Mirror Descent Policy Optimization

Manan Tomar
Facebook AI Research

Lior Shani
Technion, Israel

Yonathan Efroni
Technion, Israel

Mohammad Ghavamzadeh
Google Research

Abstract

Mirror descent (MD), a well-known first-order method in constrained convex optimization, has recently been shown as an important tool to analyze trust-region algorithms in reinforcement learning (RL). Inspired by such theoretical analyses, we propose an efficient RL algorithm, called *mirror descent policy optimization* (MDPO). MDPO iteratively updates the policy by approximately solving a trust-region problem, whose objective function consists of two terms: a linearization of the standard RL objective and a proximity term that restricts two consecutive policies to be close to each other. Each update performs this approximation by taking multiple gradient steps on this objective function. We derive *on-policy* and *off-policy* variants of MDPO, while emphasizing important design choices motivated by the existing theory of MD in RL. We highlight the connections between on-policy MDPO and two popular trust-region RL algorithms: TRPO and PPO, and show that explicitly enforcing the trust-region constraint is in fact *not* a necessity for high performance gains in TRPO. We then show how the popular soft actor-critic (SAC) algorithm can be derived by slight modifications of off-policy MDPO. Overall, MDPO is derived from the MD principles, offers a unified approach to viewing a number of popular RL algorithms, and performs better than or on-par with TRPO, PPO, and SAC in a number of continuous control tasks.

1 Introduction

An important class of reinforcement learning (RL) algorithms consider an additional objective in their policy optimization that aims at constraining the consecutive policies to remain close to each other. These algorithms are referred to as *trust region* or *proximity-based*, resonating the fact that they make the new policy to lie within a trust-region around the old one. This class include the theoretically grounded conservative policy iteration (CPI) algorithm (Kakade and Langford, 2002), as well as state-of-the-art deep RL algorithms, such as trust-region policy optimization (TRPO) (Schulman et al., 2015a) and proximal policy optimization (PPO) (Schulman et al., 2017). The main difference between these algorithms is in the way that they enforce the trust-region constraint. TRPO enforces it explicitly through a line-search procedure that ensures the new policy is selected such that its KL-divergence with the old one is below a certain threshold. PPO takes a more relaxed approach and updates its policies by solving an unconstrained optimization problem in which the ratio of the new to old policies is clipped to remain bounded. It has been shown that this procedure does not prevent the policy ratios to go out of bound, and only reduces its probability (Wang et al., 2019; Engstrom et al., 2020).

Mirror descent (MD) (Blair, 1985; Beck and Teboulle, 2003) is a first-order optimization method for solving constrained convex problems. Although MD has been studied for a while and its theory is relatively well-understood in optimization (Beck, 2017; Hazan, 2019), only recently, it has been investigated for policy optimization in RL (Neu et al., 2017; Geist et al., 2019; Liu et al., 2019; Shani et al., 2020). These results showed that despite the fact that the value function is not convex w.r.t. the policy, it is still possible to solve the trust-region problem in closed-form in *tabular* RL. They also

Preprint. Under review.

provided theoretical guarantees, consistent to those obtained in convex optimization, for the resulting tabular RL algorithms.

In this paper, motivated by the theory of MD in *tabular* RL, we study deriving scaleable and practical RL algorithms from the MD principles. Going beyond the tabular case, when the policy belongs to a parametric class, the trust-region problems for policy update in RL cannot be solved in closed-form. We propose an algorithm, called ***mirror descent policy optimization*** (MDPO), that addresses this issue by *approximately* solving these trust-region problems via taking multiple gradient steps on their objective functions. We derive *on-policy* and *off-policy* variants of MDPO (Section 4). We highlight the connection between on-policy MDPO and TRPO and PPO (Section 4.1), and empirically compare it against these algorithms (Section 5.3). Our results on several continuous control tasks from OpenAI Gym (Brockman et al., 2016) show that on-policy MDPO performs better than or on par with TRPO and better than PPO across all these tasks. We then show that if we define the trust-region w.r.t. the *uniform policy*, instead of the old one, our off-policy MDPO coincides with the popular soft actor-critic (SAC) algorithm (Haarnoja et al., 2018). We discuss this connection in detail (Section 4.2) and empirically compare these algorithms, showing that off-policy MDPO performs better than or on par with SAC in the same continuous control problems (Section 5.4).

It is important to note that our above conclusions on the comparison between the MDPO algorithms and TRPO, PPO, and SAC are the result of extensive empirical studies in which we compare different versions of these algorithms against each other (Section 5 and Appendices C and D). In particular, we first compare the vanilla versions of these algorithms in order to better understand how the core of these algorithms work relative to each other. We then add a number of code-level optimization techniques derived from the code-bases of TRPO, PPO, and SAC to these algorithms to compare their best form (those that obtain the best results reported in the literature) against each other.

We conclude the introduction by summarizing our main algorithmic contributions and important observations elicited from our experiments:

1. We address the common belief in the community that explicitly enforcing the trust-region constraint is a necessity for good performance in TRPO, by showing that MDPO, a trust-region method based on the MD principles, does not require enforcing a hard constraint and achieves strong performance by solely solving an unconstrained problem.

2. We show how SAC, a popular off-policy method, can be derived by slight modifications to off-policy MDPO. This provides an optimization perspective for SAC, instead of its initial motivation as an entropy-regularized *(soft)* approximate policy iteration algorithm.

3. We address another common belief— that PPO is a better performing algorithm than TRPO. By reporting results of both the vanilla version and the version loaded with code-level optimization techniques for all algorithms, we show that in both cases, TRPO consistently performs better than PPO. This is in line with some of the findings from a recent case study on PPO and TRPO by Engstrom et al. (2020).

4. Finally, MDPO offers a coherent view of a number of popular policy optimization algorithms in RL, by unifying them under a single theoretical construct (that of MD). Through our on-policy and off-policy experiments, we show how MDPO manifests as a fundamental algorithm with high practical utility which achieves state-of-the-art performance across a number of benchmark tasks.

2 Preliminaries

In this paper, we assume that the agent's interaction with the environment is modeled as a discrete time γ-discounted Markov decision process (MDP), denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma, \mu)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces; $P \equiv P(s'|s,a)$ is the transition kernel; $R \equiv r(s,a)$ is the reward function with the maximum value of R_{max}; $\gamma \in (0,1)$ is the discount factor; and μ is the initial state distribution. Let $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ be a stationary Markovian policy, where $\Delta_{\mathcal{A}}$ is the set of probability distributions on $\mathcal{A}$. The discounted frequency of visiting a state s by following a policy π is defined as $\rho_\pi(s) \equiv (1-\gamma)\mathbb{E}[\sum_{t\geq 0}\gamma^t \mathbb{I}\{s_t = s\} \mid \mu, \pi]$. The value function of a policy π at a state $s \in \mathcal{S}$ is defined as $V^\pi(s) \equiv \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,a_t)|s_0 = s, \pi]$. Similarly, the action-value function of π is defined as $Q^\pi(s,a) = \mathbb{E}[\sum_{t\geq 0}\gamma^t r(s_t,a_t)|s_0 = s, a_0 = a, \pi]$. The difference between the action-value Q and value V functions is referred to as the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

Since finding an optimal policy for an MDP involves solving a non-linear system of equations and the optimal policy may be deterministic (less explorative), many researchers have proposed to add a regularizer in the form of an entropy term to the reward function, and then solve the entropy-regularized (or *soft*) MDP (e.g., Kappen 2005; Todorov 2006; Neu et al. 2017). In this formulation, the reward function is modified as $r_\lambda(s,a) = r(s,a) + \lambda H(\pi(\cdot|s))$, where λ is the regularization parameter and H is an entropy-related term, such as Shannon entropy (Fox et al., 2016; Nachum et al., 2017b), Tsallis entropy (Lee et al., 2018; Nachum et al., 2018), or relative entropy (Azar et al., 2012; Nachum et al., 2017a). Setting $\lambda = 0$, we return to the original formulation, also referred to as the *hard* MDP. In what follows, we use the terms 'regularized' and 'soft' interchangeably.

