy the $|\mu_k|$ values are typically huge, most of which are well outside the interval [-10,10]. This causes the actions a_k to be persistently clustered at either M or -M, leading to essentially no exploration along that dimension. As shown in the Appendix, this property (lack of exploration for SAC without entropy maximization) does not hold for just a few dimensions, but instead for all 17 dimensions. For Walker, we see that for both algorithms, the μ_k values are sensible, mostly in the range [-1,1] and therefore the actions chosen by both algorithms exhibit exploration.

In conclusion, the principle benefit of maximum entropy RL in SAC for the Mujuco environments is that it resolves the squashing exploration problem. For some environments (such as Walker), the outputs of the policy network take on sensible values, so that sufficient exploration is maintained and overall good performance is achieved without the need for entropy maximization. For other environments (such as Humanoid), entropy maximization is needed to reduce the magnitudes of the outputs so that exploration is maintained and overall good performance is achieved.



(a) Humanoid-v2



(b) Walker2d-v2

Figure 2: μ_k and a_k values from SAC and SAC without entropy maximization

4 Streamlined Off-Policy (SOP) Algorithm

Given the observations in the previous section, a natural question is: is it possible to design a streamlined off policy algorithm that does not employ entropy maximization but offers performance comparable to SAC (which has entropy maximization)?

As we observed in the previous section, without entropy maximization, in some environments the policy network output values $|\mu_k|$, $k = 1, \ldots, K$ can become persistently huge, which leads to insufficient exploration due to the squashing. A simple solution is to modify the outputs of the policy network by normalizing the output values when they collectively (across the action dimensions) become too large. To this end, let $\mu = (\mu_1, \ldots, \mu_K)$ be the output of the original policy network, and let $G = \sum_k |\mu_k|/K$. The G is simply the average of the magnitudes of the components of μ. The normalization procedure is as follows. If $G > 1$, then we reset $\mu_k \leftarrow \mu_k/G$ for all $k = 1, \ldots, K$; otherwise, we leave μ unchanged. With this simple normalization, we are assured that the average of the normalized magnitudes is never greater than one. Henceforth we assume the policy network has been modified with the simple normalization scheme just described.

Our Streamlined Off Policy (SOP) algorithm is described in Algorithm 1. The algorithm is essentially DDPG plus the normalization described above, plus double Q-learning and target policy smoothing (Fujimoto et al., 2018). Another way of looking at it is as TD3 plus the normalization described above, minus the delayed policy updates and the target policy parameters. SOP also uses $\tanh$ squashing instead of clipping, since $\tanh$ gives somewhat better performance in our experiments. The SOP algorithm is "streamlined" as it has no entropy terms, temperature adaptation, target policy parameters or delayed policy updates. In our experiments, we also consider TD3 plus the simple normalization, and also another streamlined algorithm in which we replace the simple normalization scheme described above with the inverting gradients scheme as described in Hausknecht & Stone (2015). Inverting gradients scheme builds on the idea that if parameter approaches its boundaries, gradients are downscaled; and gradients are inverted when the parameter exceeds the boundaries.

4.1 Experimental Results for SOP

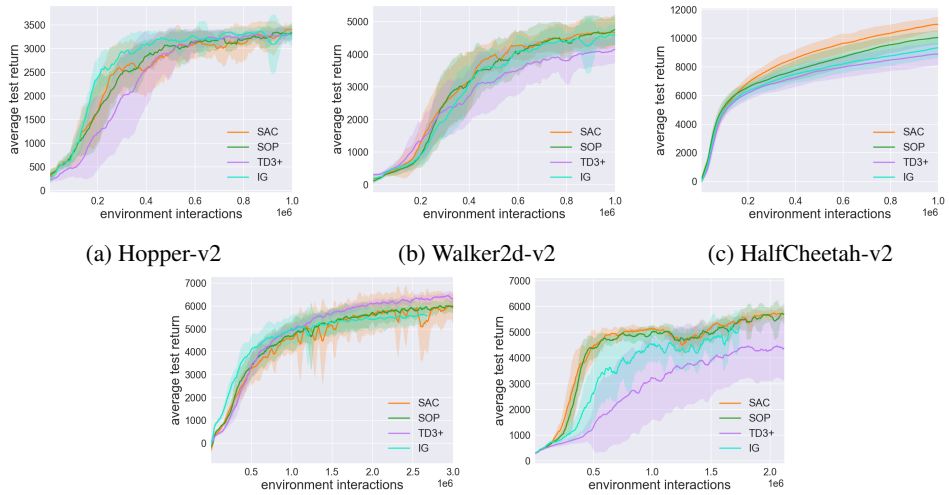
Figure 3 compares SAC (with temperature adaptation (Haarnoja et al., 2018a;b)) with SOP, TD3+ (that is, TD3 plus the simple normalization), and inverting gradients (IG) for five of the most challenging Mujuco environments. Using the same baseline code, we train with ten different random seeds for each of the two algorithms. Each algorithm performs five evaluation rollouts every 5000 environment steps. The solid curves correspond to the mean, and the shaded region to the standard deviation of the returns over the ten seeds.

Algorithm 1 Streamlined Off-Policy

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\phi_{\text{targ}_i} \leftarrow \phi_i$ for i = 1, 2
repeat
  Generate an episode using actions $a = M\tanh(\mu_\theta(s) + \epsilon)$ where $\epsilon \sim \mathcal{N}(0, \sigma_1)$.
  for j in range(however many updates) **do**
    Randomly sample a batch of transitions, $B = \{(s, a, r, s)\}$ from $\mathcal{D}$
    Compute targets for Q functions:
      $y_q(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{\text{targ}_i}}(s', M\tanh(\mu_\theta(s') + \delta)) \qquad \delta \sim \mathcal{N}(0, \sigma_2)$
    Update Q-functions by one step of gradient descent using
      $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left(Q_{\phi_i}(s, a) - y_q(r, s')\right)^2$ for $i = 1, 2$
    Update policy by one step of gradient ascent using
      $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, M \tanh(\mu_\theta(s)))$
    Update target networks with
      $\phi_{\text{targ}_i} \leftarrow \rho \phi_{\text{targ}_i} + (1 - \rho)\phi_i$ for $i = 1, 2$

Results show that SOP, SAC and IG have essentially the same sample-efficiency performance and robustness across all environments. IG tends to learn faster in the beginning in Hopper and Ant, but its final performance is about the same as the others. It also has slightly weaker performance for Humanoid and HalfCheetah. The performance of TD3+ is also good, although not quite as good as the other schemes, especially in humanoid and HalfCheetah. This result confirms that when using a simple output normalization in the policy network, the performance of SAC can be achieved without maximum entropy RL.

In the Appendix we provide an ablation study for SOP, which shows a major performance drop when removing either double Q-learning or normalization, whereas removing target policy smoothing (Fujimoto et al., 2018) results in only a small performance drop in some environments.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 3: Streamlined Off-Policy (SOP) versus SAC, TD3+ and IG

5 NON-UNIFORM SAMPLING

We now show how a small change in the sampling scheme for SOP can achieve state of the art performance for the Mujoco benchmark. We call this sampling scheme Emphasizing Recent Experience (ERE). ERE has 3 core features: (i) It is a general method applicable to any off-policy algorithm; (ii) It requires no special data structure, is very simple to implement, and has near-zero computational overhead; (iii) It only introduces one additional important hyper-parameter.

The basic idea is: during the parameter update phase, the first mini-batch is sampled from the entire buffer, then for each subsequent mini-batch we gradually reduce our range of sampling to sample more aggressively from more recent data. Specifically, assume that in the current update phase we are to make 1000 mini-batch updates. Let N be the max size of the buffer. Then for the kth update, we sample uniformly from the most recent c_k data points, where $c_k = N \cdot \eta^k$ and $\eta \in (0, 1]$ is a hyper-parameter that determines how much emphasis we put on recent data. $\eta = 1$ is uniform sampling. When $\eta < 1$, c_k decreases as we perform each update. η can made to adapt to the learning speed of the agent so that we do not have to tune it for each environment.

The effect of such a sampling formulation is twofold. The first is recent data have a higher chance of being sampled. The second is that we do this in an ordered way: we first sample from all the data in the buffer, and gradually shrink the range of sampling to only sample from the most recent data. This scheme reduces the chance of over-writing parameter changes made by new data with parameter changes made by old data (French, 1999; McClelland et al., 1995; McCloskey & Cohen, 1989; Ratcliff, 1990; Robins, 1995). This process allows us to quickly obtain new information from recent data, and better approximate the value functions near recently-visited states, while still maintaining an acceptable approximation near states visited in the more distant past.

What is the effect of replacing uniform sampling with ERE? First note if we do uniform sampling on a fixed buffer, the expected number of times a data point is sampled is the same for all data points. Now consider a scenario where we have a buffer of size 1000 (FIFO queue), we collect one data point at a time, and we then perform one update with mini-batch size of one. If we start with an empty buffer and sample uniformly, as data fills the buffer, each data point gets less and less chance of being sampled. Specifically, over a period of 1000 updates, the expected number of times the tth data point is sampled is: $1/t + 1/(t + 1) + \cdots + 1/T$. Figure 4f shows the expected number of times a data point is sampled as a function of its position in the buffer. We see that older data points have a much higher expected number of times of being sampled compared to newer data points. This is undesirable because when the agent is improving and exploring new areas of the state space; the new data points may contain more interesting information than the old ones, which have already been updated many times.

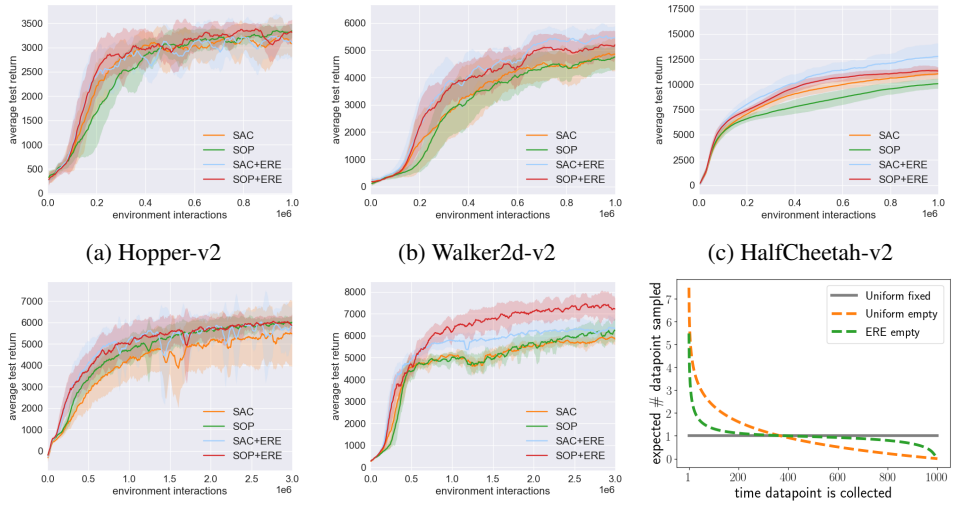
When we apply the ERE scheme, we effectively skew the curve towards assigning higher expected number of samples for the newer data, allowing the newer data to be frequently sampled soon after being collected, which can accelerate the learning process. In the Appendix, we provide further algorithmic detail and analysis on ERE, and compare ERE to two other sampling schemes: an exponential sampling scheme and Prioritized Experience Replay (Schaul et al., 2015).

5.1 Experimental results for SOP+ERE

Figure 4 compares the performance of SOP, SOP+ERE, SAC and SAC+ERE. With ERE, both SAC and SOP gain a significant performance improvement in all environments. SOP+ERE learns faster than SAC and vanilla SOP in all Mujoco environments. SOP+ERE also greatly improves overall performance for the two most challenging environments, Ant and Humanoid, and has the best performance for Humanoid. In table 1, we show the mean test episode return and std across 10 random seeds at 1M timesteps for all environments. The last column displays the percentage improvement of SOP+ERE over SAC, showing that SOP+ERE achieves state of the art performance. In both Ant and Humanoid, SOP+ERE improves average performance by 24% over SAC at 1 million timesteps. As for the std, SOP+ERE gives lower values, and for Humanoid a higher value.

Table 1: Performance comparison at one million samples. Last column shows percentage improvement of SOP+ERE over SAC.

Environment	SAC Adaptive	SOP	SOP+ERE	Improvement
Hopper	3161.2 ± 381.0	$\mathbf{3317.3} \pm 133.9$	3201.5 ± 248.7	1.3%
Walker	4801.5 ± 514.5	4666.5 ± 474.5	$\mathbf{5145.9} \pm 512.3$	7.2%
HalfCheetah	10963.7 ± 512.4	9968.0 ± 497.4	$\mathbf{11335.1} \pm 478.3$	3.4%
Ant	4153.7 ± 925.0	4674.0 ± 588.8	$\mathbf{5023.3} \pm 891.6$	21.0%
Humanoid	5076.2 ± 148.1	4900.9 ± 316.6	$\mathbf{6297.7} \pm 500.0$	24.1%



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2 (f) Uniform and ERE sampling

Figure 4: (a) to (e) show the performance of Streamlined Off-Policy (SOP) and SAC with ERE sampling. (f) shows over a period of 1000 updates, the expected number of times the tth data point is sampled (with $\eta = 0.996$). ERE allows new data to be sampled many times soon after being collected.

6 Related work

In recent years, there has been significant progress in improving the sample efficiency of DRL for continuous robotic locomotion tasks with off-policy algorithms (Lillicrap et al., 2015; Fujimoto et al., 2018; Haarnoja et al., 2018a;b). There is also a significant body of research on maximum entropy RL methods (Ziebart et al., 2008; Ziebart, 2010; Todorov, 2008; Rawlik et al., 2013; Levine & Koltun, 2013; Levine et al., 2016; Nachum et al., 2017; Haarnoja et al., 2017; 2018a;b). Some very recent papers also shed light on how entropy might provide a smoother optimization landscape (Ahmed et al., 2019).