2.1 Mirror Descent in Convex Optimization

Mirror Descent (MD) (Beck and Teboulle, 2003) is a first-order trust-region optimization method for solving constrained convex problems, i.e.,

$$x^* \in \arg\min_{x \in C} \; f(x), \tag{1}$$

where f is a convex function and the constraint set C is convex compact. In each iteration, MD minimizes a sum of two terms: **1)** a linear approximation of the objective function f at the previous estimate x_k, and **2)** a proximity term that measures the distance between the updated x_{k+1} and previous x_k estimates. MD is considered a trust-region method, since the proximity term keeps the updates x_k and x_{k+1} close to each other. We may write the MD update as

$$x_{k+1} \in \arg\min_{x \in C} \; \langle \nabla f(x_k), x - x_k \rangle + \frac{1}{t_k} B_\psi(x, x_k), \tag{2}$$

where $B_\psi(x, x_k) := \psi(x) - \psi(x_k) - \langle \nabla \psi(x_k), x - x_k \rangle$ is the Bregman divergence associated with a strongly convex function ψ, and t_k is a step-size that is determined by the analysis of MD. When $\psi = \frac{1}{2}\|\cdot\|_2^2$, the Bergman divergence is the Euclidean norm $B_\psi(x, x_k) = \frac{1}{2}\|x - x_k\|_2^2$, and (2) becomes the *projected gradient descent* algorithm (Beck, 2017). When ψ is the negative Shannon entropy, the Bregman divergence term takes the form of the KL divergence, i.e., $B_\psi(x, x_k) = \mathrm{KL}(x, x_k)$. In this case, when the constraint set C is the unit simplex, $C = \Delta_\mathcal{X}$, MD becomes the *exponentiated gradient descent* algorithm and (2) has the following closed form (Beck and Teboulle, 2003):

$$x_{k+1}^i = \frac{x_k^i \exp\big(-t_k \nabla_i f(x_k)\big)}{\sum_{j=1}^n x_k^j \exp\big(-t_k \nabla_j f(x_k)\big)}, \tag{3}$$

where x_k^i and $\nabla_i f$ are the i^{th} coordinates of x_k and ∇f.

3 Mirror Descent in RL

The goal in RL is to find an optimal policy π^*. Two common notions of optimality, and as a result, two distinct ways to formulate RL as an optimization problem are as follows:

$$\pi^*(\cdot|s) \in \arg\max_\pi \; V^\pi(s), \quad \forall s \in \mathcal{S}, \tag{4}$$

$$\pi^* \in \arg\max_\pi \; \mathbb{E}_{s\sim\mu}\big[V^\pi(s)\big]. \tag{5}$$

In (4), the value function is optimized over the entire state space $\mathcal{S}$. This formulation is mainly used in value function based RL algorithms. On the other hand, the formulation in (5) is more common in policy optimization algorithms in which a scalar that is the value function at the initial state ($s \sim \mu$) is optimized.

Unlike the MD optimization problem in (1), the objective function is not convex in π in either of the above two RL optimization problems (4) and (5). Despite this issue, Geist et al. (2019) and Shani et al. (2020) have shown that we can still use the general MD update rule (2) and derive MD-style RL algorithms with the update rules

$$\pi_{k+1}(\cdot|s) \leftarrow \arg\max_{\pi\in\Pi} \; \mathbb{E}_{a\sim\pi}\big[A^{\pi_k}(s,a)\big] - \frac{1}{t_k}\mathrm{KL}(s;\pi,\pi_k), \quad \forall s \in \mathcal{S}, \tag{6}$$

$$\pi_{k+1} \leftarrow \arg\max_{\pi\in\Pi} \; \mathbb{E}_{s\sim\rho_{\pi_k}}\Big[\mathbb{E}_{a\sim\pi}\big[A^{\pi_k}(s,a)\big] - \frac{1}{t_k}\mathrm{KL}(s;\pi,\pi_k)\Big]. \tag{7}$$

for the optimization problems (4) and (5), respectively. Note that while in (6), the policy is optimized uniformly over the state space $\mathcal{S}$, in (7), it is optimized over the measure ρ_{π_k}, i.e., the state frequency induced by the current policy π_k.

Geist et al. (2019) and Shani et al. (2020) proved $\widetilde{\mathcal{O}}(1/\sqrt{K})$ convergence rate to the global optimum for these MD-style RL algorithms in the tabular case, where K is the total number of iterations. They also showed that this rate can be improved to $\widetilde{\mathcal{O}}(1/K)$ in soft MDPs.[1] These results are important and promising, because they provide the same rates as for MD in convex optimization (Beck, 2017).

4 Mirror Descent Policy Optimization

In this section, we derive *on-policy* and *off-policy* RL algorithms based on the MD-style update rules (6) and (7). We refer to our algorithms as *mirror descent policy optimization* (MDPO). Since the trust-region optimization problems in the update rules (6) and (7) cannot be solved in closed-form, we approximate these updates with multiple steps of stochastic gradient descent (SGD) on the objective functions of these optimization problems. In our *on-policy* MDPO algorithm, described in Section 4.1, we use the update rule (7) and compute the SGD updates using the Monte-Carlo (MC) estimate of the advantage function A^{π_k} gathered by following the current policy π_k. On the other hand, our *off-policy* MDPO algorithm, described in Section 4.2, is based on the update rule (6) and calculates the SGD update by estimating A^{π_k} using samples from a replay buffer.

In our MDPO algorithms, we define the policy space, Π, as a class of smoothly parameterized stochastic polices, i.e., $\Pi = \{\pi(\cdot|s;\theta) : s \in \mathcal{S}, \theta \in \Theta\}$. We refer to θ as the policy parameter. We will use π and θ to represent a policy, and Π and Θ to represent the policy space, interchangeably.

4.1 On-Policy MDPO

In this section, we derive an on-policy RL algorithm based on the MD-based update rule (7), whose pseudo-code is shown in Algorithm 1. We refer to this algorithm as *on-policy* MDPO. We may write the update rule (7) for the policy space Θ (defined above) as

$$\theta_{k+1} \leftarrow \arg\max_{\theta\in\Theta} \; \Psi(\theta,\theta_k), \qquad \textit{where} \tag{8}$$

$$\Psi(\theta,\theta_k) = \mathbb{E}_{s\sim\rho_{\theta_k}}\Big[\mathbb{E}_{a\sim\pi_\theta}\big[A^{\theta_k}(s,a)\big] - \frac{1}{t_k}\mathrm{KL}(s;\pi_\theta,\pi_{\theta_k})\Big].$$

Each policy update in (8) requires solving a constraint (over the policy space Θ) optimization problem. In on-policy MDPO (Alg. 1), instead of solving this constraint optimization problem, we update the policy by performing multiple SGD steps on the objective function $\Psi(\theta,\theta_k)$. Interestingly, performing only a single SGD step on $\Psi(\theta,\theta_k)$ is not sufficient as $\nabla_\theta \mathrm{KL}(\cdot;\pi_\theta,\pi_{\theta_k})|_{\theta=\theta_k} = 0$, and thus, if we perform a single-step SGD, i.e.,

$$\nabla_\theta \Psi(\theta,\theta_k)|_{\theta=\theta_k} = \mathbb{E}_{\substack{s\sim\rho_{\theta_k}\\ a\sim\pi_\theta}}\big[\nabla \log \pi_{\theta_k}(a|s) A^{\theta_k}(s,a)\big],$$

the resulting algorithm would be equivalent to vanilla policy gradient and misses the entire purpose of enforcing the trust-region constraint. As a result, the policy update at each iteration k of on-policy MDPO involves m SGD steps as

[1]Note that in this case, the convergence is to the global optimum of the *soft* MDP (a biased solution).

$$\theta_k^{(0)} = \theta_k, \qquad \text{for} \quad i = 0, \dots, m-1$$
$$\theta_k^{(i+1)} \leftarrow \theta_k^{(i)} + \eta \nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k^{(i)}}, \qquad \theta_{k+1} = \theta_k^{(m)},$$

where the gradient of the objective function

$$\nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k^{(i)}} = \mathbb{E}_{\substack{s\sim\rho_{\theta_k} \\ a\sim\pi_{\theta_k}}}\Big[\frac{\pi_{\theta_k}^{(i)}}{\pi_{\theta_k}} \nabla \log \pi_{\theta_k^{(i)}}(a|s) A^{\theta_k}(s,a)\Big] - \frac{1}{t_k}\mathbb{E}_{s\sim\rho_{\theta_k}}\Big[\nabla_\theta \text{KL}(s; \pi_\theta, \pi_{\theta_k})|_{\theta=\theta_k^{(i)}}\Big] \tag{9}$$

can be estimated in an *on-policy* fashion using the data generated by the current policy π_{θ_k}. Since in practice, the policy space is often selected as Gaussian, we use the closed-form of KL in this estimation.