By taking clipping in the Mujoco environments explicitly into account, Fujita & Maeda (2018) modified the policy gradient algorithm to reduce variance and provide superior performance among on-policy algorithms. Eisenach et al. (2018) extend the work of Fujita & Maeda (2018) for when an action may be direction. Hausknecht & Stone (2015) and Chou et al. (2017) also explores DRL in the context of bounded action spaces. Dalal et al. (2018) consider safe exploration in the context of constrained action spaces.

Uniform sampling is the most common way to sample from a replay buffer. One of the most well-known alternatives is prioritized experience replay (PER) (Schaul et al., 2015). PER uses the absolute TD-error of a data point as the measure for priority, and data points with higher priority will have a higher chance of being sampled. This method has been tested on DQN (Mnih et al., 2015) and double DQN (DDQN) (Van Hasselt et al., 2016) with significant improvement. PER has been combined with the dueling architecture (Wang et al., 2015), with an ensemble of recurrent DQN (Schulze & Schulze, 2018), and PER is one of six crucial components in Rainbow (Hessel et al., 2018), which achieves state-of-the-art on the Atari game environments. PER has also been successfully applied to other algorithms such as DDPG (Hou et al., 2017) and can be implemented in a distributed manner (Horgan et al., 2018). There are other methods proposed to make better use of the replay buffer. In Sample Efficient Actor-Critic with Experience Replay (ACER), the algorithm has an on-policy part and an off-policy part, with a hyper-parameter controlling the ratio of off-policy updates to on-policy updates (Wang et al., 2016). The RACER algorithm (Novati & Koumoutsakos, 2018) selectively removes data points from the buffer, based on the degree of "off-policyness" which is measured by their importance sampling weight, bringing improvement to DDPG (Lillicrap et al., 2015), NAF (Gu et al., 2016) and PPO (Schulman et al., 2017). In De Bruin et al. (2015), replay buffers of different sizes were tested on DDPG, and result shows that a large enough buffer with enough data diversity

can lead to better performance. Finally, with Hindsight Experience Replay (HER)(Andrychowicz et al., 2017), priority can be given to trajectories with lower density estimation(Zhao & Tresp, 2019) to tackle multi-goal, sparse reward environments.

7 CONCLUSION

In this paper we first showed that the primary role of maximum entropy RL for the Mujoco benchmark is to maintain satisfactory exploration in the presence of bounded action spaces. We then developed a new streamlined algorithm which does not employ entropy maximization but nevertheless matches the sampling efficiency and robustness performance of SAC for the Mujoco benchmarks. Our experimental results demonstrate a need to revisit the benefits of entropy regularization in DRL. Finally, we combined our streamlined algorithm with a simple non-uniform sampling scheme to achieve state-of-the art performance for the Mujoco benchmark.

REFERENCES

Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning*, pp. 151–160, 2019.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996.

Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 834–843. JMLR. org, 2017.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Tim De Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep reinforcement learning workshop, NIPS*, 2015.

Carson Eisenach, Haichuan Yang, Ji Liu, and Han Liu. Marginal policy gradients: A unified family of estimators for bounded action spaces with applications. *arXiv preprint arXiv:1806.05134*, 2018.

Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. *arXiv preprint arXiv:1902.10250*, 2019.

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

Yasuhiro Fujita and Shin-ichi Maeda. Clipped action policy gradient. *arXiv preprint arXiv:1802.07564*, 2018.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pp. 2829–2838, 2016.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1352–1361. JMLR. org, 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*, 2018a.

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

Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space. *arXiv preprint arXiv:1511.04143*, 2015.

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.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 316–321. IEEE, 2017.

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

Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pp. 1–9, 2013.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

James L McClelland, Bruce L McNaughton, and Randall C O'reilly. Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory. *Psychological review*, 102(3):419, 1995.

Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pp. 109–165. Elsevier, 1989.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2775–2785, 2017.

Guido Novati and Petros Koumoutsakos. Remember and forget for experience replay. *arXiv preprint arXiv:1807.05827*, 2018.

Martin L Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

Roger Ratcliff. Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285, 1990.

Konrad Rawlik, Marc Toussaint, and Sethu Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2): 123–146, 1995.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

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

Christopher Schulze and Marcus Schulze. Vizdoom: Drqn with prioritized experience replay, double-q learning and snapshot ensembling. In *Proceedings of SAI Intelligent Systems Conference*, pp. 1–17. Springer, 2018.

Emanuel Todorov. General duality between optimal control and estimation. In *2008 47th IEEE Conference on Decision and Control*, pp. 4286–4292. IEEE, 2008.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, pp. 5. Phoenix, AZ, 2016.

Quan Vuong, Yiming Zhang, and Keith W Ross. Supervised policy update for deep reinforcement learning. *arXiv preprint arXiv:1805.11706*, 2018.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Rui Zhao and Volker Tresp. Curiosity-driven experience prioritization via density estimation. *arXiv preprint arXiv:1902.08039*, 2019.

Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. PhD thesis, figshare, 2010.

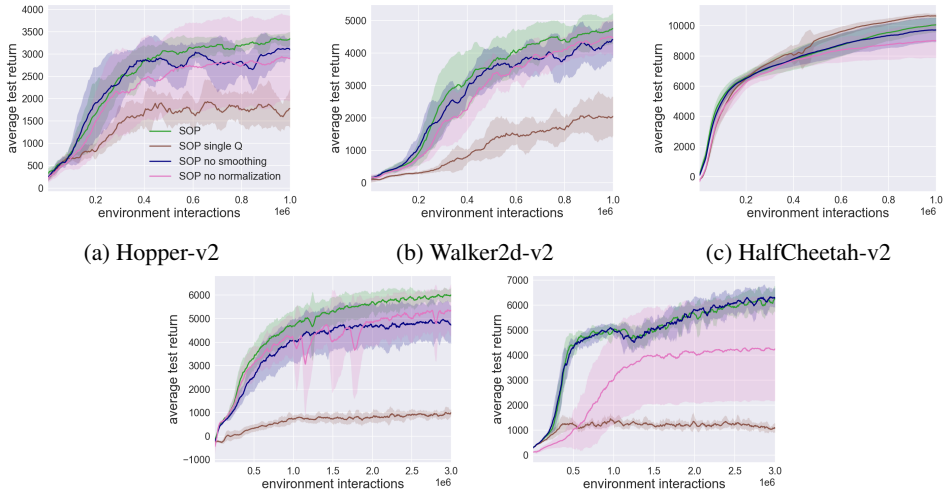
Brian D Ziebart, Andrew Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. 2008.

A ABLATION STUDY

In this ablation study we separately examine the importance of (i) the normalization at the output of the policy network; (ii) the double Q networks; (iii) and randomization used in the line 8 of the SOP algorithm (that is, target policy smoothing (Fujimoto et al., 2018)).

Figure 5 shows the results for the five environments considered in this paper. In Figure 5, "no normalization" is SOP without the normalization of the outputs of the policy network; "single Q" is SOP with one Q-network instead of two; and "no smoothing" is SOP without the randomness in line 8 of the algorithm.

Figure 5 confirms that double Q-networks are critical for obtaining good performance (Van Hasselt et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2018a). Figure 5 also shows that output normalization is also critical. Without output normalization, performance fluctuates wildly, and average performance can decrease dramatically, particularly for Humanoid and HalfCheetah. Target policy smoothing improves performance by a relatively small amount.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 5: Ablation Study

B HYPERPARAMETERS

Table 2 shows hyperparameters used for SOP, SOP+ERE and SOP+PER. For adaptive SAC, we use our own PyTorch implementation for the comparisons. Our implementation uses the same hyperparameters as used in the original paper (Haarnoja et al., 2018b). Our implementation of SOP variants and adaptive SAC share most of the code base, to make comparisons fair.

For the η value in the ERE scheme, in our early experiments we tried the values (0.993, 0.994, 0.995, 0.996, 0.997, 0.998) on the Ant environment and found 0.995 to work pretty well. This initial range of values was decided by computing the ERE sampling range for the oldest data, since we found that for smaller values, the range would simply be too small, we did not try any value smaller than 0.993. For the PER scheme, we did some informal preliminary search, and then searched on Ant environment for β_1 in (0, 0.4, 0.6, 0.8), β_2 in (0, 0.4, 0.5, 0.6, 1), and learning rate in (1e-4, 2e-4, 3e-4, 5e-4, 8e-4, 1e-3), we decided to search these values because the original paper used $\beta_1 = 0.6$, $\beta_2 = 0.4$ and with reduced learning rate. For the exponential sampling scheme, we searched the λ value in (3e-7, 1e-6, 3e-6, 5e-6, 1e-5, 3e-5, 5e-5, 1e-4) in Ant, this search range was decided by plotting out the probabilities of sampling, and then pick a set of values that are not too extreme. For σ in SOP, in some of our early experiments with SAC, we accidentally found that $\sigma = 0.3$ gives good performance for SAC without entropy and with Gaussian noise. We searched values (0.27, 0.28, 0.29, 0.3).

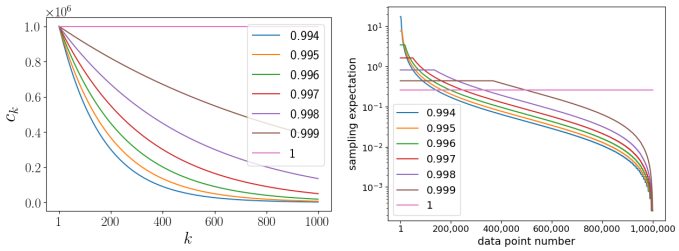
Table 2: SOP Hyperparameters

Parameter	Value
Shared	
optimizer	Adam (Kingma & Ba, 2014)
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
target smoothing coefficient (ρ)	0.005
target update interval	1
replay buffer size	10^6
number of hidden layers for all networks	2
number of hidden units per layer	256
mini-batch size	256
nonlinearity	ReLU
SAC adaptive	
entropy target	dim($\mathcal{A}$) (e.g., 6 for HalfCheetah-v2)
SOP	
gaussian noise std $\sigma = \sigma_1 = \sigma_2$	0.29
ERE	
ERE initial η_0	0.995
PER	
PER β_1 (α in PER paper)	0.4
PER β_2 (β in PER paper)	0.4
EXP	
Exponential λ	$5e-06$

C ERE PSEUDOCODE

Algorithm 2 SOP with Emphasizing Recent Experience

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$ of size N, initial η_0, recent and max performance improvement $I_{recent} = I_{max} = 0$.
Set target parameters equal to main parameters $\phi_{\text{targ,i}} \leftarrow \phi_i$ for i = 1, 2
repeat
  Generate an episode using actions $a = M\tanh(\mu_\theta(s) + \epsilon)$ where $\epsilon \sim \mathcal{N}(0, \sigma_1)$.
  update I_{recent}, I_{max} with training episode returns, let $K =$ length of episode
  compute $\eta = \eta_0 \cdot \frac{I_{recent}}{I_{max}} + (1 - \frac{I_{recent}}{I_{max}})$
  for j in range(K) **do**
    Compute $c_k = N \cdot \eta^{k\frac{1000}{K}}$
    Sample a batch of transitions, $B = \{(s, a, r, s)\}$ from most recent c_k data in $\mathcal{D}$
    Compute targets for Q functions:
      $y_q(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', M\tanh(\mu_\theta(s') + \delta)) \quad \delta \sim \mathcal{N}(0, \sigma_2)$
    Update Q-functions by one step of gradient descent using
      $\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} (Q_{\phi,i}(s, a) - y_q(r, s'))^2$ for $i = 1, 2$
    Update policy by one step of gradient ascent using
      $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi,1}(s, M \tanh(\mu_\theta(s)))$
    Update target networks with
      $\phi_{\text{targ, i}} \leftarrow \rho\phi_{\text{targ, i}} + (1 - \rho)\phi_i$ for $i = 1, 2$



(a) (b)

Figure 6: Effect of different η values. The plots assume a replay buffer with 1 million samples, and 1,000 mini-batches of size 256 in an update phase. Figure 6a plots c_k (ranging from 0 to 1 million) as a function of k (ranging from 1 to 1,000). Figure 6b plots the expected number of times a data point in the buffer is sampled, with the data points ordered from most to least recent.

D SOP WITH OTHER SAMPLING SCHEMES

We also investigate the effect of other interesting sampling schemes.

D.1 SAC WITH PRIORITIZED EXPERIENCE REPLAY

We also implement the proportional variant of Prioritized Experience Replay (Schaul et al., 2015) with SOP.

Since SOP has two Q-networks, we redefine the absolute TD error $|\delta|$ of a transition (s, a, r, s') to be the average absolute TD error in the Q network update:

$$|\delta| = \frac{1}{2}\sum_{l=1}^{2} |y_q(r, s') - Q_{\phi,l}(s, a)| \tag{3}$$

Within the sum, the first term $y_q(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tanh(\mu_\theta(s') + \delta))$, $\delta \sim \mathcal{N}(0, \sigma_2)$ is simply the target for the Q network, and the term $Q_{\theta,l}(s, a)$ is the current estimate of the lth Q network. For the ith data point, the definition of the priority value p_i is $p_i = |\delta_i| + \epsilon$. The probability of sampling a data point $P(i)$ is computed as:

$$P(i) = \frac{p_i^{\beta_1}}{\sum_j p_j^{\beta_1}} \tag{4}$$

where β_1 is a hyperparameter that controls how much the priority value affects the sampling probability, which is denoted by α in Schaul et al. (2015), but to avoid confusion with the α in SAC, we denote it as β_1. The importance sampling (IS) weight w_i for a data point is computed as:

$$w_i = (\frac{1}{N} \cdot \frac{1}{P(i)})^{\beta_2} \tag{5}$$

where β_2 is denoted as β in Schaul et al. (2015).

Based on the SOP algorithm, we change the sampling method from uniform sampling to sampling using the probabilities $P(i)$, and for the Q updates we apply the IS weight w_i. This gives SOP with Prioritized Experience Replay (SOP+PER). We note that as compared with SOP+PER, ERE does not require a special data structure and has negligible extra cost, while PER uses a sum-tree structure with some additional computational cost. We also tried several variants of SOP+PER, but preliminary results show that it is unclear whether there is improvement in performance, so we kept the algorithm simple.