Algorithm 1 On-Policy MDPO

```
 1: Initialize Value network $V_\phi$; Policy networks $\pi_{\text{new}}$ and $\pi_{\text{old}}$;
 2: for $k = 1, \dots, K$ do
 3:     # On-policy Data Generation
 4:     Simulate the current policy $\pi_{\theta_k}$ for $M$ steps;
 5:     for $t = 1, \dots, M$ do
 6:         Calculate the return $R_t := R(s_t, a_t) = \sum_{j=t}^{M} \gamma^{j-t} r_j$;
 7:         Estimate the advantage $A(s_t, a_t) = R(s_t, a_t) - V_\phi(s_t)$;
 8:     end for
 9:     # Policy Improvement   (policy update)   (Actor Update)
10:     $\theta_k^{(0)} = \theta_k$;
11:     for $i = 0, \dots, m-1$ do
12:         $\theta_k^{(i+1)} \leftarrow \theta_k^{(i)} + \eta \nabla_\theta \Psi(\theta, \theta_k)|_{\theta=\theta_k^{(i)}}$;          (Eq. 9)
13:     end for
14:     $\theta_{k+1} = \theta_k^{(m)}$;
15:     # Policy Evaluation   (value update)   (Critic Update)
16:     Update $\phi$ by minimizing the $N$-minibatch ($N \le M$) loss function   $L_{V_\phi} = \frac{1}{N}\sum_{t=1}^{N}\big[V_\phi(s_t) - R_t\big]^2$;
17: end for
```

Our on-policy MDPO algorithm (Alg. 1) has close connection to two popular on-policy trust-region RL algorithms: trust-region policy optimization (TRPO) (Schulman et al., 2015a) and proximal policy optimization (PPO) (Schulman et al., 2017). We now discuss the similarities and differences between on-policy MDPO and these algorithms.

Comparison with TRPO: At each iteration k, TRPO considers the constrained optimization problem

$$\max_{\theta\in\Theta} \; \mathbb{E}_{\substack{s\sim\rho_{\theta_k} \\ a\sim\pi_{\theta_k}}}\Big[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\theta_k}(s,a)\Big], \qquad \text{s.t.} \quad \mathbb{E}_{s\sim\rho_{\theta_k}}\big[\text{KL}(s; \pi_{\theta_k}, \pi_\theta)\big] \le \delta, \tag{10}$$

and updates its policy parameter by taking a step in the direction of the *natural gradient* of the objective function in (10) as

$$\theta_{k+1} \leftarrow \theta_k + \eta F^{-1} \mathbb{E}_{\substack{s\sim\rho_{\theta_k} \\ a\sim\pi_{\theta_k}}}\big[\nabla \log \pi_{\theta_k}(a|s) A^{\theta_k}(s,a)\big],$$

where $F = \mathbb{E}_{\substack{s\sim\rho_{\theta_k} \\ a\sim\pi_{\theta_k}}}\big[\nabla \log \pi_{\theta_k}(a|s) \nabla \log \pi_{\theta_k}(a|s)^\top\big]$ is the *Fisher information matrix* for the current policy π_{θ_k}. It then explicitly enforces the trust-region constraint in (10) by computing the KL-term for $\theta = \theta_{k+1}$ and checking if it is larger than the threshold δ. If this is the case, the step size is reduced until the constraint is satisfied.

In comparison to TRPO, **1)** On-policy MDPO does not explicitly enforce the trust-region constraint, but approximately satisfies it by performing multiple steps of SGD on the objective function of the optimization problem in the MD-style update rule (8). We say "*it approximately satisfies the constraint*" because instead of fully solving (8), it takes multiple steps in the direction of the gradient of its objective function. **2)** On-policy MDPO uses simple SGD instead of natural gradient, and thus, does not have to deal with the computational overhead of computing (or approximating) the inverse of the Fisher information matrix.[2] **3)** The direction of KL in on-policy MDPO, $\text{KL}(\pi, \pi_k)$, is consistent with that in the MD update rule in convex optimization and is different than that in TRPO, $\text{KL}(\pi_k, \pi)$. This does not cause any problem (sampling issue) for either algorithm, as both calculate the KL-term in closed-form (Gaussian policies). **4)** While TRPO uses multiple heuristics to define the step-size and to reduce it in case the trust-region constraint is violated, on-policy MDPO uses a simple schedule for the step-size, motivated by the theory of MD (Beck and Teboulle, 2003), and sets $t_k = K/k$, where K is the maximum number of iterations. This way it anneals the step-size $1/t_k$ from 1 to 0 over the iterations of the algorithm.

Comparison with PPO: At each iteration k, PPO performs multiple steps of SGD on the objective function of the following unconstrained optimization problem:

$$\max_{\theta\in\Theta} \mathbb{E}_{\substack{s\sim\rho_{\theta_k} \\ a\sim\pi_{\theta_k}}} \Big[\min\Big\{ \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\theta_k}(s,a),\ \text{clip}\big(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\big) A^{\theta_k}(s,a) \Big\} \Big], \tag{11}$$

in which the hyper-parameter ϵ determines how the policy ratio, $\pi_\theta/\pi_{\theta_k}$, is clipped. It is easy to see that the gradient of the objective function in (11) is zero for the state-action pairs at which the policy ratio is clipped and is non-zero, otherwise. However, since the gradient is averaged over all the state-action pairs in the batch, the policy is updated even if its ratio is out of bound for some state-action pairs. This phenomenon, which has been reported by Wang et al. (2019) and Engstrom et al. (2020), shows that clipping in PPO does not prevent the policy ratios to go out of bound, but it only reduces its probability. This means that despite using clipping, PPO does not guarantee that the trust-region constraint is always satisfied. In fact, recent results, including those in Engstrom et al. (2020) and our experiments in this paper (see Section 5.3), show that most of the improved performance exhibited by PPO is due to code-level optimization techniques, such as learning rate annealing, observation and reward normalization, and in particular, the use of generalized advantage estimation (GAE) (Schulman et al., 2015b). We delve deeper into this in Appendix A. Although both on-policy MDPO and PPO take multiple SGD steps on the objective function of unconstrained optimization problems (8) and (11), respectively, the way they handle the trust-region constraint is completely different.

Another interesting observation is that the adaptive and fixed KL algorithms (we refer to as KL-PPO here), proposed in the PPO paper (Schulman et al., 2017), have policy update rules similar to on-policy MDPO. However, these algorithms have not been used much in practice, because it was shown in the same paper that they perform much worse than the PPO algorithm. Despite the similarities, there are two main differences between the update rules of KL-PPO and on-policy MDPO. **First,** there is no explicit mention of *multiple* SGD updates in KL-PPO. As discussed earlier in this section, a single-step SGD update in on-policy MDPO is equivalent to vanilla policy gradient, since the gradient of the KL-term is zero, i.e., $\nabla_\theta \text{KL}(\cdot;\pi_\theta,\pi_{\theta_k})|_{\theta=\theta_k} = 0$. **Second,** similar to TRPO, the direction of KL in KL-PPO, $\text{KL}(\pi_k, \pi)$, is different than that in on-policy MDPO, $\text{KL}(\pi, \pi_k)$, which is consistent with that in the MD literature. Since in our experiments, on-policy MDPO performs significantly better than PPO (see Section 5.3), we conjecture that either a single-step SGD update or the direction of KL is the reason for the inferior performance of KL-PPO, compared to PPO, as reported in Schulman et al. (2017).