D.2 SOP WITH EXPONENTIAL SAMPLING

The ERE scheme is similar to an exponential sampling scheme where we assign the probability of sampling according to the probability density function of an exponential distribution. Essentially,

in such a sampling scheme, the more recent data points get exponentially more probability of being sampled compared to older data.

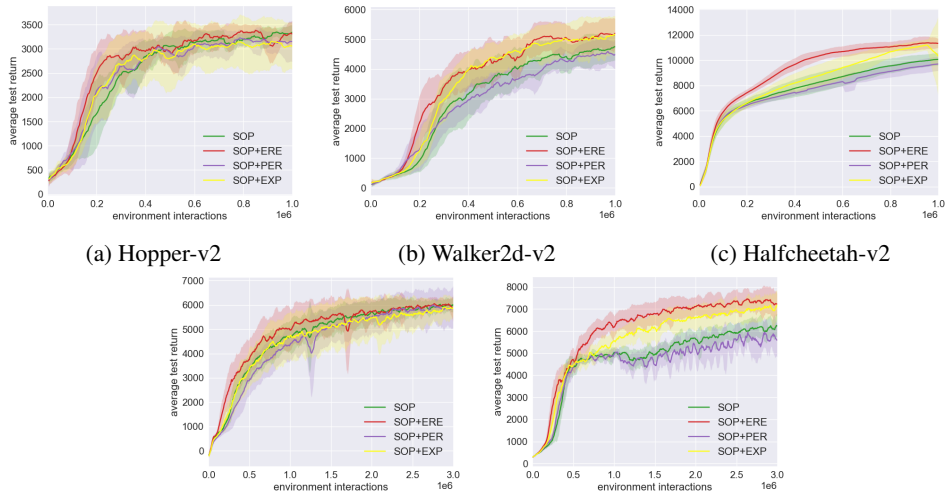
For the ith most recent data point, the probability of sampling a data point $P(i)$ is computed as:

$$P(i) = \lambda e^{-\lambda x} \tag{6}$$

We apply this sampling scheme to SOP and refer to this variant as SOP+EXP.

D.3 PER AND EXP EXPERIMENT RESULTS

Figure 7 shows a performance comparison of SOP, SOP+ERE, SOP+EXP and SOP+PER. Results show that the exponential sampling scheme gives a boost to the performance of SOP, and especially in the Humanoid environment, although not as good as ERE. Surprisingly, SOP+PER does not give a significant performance boost to SOP (if any boost at all). We also found that it is difficult to find hyperparameter settings for SOP+PER that work well for all environments. Some of the other hyperparameter settings actually reduce performance. It is unclear why PER does not work so well for SOP. A similar result has been found in another recent paper (Fu et al., 2019), showing that PER can significantly reduce performance on TD3. Further research is needed to understand how PER can be successfully adapted to environments with continuous action spaces and dense reward structure.



(a) Hopper-v2 (b) Walker2d-v2 (c) Halfcheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 7: Streamlined Off-Policy (SOP), with ERE and PER sampling schemes

E ADDITIONAL ERE ANALYSIS

Figure 6 shows, for fixed η, how η affects the data sampling process, under the ERE sampling scheme. Recent data points have a much higher probability of being sampled compared to older data, and a smaller η value gives more emphasis to recent data.

Different η values are desirable depending on how fast the agent is learning and how fast the past experiences become obsolete. So to make ERE work well in different environments with different reward scales and learning progress, we adapt η to the the speed of learning. To this end, define performance to be the training episode return. Define I_{recent} to be how much performance improved from $N/2$ timesteps ago, and I_{max} to be the maximum improvement throughout training, where N is the buffer size. Let the hyperparameter η_0 be the initial η value. We then adapt η according to the formula: $\eta = \eta_0 \cdot I_{recent}/I_{max} + 1 - (I_{recent}/I_{max})$.

Under such an adaptive scheme, when the agent learns quickly, the η value is low in order to learn quickly from new data. When progress is slow, η is higher to make use of the stabilizing effect of uniform sampling from the whole buffer.

F ADDITIONAL IMPLEMENTATION DETAILS

F.1 ERE IMPLEMENTATION

In this section we discuss some programming details. These details are not necessary for understanding the algorithm, but they might help with reproducibility.

In the ERE scheme, the sampling range always starts with the entire buffer (1M data) and then gradually shrinks. This is true even when the buffer is not full. So even if there are not many data points in the buffer, we compute c_k based as if there are 1M data points in the buffer. One can also modify the design slightly to obtain a variant that uses the current amount of data points to compute c_k. In addition to the reported scheme, we also tried shrinking the sampling range linearly, but it gives less performance gain.

In our implementation we set the number of updates after an episode to be the same as the number of timesteps in that episode. Since environments do not always end at 1000 timesteps, we can give a more general formula for c_k. Let K be the number of mini-batch updates, let N be the max size of the replay buffer, then:

$$c_k = N \cdot \eta^{k \frac{1000}{K}} \tag{7}$$

With this formulation, the range of sampling shrinks in more or less the same way with varying number of mini-batch updates. We always do uniform sampling in the first update, and we always have $\eta^{K \frac{1000}{K}} = \eta^{1000}$ in the last update.

When η is small, c_k can also become small for some of the mini-batches. To prevent getting a mini-batch with too many repeating data points, we set the minimum value for c_k to 5000. We did not find this value to be too important and did not find the need to tune it. It also does not have any effect for any $\eta \geq 0.995$ since the sampling range cannot be lower than 6000.

In the adaptive scheme with buffer of size 1M, the recent performance improvement is computed as the difference of the current episode return compared to the episode return 500,000 timesteps earlier. Before we reach 500,000 timesteps, we simply use η_0. The exact way of computing performance improvement does not have a significant effect on performance as long as it is reasonable.

F.2 PROGRAMMING AND COMPUTATION COMPLEXITY

In this section we give analysis on the additional programming and computation complexity brought by ERE and PER.

In terms of programming complexity, ERE is a clear winner since it only requires a small adjustment to how we sample mini-batches. It does not modify how the buffer stores the data, and does not require a special data structure to make it work efficiently. Thus the implementation difficulty is minimal. PER (proportional variant) requires a sum-tree data structure to make it run efficiently. The implementation is not too complicated, but compared to ERE it is a lot more work.

The exponential sampling scheme is very easy to implement, although a naive implementation will incur a significant computation overhead when sampling from a large buffer. To improve its computation efficiency, we instead uses an approximate sampling method. We first sample data indexes from segments of size 100 from the replay buffer, and then for each segment sampled, we sample one data point uniformly from that segment.

In terms of computation complexity (not sample efficiency), and wall-clock time, ERE's extra computation is negligible. In practice we observe no difference in computation time between SOP and SOP+ERE. PER needs to update the priority of its data points constantly and compute sampling probabilities for all the data points. The complexity for sampling and updates is $O(log(N))$, and the rank-based variant is similar (Schaul et al., 2015). Although this is not too bad, it does impose a significant overhead on SOP: SOP+PER runs twice as long as SOP. Also note that this overhead grows linearly with the size of the mini-batch. The overhead for the Mujoco environments is higher compared to Atari, possibly because the Mujoco environments have a smaller state space dimension while a larger batch size is used, making PER take up a larger portion of computation cost. For

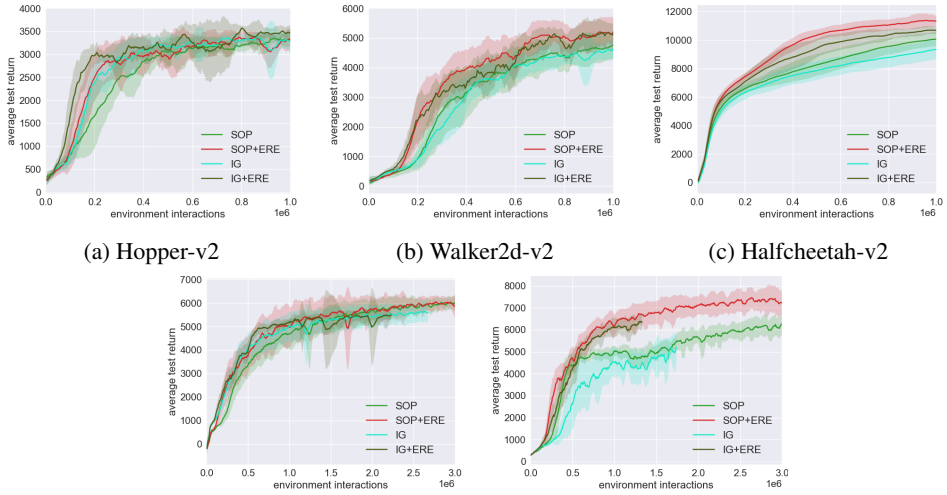
the exponential sampling scheme, the extra computation is also close to negligible when using the approximate sampling method.

In terms of the proposed normalization scheme and the Inverting Gradients (IG) method, the normalization is very simple and can be easily implemented and has negligible computation overhead. IG has a simple idea, but its implementation is slightly more complicated than the normalization scheme. When implemented naively, IG can have a large computation overhead, but it can be largely avoided by making sure the gradient computation is still done in a batch-manner. We have made a very efficient implementation and our code is publicly available so that interested reader can easily reproduce it.

G ADDITIONAL EXPERIMENTAL RESULTS

G.1 INVERTING GRADIENTS WITH ERE

In Figure 8 we show additional results on applying ERE to SOP+IG. The result shows that after applying the ERE scheme, SOP and IG both get a performance boost. The performance of the SOP+ERE and IG+ERE are similar.



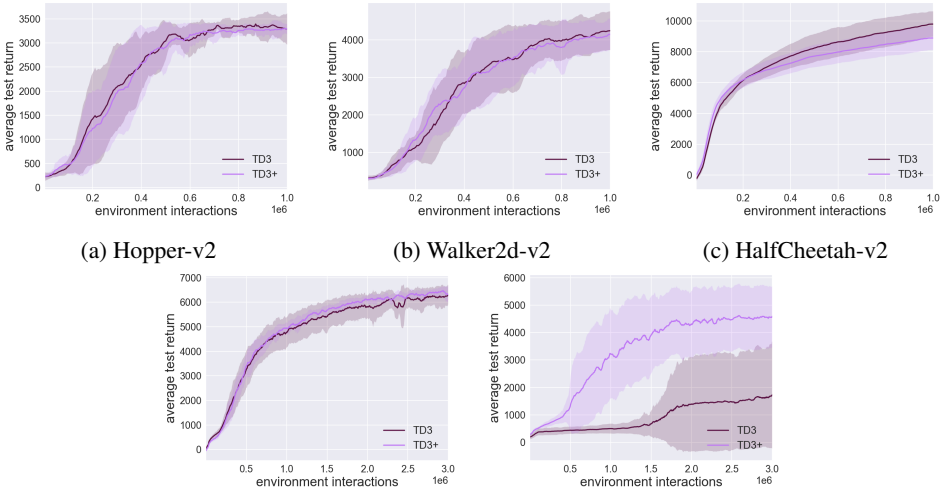
(a) Hopper-v2 (b) Walker2d-v2 (c) Halfcheetah-v2

(d) Ant-v2 (e) Humanoid-v2

Figure 8: SOP and inverting gradients with ERE sampling scheme

G.2 TD3 VERSUS TD3+

In figure 9, we show additional results comparing TD3 with TD3 plus our normalization scheme, which we refer as TD3+. The results show that after applying our normalization scheme, TD3+ has a big performance boost in Humanoid, while in other environments, both algorithms achieve similar performance.



(a) Hopper-v2 (b) Walker2d-v2 (c) HalfCheetah-v2

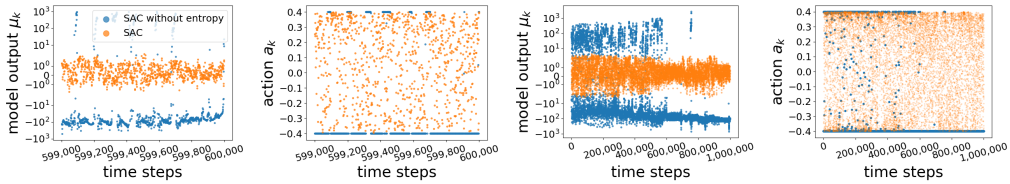
(d) Ant-v2 (e) Humanoid-v2

Figure 9: TD3 versus TD3+ (TD3 plus the normalization scheme)

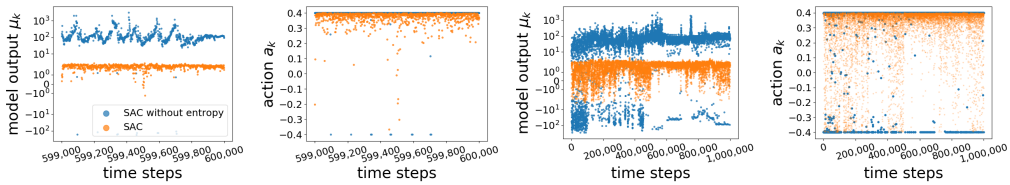
H ADDITIONAL ANALYSIS AND RESULTS COMPARING SAC WITH AND WITHOUT ENTROPY

To understand why entropy maximization is important for one environment but less so for another, we examine the actions selected when training SAC with and without entropy. Humanoid and Walker2d have action dimensions $K = 17$ and $K = 6$, respectively. In addition to the representative results shown for one dimension for both environments in Section 3.2, the results for all the dimensions are provided here in Figures 10 and 11.

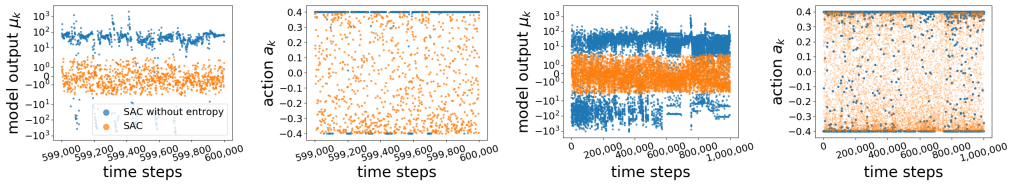
From Figure 10, we see that for Humanoid using SAC (which uses entropy maximization), the $|\mu_k|$ values are small and fluctuate significantly for all 17 dimensions. On the other hand, for SAC without entropy the $|\mu_k|$ values are typically huge, again for all 17 dimensions. This causes the actions a_k to be persistently clustered at either M or -M. As for Walker, the $|\mu_k|$ values are sensible for both algorithms for all 6 dimensions, as shown in figure 11.