4.2 Off-Policy MDPO

In this section, we derive an off-policy RL algorithm based on the MD update rule (6). Algorithm 2 contains the pseudo-code of this algorithm, which we refer to as off-policy MDPO. To emulate the

[2]TRPO does not explicitly invert F, but instead, approximates the natural gradient update using conjugate gradient descent.

uniform sampling over the state space required by (6), Algorithm 2 samples a batch of states from a replay buffer $\mathcal{D}$ (Line 4). While this sampling scheme is not truly uniform, it makes the update less dependent on the current policy. Similar to the on-policy case, we may write the update rule (6) for the policy class Θ as

$$\theta_{k+1} \leftarrow \arg\max_{\theta\in\Theta} \; \Psi(\theta,\theta_k), \qquad \textit{where} \tag{12}$$
$$\Psi(\theta,\theta_k) = \mathbb{E}_{s\sim\mathcal{D}}\Big[\mathbb{E}_{a\sim\pi_\theta}\big[A^{\theta_k}(s,a)\big] - \frac{1}{t_k}\text{KL}(s;\pi_\theta,\pi_{\theta_k})\Big].$$

The main idea in Alg. 2 is to estimate the advantage or action-value function of the current policy, A^{θ_k} or Q^{θ_k}, in an off-policy fashion, using a batch of data randomly sampled from the replay buffer $\mathcal{D}$. In a similar manner to the policy update of our on-policy MDPO algorithm (Alg. 1), described in Section 4.1, we then update the policy by taking multiple SGD steps on the objective function $\Psi(\theta,\theta_k)$ of the optimization problem (12) (by keeping θ_k fixed). Algorithm 2 uses two neural networks V_ϕ and Q_ψ to estimate the value and action-value functions of the current policy. It then uses them to estimate the advantage function, i.e., $A^{\theta_k} \approx Q_\psi^{\theta_k} - V_\phi^{\theta_k}$. The Q_ψ update is done in a TD(0) fashion, while the V_ϕ update involves fitting the value function to the Q_ψ estimate of the current policy. For policy update, Alg. 2 uses the reparameterization trick and replaces the objective function in (12) with the following loss:

$$L(\theta,\theta_k) = \mathbb{E}_{s\sim\mathcal{D}\epsilon\sim\mathcal{N}}\big[\log\pi_\theta\big(\widetilde{a}_\theta(\epsilon,s)|s\big) - \log\pi_{\theta_k}\big(\widetilde{a}_\theta(\epsilon,s)|s\big) - t_k Q_\psi^{\theta_k}\big(s,\widetilde{a}_\theta(\epsilon,s)\big)\big], \tag{13}$$

where $\widetilde{a}_\theta(\epsilon,s)$ is the action generated by sampling the ϵ noise from a zero-mean normal distribution $\mathcal{N}$. We can easily modify Algorithm 2 to optimize *soft* (entropy regularized) MDPs. In this case, the V_ϕ update (Line 12 of Alg. 2) remains unchanged, while in the Q_ψ update the reward function is replaced by its soft version $r_\lambda(s,a) = r(s,a) - \lambda\log\pi_\theta(a|s)$. The loss function (13) used for policy update is also modified, Q_ψ becomes the soft Q-function and a term $\lambda t_k \log\pi_{\theta_k}\big(\widetilde{a}_\theta(\epsilon,s)|s\big)$ is added inside the expectation.

Algorithm 2 Off-Policy MDPO

```
1: Initialize Replay buffer $\mathcal{D}=\emptyset$; Value networks $V_\phi$ and $Q_\psi$; Policy networks $\pi_{\text{new}}$ and $\pi_{\text{old}}$;
2: for $k=1,\dots,K$ do
3:     Take action $a_k \sim \pi_{\theta_k}(\cdot|s_k)$, observe $r_k$ and $s_{k+1}$, and add $(s_k,a_k,r_k,s_{k+1})$ to the replay buffer $\mathcal{D}$;
4:     Sample a batch $\{(s_j,a_j,r_j,s_{j+1})\}_{j=1}^N$ from $\mathcal{D}$;
5:     # Policy Improvement  (policy update)  (Actor Update)
6:     $\theta_k^{(0)} = \theta_k$;
7:     for $i=0,\dots,m-1$ do
8:         $\theta_k^{(i+1)} \leftarrow \theta_k^{(i)} + \eta\nabla_\theta L(\theta,\theta_k)|_{\theta=\theta_k^{(i)}}$;
9:     end for
10:    $\theta_{k+1} = \theta_k^{(m)}$;
11:    # Policy Evaluation  (Q and V updates)  (Critic Update)
12:    Update $\phi$ and $\psi$ by minimizing the loss functions
       $L_{V_\phi} = \frac{1}{N}\sum_{j=1}^N \big[V_\phi(s_j) - Q_\psi\big(s_j,\pi_{\theta_{k+1}}(s_j)\big)\big]^2$;
       $L_{Q_\psi} = \frac{1}{N}\sum_{j=1}^N \big[r(s_j,a_j) + \gamma V_\phi(s_{j+1}) - Q_\psi(s_j,a_j)\big]^2$;
13: end for
```

Similarly to on-policy MDPO that has close connection to TRPO and PPO, discussed in Section 4.1, off-policy MDPO (Alg. 2) is related to the popular soft actor-critic (SAC) algorithm (Haarnoja et al., 2018). We now derive SAC by slight modifications in the derivation of off-policy MDPO. This gives an optimization interpretation of SAC, which we then use to show strong ties between the two algorithms.

Comparison with SAC: Soft actor-critic is an approximate policy iteration algorithm in soft MDPs. At each iteration k, it first estimates the (soft) Q-function of the current policy, Q^{π_k}, and then set the next policy to the (soft) greedy policy w.r.t. the estimated Q-function as

$$\pi_{k+1}(a|s) \leftarrow \exp\big(Q^{\pi_k}(s,a)\big) \,/\, Z^{\text{SAC}}(s), \tag{14}$$

where $Z^{\text{SAC}}(s) = \mathbb{E}_{a\sim\pi_k(\cdot|s)}\big[\exp\big(Q^{\pi_k}(s,a)\big)\big]$ is a normalization term. However, since tractable policies are preferred in practice, SAC suggests to project the improved policy back into the policy space considered by the algorithm, using the following optimization problem:

$$\theta_{k+1} \leftarrow \arg\min_{\theta\in\Theta}\ \mathcal{L}^{\text{SAC}}(\theta,\theta_k),$$
$$\mathcal{L}^{\text{SAC}}(\theta,\theta_k) = \mathbb{E}_{s\sim\mathcal{D}}\Big[\text{KL}\big(s;\pi_\theta,\frac{\exp\big(Q^{\theta_k}(s,\cdot)\big)}{Z^{\text{SAC}}(s)}\big)\Big]. \tag{15}$$

This update rule computes the next policy as the one with the minimum KL-divergence to the term on the RHS of (14)

Since the optimization problem in (15) is invariant to the normalization term, unlike (14), the policy update (15) does not need to compute $Z^{\text{SAC}}(s)$. By writing the KL definition and using the reparameterization trick in (15), SAC updates its policy by minimizing the following loss function:

$$L^{\text{SAC}}(\theta,\theta_k) = \mathbb{E}_{\substack{s\sim\mathcal{D}\\ \epsilon\sim\mathcal{N}}}\big[\lambda\log\pi_\theta\big(\widetilde{a}_\theta(\epsilon,s)|s\big) - Q_\psi^{\theta_k}\big(s,\widetilde{a}_\theta(\epsilon,s)\big)\big]. \tag{16}$$

Comparing the loss function (16) with the one used in the policy update of off-policy MDPO (13), we notice that despite the similarities, the main difference is the absence of the current policy, π_{θ_k}, in the SAC's loss function. To explain the relationship between off-policy MDPO and SAC, recall from Section 2.1 that if the constraint set is the unit simplex, i.e., $C = \Delta_{\mathcal{X}}$, the MD update has a closed-from shown in (3). Thus, if the policy class (constraint set) Π in the update rule (6) is the entire space of stochastic policies, then we may write (6) in closed-form as (see e.g., Mei et al., 2019; Shani et al., 2020)

$$\pi_{k+1}(a|s) \leftarrow \pi_k(a|s)\exp\big(t_kQ^{\pi_k}(s,a)\big)\ /\ Z(s), \tag{17}$$

where $Z(s) = \mathbb{E}_{a\sim\pi_k(\cdot|s)}\big[\exp\big(t_kQ^{\pi_k}(s,a)\big)\big]$ is a normalization term. The closed-form solution (17) is equivalent to solving the constrained optimization problem (6) in two phases (see Hazan, 2019): **1)** solving the *unconstrained* version of (6) that leads to the numerator of (17), followed by **2)** *projecting* this (unconstrained) solution back into the constrained set (all stochastic policies) using the same choice of Bregman divergence (KL in our case), which accounts for the normalization term in (17). Hence, when we optimize over the parameterized policy space Θ (instead of all stochastic policies), the MD update would be equivalent to finding a policy $\theta\in\Theta$ with minimum KL-divergence to the solution of the *unconstrained* optimization problem obtained in the first phase (the numerator of (17)). This leads to the following policy update rule:

$$\theta_{k+1} \leftarrow \arg\min_{\theta\in\Theta}\ \mathcal{L}(\theta,\theta_k),$$
$$\mathcal{L}(\theta,\theta_k) = \mathbb{E}_{s\sim\mathcal{D}}\Big[\text{KL}\big(s;\pi_\theta,\pi_{\theta_k}\exp(t_kQ^{\theta_k})\big)\Big]. \tag{18}$$

If we write the definition of KL and use the reparameterization trick in (18), we will rederive the loss function (13) used by our off-policy MDPO algorithm.[3] Note that both SAC (15) and off-policy MDPO (18) use KL projection to project back to the set of policies. For SAC, the authors argue that any projection can be chosen arbitrarily. However, our derivation clearly shows that the selection of KL projection is dictated by the choice of the Bregman divergence.

As mentioned earlier, the main difference between the loss functions used in the policy updates of SAC (16) and off-policy MDPO (13) is the absence of the current policy, π_{θ_k}, in the SAC's loss function.[4] The current policy, π_{θ_k}, appears in the policy update of off-policy MDPO, because it is

[3]In soft MDPs, Q^{π_k} is replaced by its soft version and a term $\lambda t_k\log\pi_{\theta_k}(a|s)$ is added to the exponential in the closed-form solution (17). This will result in the same changes in (18). Similar to the hard case, applying the reparameterization trick to the the soft version of (18) gives us the soft version of the loss function (13).

[4]The same difference can also be seen in the policy updates (14) and (17) of SAC and off-policy MDPO, respectively.

a trust-region algorithm, and thus, tries to keep the new policy close to the old one. On the other hand, following the original SAC interpretation as an approximate policy iteration algorithm, its policy update does not contain a term to keep the new and old policies close to each other. It is interesting to note that SAC's loss function can be re-obtained by repeating the derivation which leads to off-policy MDPO, and replacing the current policy, π_{θ_k}, with the *uniform policy* in the objective (12) of off-policy MDPO. Therefore, SAC can be considered as a trust-region algorithm w.r.t. the uniform policy (or an entropy regularized algorithm). This means its update encourages the new policy to remain *explorative*, by keeping it close to the uniform policy.

SAC as FTRL: The above perspective suggests that both off-policy MDPO and SAC use similar regularization techniques to optimize the policy. This view allows us to regard the *hard* version of SAC as an RL implementation of the *follow-the-regularized-leader* (FTRL) algorithm. Thus, the equivalency between MD and FTRL (see e.g., Hazan, 2019) suggests that off-policy MDPO and SAC are in fact closely related and somewhat equivalent. This is further confirmed by our experimental results reported in Section 5.4.

Reverse vs. Forward KL Direction: Similar to the on-policy case, the *mode-seeking* or *reverse* direction of the KL term (in Eq. 18) in off-policy MDPO is consistent with that in the MD update rule in convex optimization. With this direction of KL, the optimization problems for policy update in both off-policy MDPO and SAC are invariant to the normalization term $Z(s)$. Thus, these algorithms can update their policies without computing $Z(s)$. Mei et al. (2019) proposed an algorithm, called *exploratory conservative policy optimization* (ECPO), that resembles our *soft* off-policy MDPO, except in the direction of KL. Switching the direction of KL to *mean-seeking* or *forward* has the extra overhead of estimating the normalization term for ECPO. However, Mei et al. (2019) argue that it results in better performance. They empirically show that ECPO performs better than several algorithms, including one that is close to our off-policy MDPO. They refer to this algorithm as *policy mirror descent* (PMD) and report poor performance for it. We did not use their code-base and exact configuration, but we did not observe such poor performance for our off-policy MDPO algorithm. In fact, our experimental results in Sec. 5.4 show that off-policy MDPO performs better than or on-par with SAC in six commonly used MuJoCo domains. More experiments and further investigation are definitely required to better understand the effect of the KL direction in MDPO algorithms.

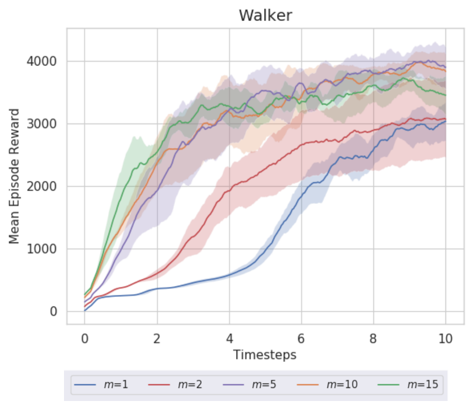
5 Experimental Results

In this section, we empirically evaluate on-policy and off-policy MDPO algorithms on a number of continuous control tasks from OpenAI Gym (Brockman et al., 2016), and compare them with state-of-the-art baselines: TRPO, PPO, and SAC. We report all experimental details, including the hyper-parameter values used by the algorithms in the on-policy and off-policy experiments in Tables 2 and 3 in Appendix A. In our tabular results, both in the main paper and in Appendices C and D, we report the final training scores averaged over 5 runs and their 95% confidence intervals (CI). We bold all the best values that have overlapping CIs.

For off-policy MDPO, we consider two ways of defining the Bregman divergence in (2) using: **1)** Shannon entropy, which results in the KL version, as described in Section 4.2, and **2)** Tsallis entropy, which results in a Tsallis-induced Bregman version. We only report the results for the Tsallis-based choice and refer the reader to Appendix B for its complete description and detailed derivation.

5.1 On Multiple SGD Steps

In on-policy MDPO, we implement the multi-step update at each MD iteration of the algorithm, by sampling M trajectories from the current policy, generating estimates of the advantage function, and performing m gradient steps using the same set of trajectories. We evaluated on-policy MDPO for different values of m in all tasks. We show the results for Walker2d in Figure 1 (top). The results for all tasks show a clear trade-off between m and the performance. Moreover, $m = 10$ seems to be the best value for most tasks. This is why we use $m = 10$ in all our on-policy MDPO experiments. Our results clearly indicate that using $m = 1$ leads to inferior performance as compared to $m = 10$, reaffirming the theory that suggests solving the trust-region problem in RL requires taking several gradient steps at each MD iteration. Finally, we ran a similar experiment for TRPO which shows that performing multiple gradient steps at each iteration of TRPO does not lead to any improvement. In



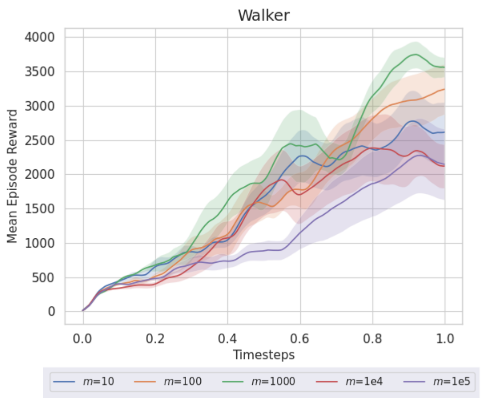


Figure 1: Performance of on-policy (top) and off-policy (bottom) MDPO (code level optimizations included) for different values of m on the Walker2d task. X-axis represents time steps in millions.

fact our results showed that in some cases, it even leads to worse performance than when performing a single-step update.

For off-policy MDPO, performing multiple SGD steps at each MD iteration (Lines 6 to 10 in Alg. 2) becomes increasingly time-consuming as the value of m grows. Therefore, as a proxy to this method, we resort to staying close to an m step old copy of the current policy, while performing a single gradient update at each iteration of the algorithm. This copy is updated every m iterations with the parameters of the current policy. We believe our results can be further improved if we do perform m gradient updates at each iteration, but we omit from performing these experiments because of their high wall-clock time. Note that the results reported in Haarnoja et al. (2018) for SAC are also obtained from multiple gradient updates. However, in order to ensure a fair comparison, we use single-step gradient updates for SAC in our experiments. Finally, we evaluated off-policy MDPO for different values of m in all tasks and show the results for Walker2d in Figure 1 (bottom). We found it hard to identify a single best value of m for off-policy MDPO across the tasks. However, $m = 1000$ had the most reasonable performance across the tasks, and thus, we use it in all our off-policy MDPO experiments.