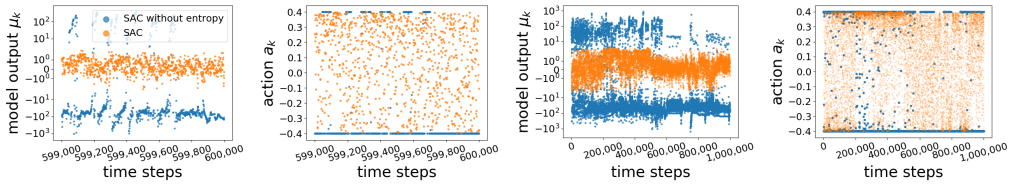
(a) k = 1



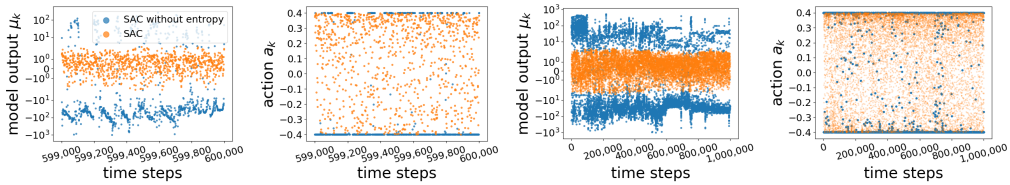
(b) k = 2



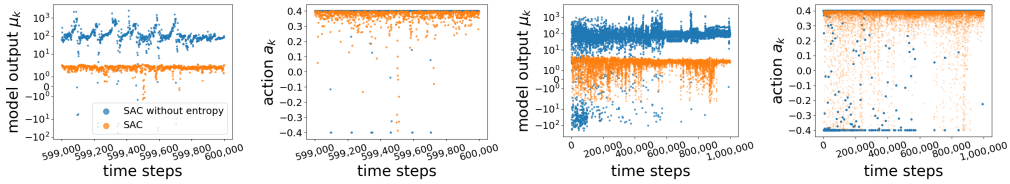
(c) k = 3



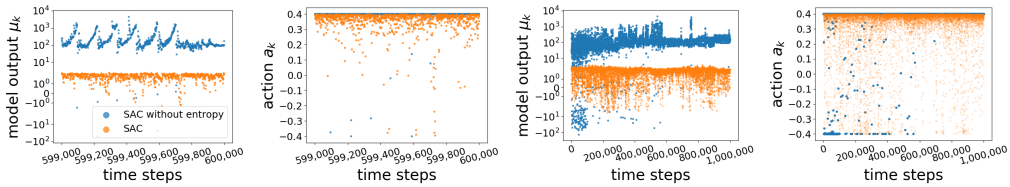
(d) k = 4



(e) k = 5

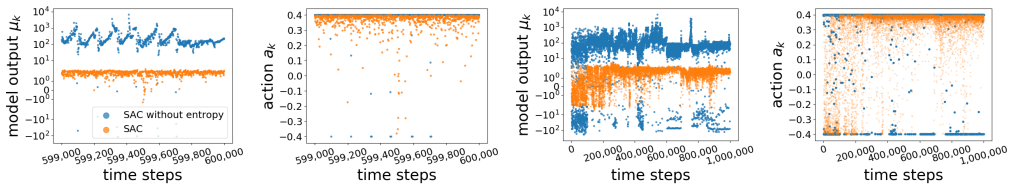


(f) k = 6

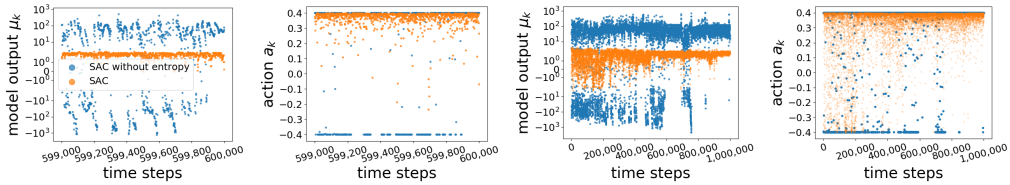


(g) k = 7

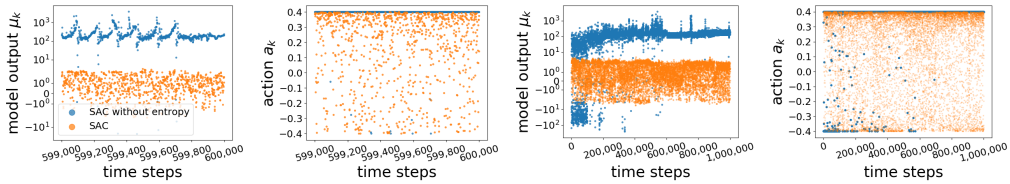
Figure 10: Humanoid-v2: μ_k and a_k values from SAC and SAC without entropy maximization