5.2 On Code-level Optimizations

We report the average of final training scores for on-policy MDPO, compared against TRPO and PPO, and off-policy MDPO (both KL and Tsallis versions), compared against SAC in Table 1. The results reported in Table 1 are for the vanilla implementations of these methods. However, there are certain "code-level optimization techniques" in a lot of code-bases of TRPO, PPO, and SAC that result in enhanced performance. Engstrom et al. (2020) provided a case study of these techniques in PPO and TRPO. We provide a detailed description of these techniques in Appendix A, and report the performance of the algorithms when these techniques are added to them in Appendices C and D. Note that the results we report for TRPO, PPO, and SAC with these techniques match their state-of-the-art results in the literature.

Overall, the key takeaway from our results is that MDPO performs significantly better than PPO and on-par or better than TRPO and SAC, while being much simpler to implement, and more general as being derived from the theory of MD in RL. In the next two sections, we report our main observations from our on-policy and off-policy results.

5.3 On-policy Results

We implemented three versions of on-policy MDPO, TRPO, and PPO: **1)** the vanilla version reported both in this section (Table 1) and in Appendix C, **2)** the loaded version in which we add the code-level optimization techniques to these algorithms, and **3)** the loaded version plus generalized advantage estimation (GAE) (Schulman et al., 2015b). We report the results for the second and third cases in Appendix C.

	On-Policy			*Off-Policy*		
	MDPO	TRPO	PPO	MDPO *KL*	MDPO *Tsallis*	SAC
Hopper-v2	**1964 (±217)**	**2382 (±445)**	1281 (±353)	**1385 (±648)**	**1108 (±417)**	**1501 (±414)**
Walker2d-v2	**2948 (±298)**	2454 (±171)	424 (±92)	**873 (±180)**	**1151 (±218)**	635 (±137)
HalfCheetah-v2	**2873 (±835)**	**1726 (±690)**	617 (±135)	8098 (±428)	**8477 (±450)**	**9298 (±371)**
Ant-v2	**1162 (±738)**	**1716 (±338)**	-40 (±33)	1051 (±284)	**2348 (±338)**	378 (±33)
Humanoid-v2	**635 (±46)**	449 (±9)	448 (±56)	2258 (±372)	**4426 (±229)**	3598 (±172)
H.Standup-v2	**127901 (±6217)**	100408 (±12564)	96068 (±11721)	**131702 (±7203)**	**138157 (±8983)**	**142774 (±4864)**

Table 1: Averaged cumulative returns for minimal/vanilla versions of MDPO, TRPO, PPO, and SAC algorithms. The results are averaged over 5 runs, together with their 95% confidence interval. The best values with overlapping CIs are bolded.

We may elicit the followings observations from these results. **First,** on-policy MDPO performs better than or on par with TRPO and better than PPO across all tasks. This contradicts the common belief that explicitly enforcing the constraint (e.g., through line-search) is necessary for achieving good performance in TRPO. **Second,** on-policy MDPO can be implemented more efficiently than TRPO, because it does not require the extra line-search step. Notably, TRPO suffers from scaling issues as it requires computing the correct step-size of the gradient update using a line search, which presents as an incompatible part of the computation graph in popular auto-diff packages such as TensorFlow. Moreover, MDPO performs significantly better than PPO, while remaining equally efficient in terms of implementation. **Third,** TRPO performs better than PPO consistently, both in the vanilla case and when the code-level optimizations (including GAE) are added. This is in contrast to the common belief that PPO is a better performing algorithm than TRPO. Our observation is in line with what Engstrom et al. (2020) noted in their empirical study of these two algorithms, and we believe it further reinforces it. Adding code-level optimizations and GAE improve the performance of PPO, but not enough to outperform TRPO, as TRPO also benefits from these additions. Lastly, **fourth,** Wang et al. (2019) have recently shown that PPO is prone to instability issues. When run for longer than the usual number of time-steps, our experiments show that this is indeed the case as the performance improves until the standard time-step mark 1M, and then decreases in some of the tasks.

5.4 Off-policy Results

Similar to the on-policy case, we implemented both vanilla and loaded versions of off-policy MDPO and PPO. We report the results of the vanilla versions in this section (Table 1) and in Appendix D, while the results of the loaded versions are only reported in Appendix D.

We observe the followings from these results. **First,** off-policy MDPO (both *KL* and *Tsallis* versions) performs better than or on par with SAC across all tasks. **Second,** off-policy MDPO results in a performance increase in most tasks, both in terms of sample efficiency and final performance, in comparison to on-policy MDPO. This is consistent with the common belief about on-policy and off-policy algorithms. Finally, **third,** our off-policy MDPO experiments with regularized (for both KL and Tsallis) and unregularized formulations, reported here and in Appendix D, did not produce a clear winner over all tasks. This is consistent with the conclusions made in Haarnoja et al. (2018, Appendix E) that perhaps entropy regularization is not the main cause for improved performance in SAC.

6 Conclusions

We derived on-policy and off-policy algorithms from the theory of MD in RL. Each policy update in our mirror descent policy optimization (MDPO) algorithms is formulated as a trust-region optimization problem. However, our algorithms do not update their policies by solving these problems, instead, they perform their policy updates by taking multiple gradient steps on the objective function of these problems. We described the relationship between on-policy MDPO and two popular trust-region RL algorithms: TRPO and PPO, and showed that it performs better than or on par with TRPO and better than PPO across a number of continuous control tasks. We then discussed how the popular SAC algorithm can be derived by slight modifications of off-policy MDPO, and showed that off-policy MDPO performs better than or on par with SAC in these tasks. To summarize, our MDPO algorithms derived from the MD principles, offer a unifying view for a number of popular RL algorithms, and can achieve performance better than or equal to state-of-the-art RL algorithms. We can think of several future directions. In addition to evaluating MDPO algorithms in more complex and realistic problems, we would like to see their performance in discrete action problems in comparison with algorithms like DQN. Investigating the use of Bregman divergences other than KL seems to be promising. Our work with Tsallis entropy is in this direction but more algorithmic and empirical work needs to be done. Finally, there are recent theoretical results on incorporating exploration into the MD-based updates. Applying exploration to MDPO could prove most beneficial, especially in complex environments.

References

M. Azar, V. Gómez, and H. Kappen. Dynamic policy programming. *Journal of Machine Learning Research*, 13:3207–3245, 2012.

A. Beck. First-order methods in optimization. *SIAM*, 25, 2017.

A. Beck and M. Teboulle. Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175, 2003.

C. Blair. Problem complexity and method efficiency in optimization (A. Nemirovsky and D. Yudin). *SIAM Review*, 27(2):264, 1985.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *Preprint arXiv:1606.01540*, 2016.

P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI baselines, 2017.

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation matters in deep RL: A case study on PPO and TRPO. In *Proceeding of the 8th International Conference on Learning Representations*, 2020.

R. Fox, A. Pakman, and N. Tishby. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 202–211, 2016.

S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

M. Geist, B. Scherrer, and O. Pietquin. A theory of regularized Markov decision processes. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1861–1870, 2018.

E. Hazan. Introduction to online convex optimization. *arXiv preprint arXiv:1909.05207*, 2019.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 267–274, 2002.

H. Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics*, 11, 2005.

K. Lee, S. Choi, and S. Oh. Sparse Markov decision processes with causal sparse Tsallis entropy regularization for reinforcement learning. *Robotics and Automation Letters*, 2018.

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

B. Liu, Q. Cai, Z. Yang, and Z. Wang. Neural proximal/trust region policy optimization attains globally optimal policy. *Preprint arXiv:1906.10306*, 2019.

J. Mei, C. Xiao, R. Huang, D. Schuurmans, and M. Muller. On principled entropy exploration in policy optimization. In *Proceedings of the 28th Joint Conference on Artificial Intelligence*, pages 3130–3136, 2019.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Trust-PCL: An off-policy trust region method for continuous control. *Preprint arXiv:1707.01891*, 2017a.