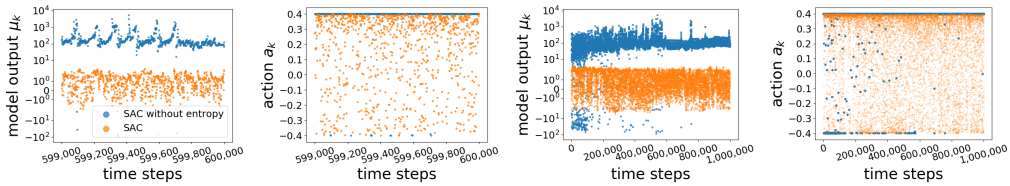
(h) k = 8



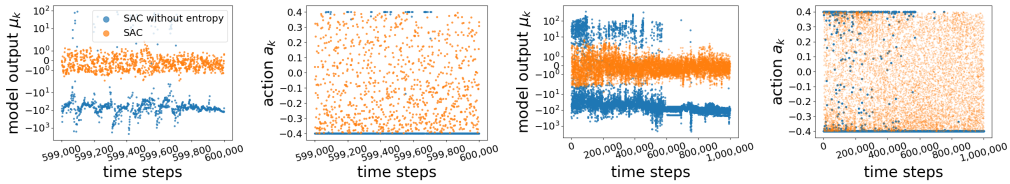
(i) k = 9



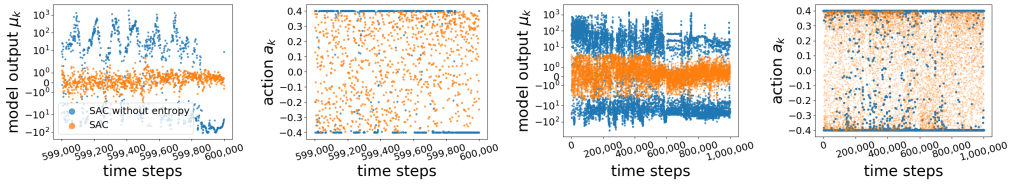
(j) k = 10



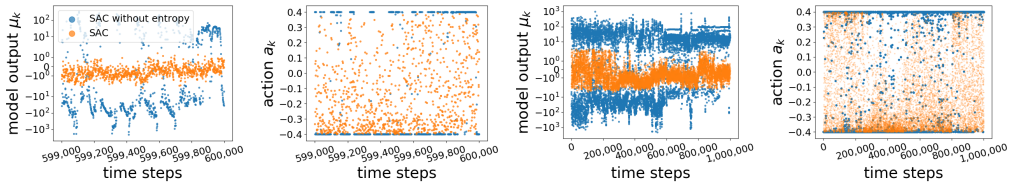
(k) k = 11



(l) k = 12

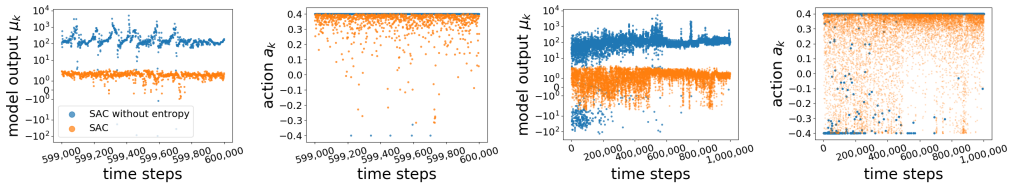


(m) k = 13

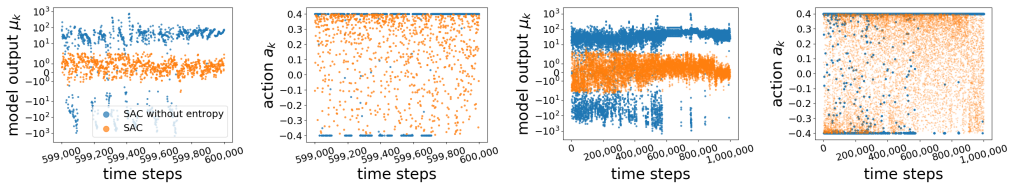


(n) k = 14

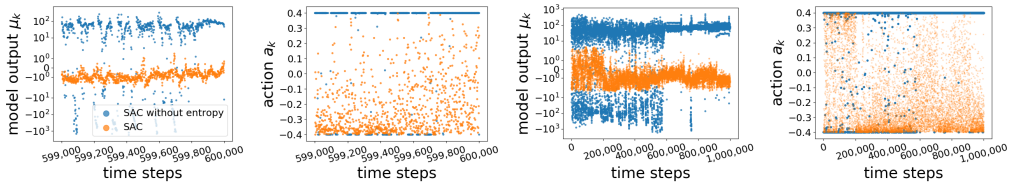
Figure 10: Humanoid-v2: μ_k and a_k values from SAC and SAC without entropy maximization



(o) k = 15

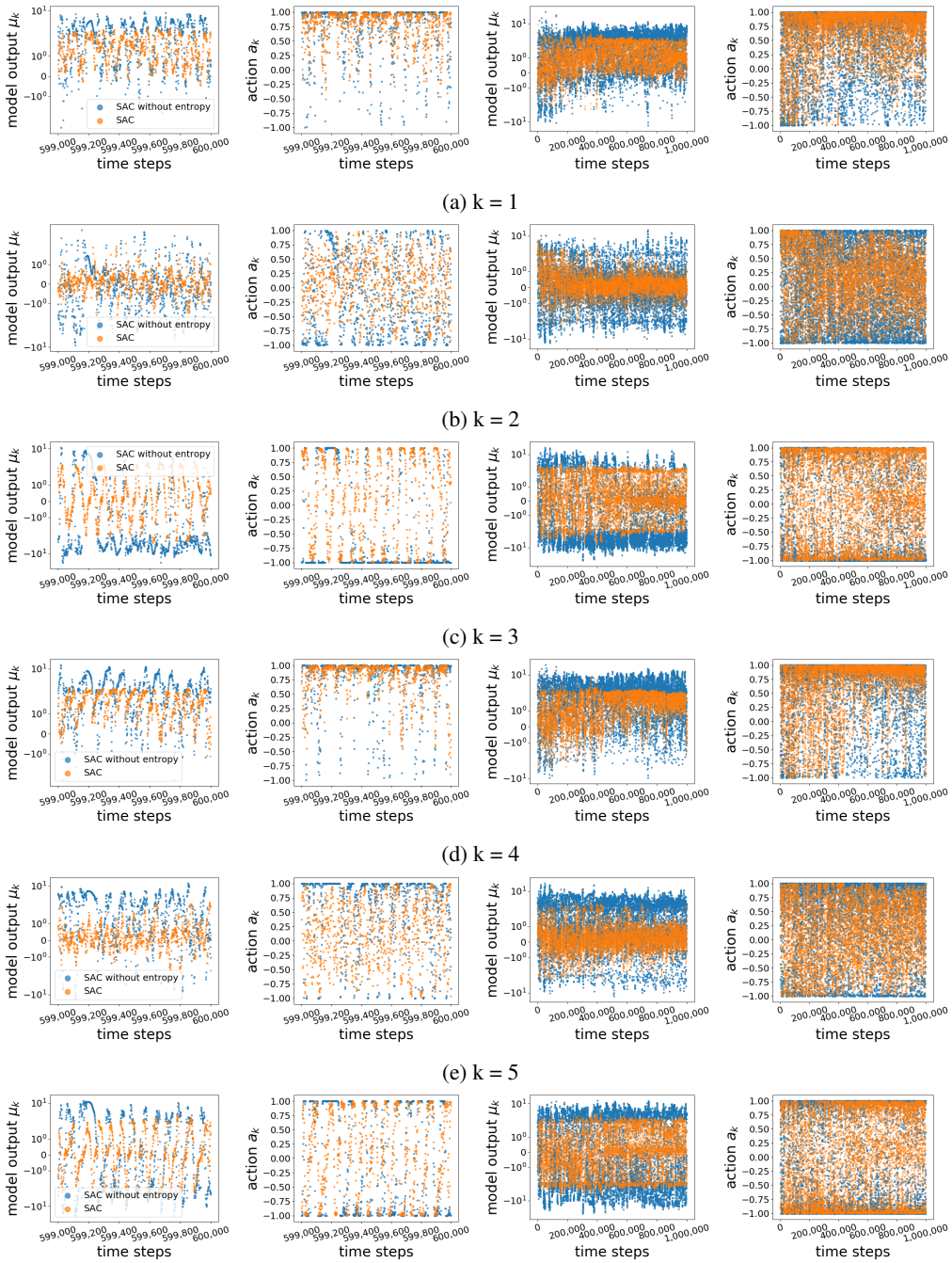


(p) k = 16



(q) k = 17

Figure 10: Humanoid-v2: μ_k and a_k values from SAC and SAC without entropy maximization



(a) k = 1

(b) k = 2

(c) k = 3

(d) k = 4

(e) k = 5

(f) k = 6

Figure 11: Walker2d-v2: μ_k and a_k values from SAC and SAC without entropy maximization