O. Nachum, M. Norouzi, K. Xu, and D. Schuurmans. Bridging the gap between value and policy based reinforcement learning. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, pages 2772–2782, 2017b.

O. Nachum, Y. Chow, and M. Ghavamzadeh. Path consistency learning in tsallis entropy regularized mdps. In *Proceedings of the 35th International Conference on Machine Learning*, pages 979–988, 2018.

G. Neu, A. Jonsson, and V. Gómez. A unified view of entropy-regularized markov decision processes. *Preprint arXiv:1705.07798*, 2017.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International conference on machine learning*, pages 1889–1897, 2015a.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *Preprint arXiv:1506.02438*, 2015b.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *Preprint arXiv:1707.06347*, 2017.

L. Shani, Y. Efroni, and S. Mannor. Adaptive trust region policy optimization: Global convergence and faster rates for regularized MDPs. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, 2020.

E. Todorov. Linearly-solvable Markov decision problems. In *Proceedings of the 19th Advances in Neural Information Processing*, pages 1369–1376, 2006.

Y. Wang, H. He, and X. Tan. Truly proximal policy optimization. In *Proceeding of the 35th Conference on Uncertainty in Artificial Intelligence*, 2019.

Appendix

A Experimental Details

A.1 Setup

We evaluate all algorithms on OpenAI Gym (Brockman et al., 2016) based continuous control tasks, including Hopper-v2, Walker2d-v2, HalfCheetah-v2, Ant-v2, Humanoid-v2 and HumanoidStandup-v2. All experiments are run across 5 random seeds. Each plot shows the empirical mean of the random runs while the shaded region represents a 95% confidence interval (*empirical mean* $\pm 1.96\times$ *empirical standard deviation /* $\sqrt{n=5}$). We report results in both figure and tabular forms. The tabular results denote the mean final training performance and the best values with overlapping confidence intervals are bolded.

For all off-policy experiments, we use $\lambda = 0.2$ across all tasks, which is known to be the best performing value for all tasks according to (Haarnoja et al., 2018) (In our experiments, a value of 0.2 worked equally well for Humanoid as the reported 0.05 in the SAC paper). We report all details of our off-policy experiments including hyperparameter values in Table 3. Moreover, since doing multiple gradient steps at each iteration becomes quite time consuming for the off-policy case, we get around this issue by fixing the old policy (π_{θ_k}) for m number of gradient steps, in order to mimic the effect from taking multiple gradients steps at each iteration. This ensures that the total number of environment steps are always equal to the total number of gradients steps, irrespective of the value of m. Finally, for all experiments, we use a fixed Bregman stepsize ($1/t_k$) as opposed to an annealed version like in the on-policy case.

A.2 Code-level Optimization Techniques

The widely available OpenAI Baselines (Dhariwal et al., 2017) based PPO implementation uses the following five major modifications to the original algorithm presented in Schulman et al. (2017) – value function clipping, reward normalization, observation normalization, orthogonal weight initialization and an annealed learning rate schedule for the Adam optimizer. These are referred to as code level optimization techniques (as mentioned in above sections) and are originally noted in (Engstrom et al., 2020). Following the original notation, we refer to the vanilla or minimal version of PPO, i.e. without these modifications as PPO-M. Then, we consider two PPO versions which include all such code level optimizations, with the hyperparameters given in Schulman et al. (2017). One of them does not use GAE while the other version includes GAE. Therefore they are referred to as PPO-LOADED and PPO-LOADED+GAE respectively. These versions, although being far from the theory, have been shown to be the best performing ones, and so form as a good baseline. We do a similar bifurcation for TRPO and on-policy MDPO. We report all details of our on-policy experiments including hyperparameter values in Table 2.

Similarly, for the off-policy MDPO versions, we again restrain from using the optimization tricks mentioned above. However we do employ three techniques that are common in actor-critic based algorithms, namely: using separate Q and V functions as in (Haarnoja et al., 2018), using two Q functions to reduce overestimation bias and using soft target updates for the value function. Prior work (Fujimoto et al., 2018; Lillicrap et al., 2015) has shown these techniques help improve stability.

Similar to the on-policy experiments, we include a minimal and loaded version for the off-policy experiments as well, which are described in Appendix D. In particular, this branching is done based on the neural network and batch sizes used. Since the standard values in all on-policy algorithms is different from the standard values used by most off-policy approaches, we show results for both set of values. This elicits a better comparison between on-policy and off-policy methods.

Hyperparameter	TRPO-M	TRPO-LOADED	PPO-M	PPO-LOADED	MDPO-M	MDPO-LOADED
Adam stepsize	-	-	3×10^{-4}	Annealed from 1 to 0	3×10^{-4}	Annealed from 1 to 0
minibatch size	128	128	64	64	128	128
number of gradient updates (m)	-	-	-	-	5	10
reward normalization	✗	✓	✗	✓	✗	✓
observation normalization	✗	✓	✗	✓	✗	✓
orthogonal weight initialization	✗	✓	✗	✓	✗	✓
value function clipping	✗	✓	✗	✓	✗	✓
horizon (T)	2048					
entropy coefficient	0.0					
discount factor	0.99					
total number of timesteps	10^7					
#runs used for plot averages	5					
confidence interval for plot runs	$\sim 95\%$					

Table 2: Hyperparameters of all on-policy methods.

Hyperparameter	MDPO-M *KL*	MDPO-M *Tsallis*	SAC-M	MDPO-LOADED *KL*	MDPO-LOADED *Tsallis*	SAC-LOADED
number of hidden units per layer	64	64	64	256	256	256
minibatch size	64	64	64	256	256	256
entropy coefficient (λ)	0.2					
Adam stepsize	3×10^{-4}					
reward normalization	✗					
observation normalization	✗					
orthogonal weight initialization	✗					
value function clipping	✗					
replay buffer size	10^6					
target value function smoothing coefficient	0.005					
number of hidden layers	2					
discount factor	0.99					
#runs used for plot averages	5					
confidence interval for plot runs	$\sim 95\%$					

Table 3: Hyper-parameters of all off-policy methods.

	Hopper-v2	Walker2d-v2	HalfCheetah-v2	Ant-v2	Humanoid-v2	HumanoidStandup-v2
Bregman stepsize ($1/t_k$)	0.8	0.4	0.3	0.5	0.5	0.3

Table 4: Bregman stepsize for each domain, used by off-policy MDPO.

B Tsallis-based Bregman Divergence

As described in section 2.1, the MD update contains a Bregman divergence term. A Bregman divergence is a measure of distance between two points, induced by a strongly convex function ψ. In the case where ψ is the negative Shannon entropy, the resulting Bregman is the KL divergence. Similarly, when ψ is the negative Tsallis entropy, we get the Tsallis based Bregamn divergence. Particularly, the negative Tsallis entropy for any real number q is defined as:

$$\psi(\pi) = \frac{k}{1-q}\left(1 - \sum_a (\pi(a \mid s))^q\right)$$

Note that as $q \to 1$, the Tsallis collapses to the negative Shannon entropy $\sum_a \pi(a \mid s) \log \pi(a \mid s)$. Therefore, it generalizes the Shannon entropy. Hence, by definition, the Bregman divergence induced by the Tsallis entropy B_ψ can be written as

$$B_\psi(\pi, \pi_k) = \frac{kq}{1-q} \sum_a \pi(a \mid s)(\pi_k(a \mid s))^{q-1} - \frac{k}{1-q} \sum_a (\pi(a \mid s))^q$$

At first glance, the above expression is very different from the definition of the KL divergence. However, by defining the function $\log_q$ as

$$\log_q x \triangleq \frac{k\left(x^{q-1} - 1\right)}{q-1},$$

the Tsallis entropy can be written in a similar manner to the KL divergence,

$$\sum_a \pi(a \mid s)(\log_q \pi(a \mid s) - q \log_q \pi_k(a \mid s)).$$

With this convenient definition, we can write the Tsallis based version for the off-policy MDPO objective defined in Equation 13 as:

$$L^{Tsallis}(\theta, \theta_k) = \mathbb{E}_{\substack{s \sim \mathcal{D} \\ \epsilon \sim \mathcal{N}}}\big[\log_q \pi_\theta\big(\widetilde{a}_\theta(\epsilon, s)|s\big) - q \log_q \pi_{\theta_k}\big(\widetilde{a}_\theta(\epsilon, s)|s\big) - t_k Q_\psi^{\theta_k}\big(s, \widetilde{a}_\theta(\epsilon, s)\big)\big], \tag{19}$$

Note that although there is a q term next to the $\log_q \pi_{\theta_k}\big(\widetilde{a}_\theta(\epsilon, s)|s\big)$ term, in practice we found to have better performance without it, and therefore excluded it for our experiments. Note that on-policy MDPO uses a closed form version for the Bregman divergence (since both policies are Gaussian in our implementation, a closed form of their KL exists). Such a closed form version for the Tsallis based Bregman is quite cumbersome to handle in terms of implementation, and thus we did not pursue the Tsallis based version in the on-policy experiments. However, in principle, it is very much feasible and we leave this for future investigation. If not mentioned otherwise, k is set to 1.0.

C On-policy Results

Here, we report the results for all on-policy algorithms, i.e. TRPO, PPO and MDPO. We have three variants here, **1)** the *minimal version*, i.e. {TRPO, PPO, MDPO}-M, which makes use of no code level optimizations, **2)** the *loaded version*, i.e. {TRPO, PPO, MDPO}-LOADED, which includes all code level optimizations, and **3)** the *loaded+GAE* version, i.e. {TRPO, PPO, MDPO}-LOADED+GAE, which includes all code level optimizations and also includes the use of GAE. We see that the overall performance increases in most cases as compared to the minimal versions. However, the trend in performance between these algorithms remains consistent to the main results.

	MDPO	TRPO	PPO
Hopper-v2	**1964 (±217)**	**2382 (±445)**	1281 (±353)
Walker2d-v2	**2948 (±298)**	2454 (±171)	424 (±92)
HalfCheetah-v2	**2873 (±835)**	**1726 (±690)**	617 (±135)
Ant-v2	**1162 (±738)**	**1716 (±338)**	-40 (±33)
Humanoid-v2	**635 (±46)**	449 (±9)	448 (±56)
HumanoidStandup-v2	**127901 (±6217)**	100408 (±12564)	96068 (±11721)

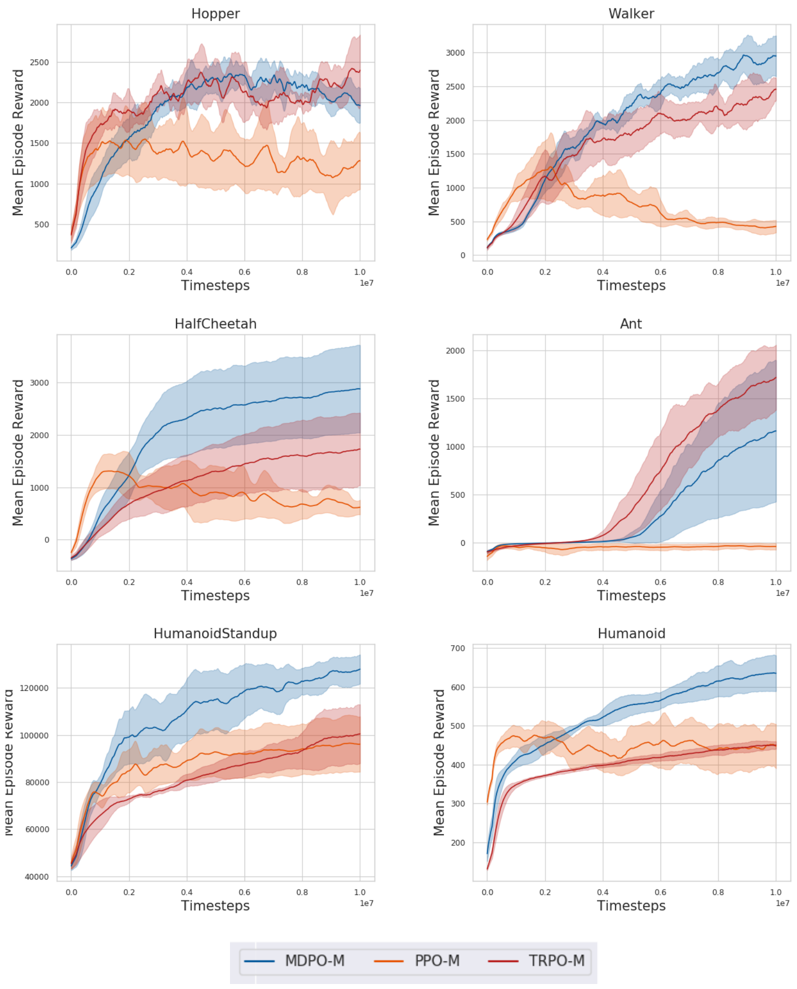


Figure 2: Performance of MDPO-M, compared against PPO-M, TRPO-M on six MuJoCo tasks.

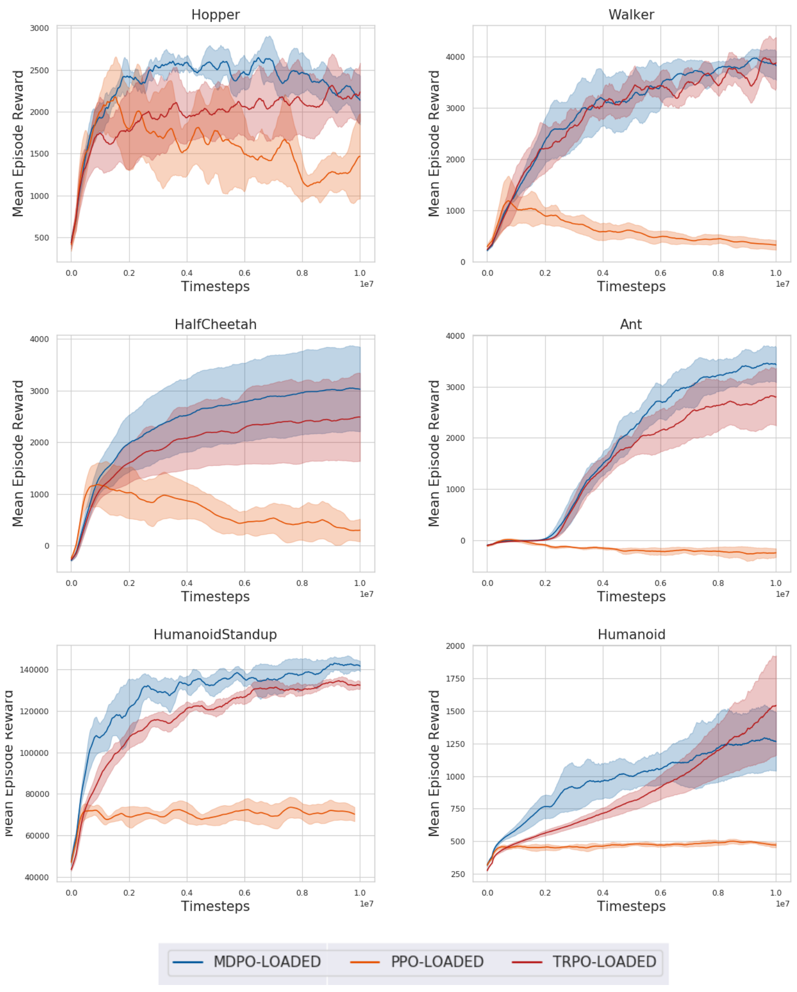


Figure 3: Performance of MDPO-LOADED, compared against loaded implementations (excluding GAE) of PPO and TRPO (PPO-LOADED, TRPO-LOADED) on six MuJoCo tasks.

	MDPO	TRPO	PPO
Hopper-v2	**2361 (±518)**	**1979 (±672)**	**2051 (±241)**
Walker2d-v2	**4834 (±607)**	**4473 (±558)**	1490 (±292)
HalfCheetah-v2	**4172 (±1156)**	**3751 (±910)**	2041 (±1319)
Ant-v2	**5211 (±43)**	4682 (±278)	59 (±133)
Humanoid-v2	3234 (±566)	**4414 (±132)**	529 (±47)
HumanoidStandup-v2	**155261 (±3898)**	149847 (±2632)	97223 (±4479)

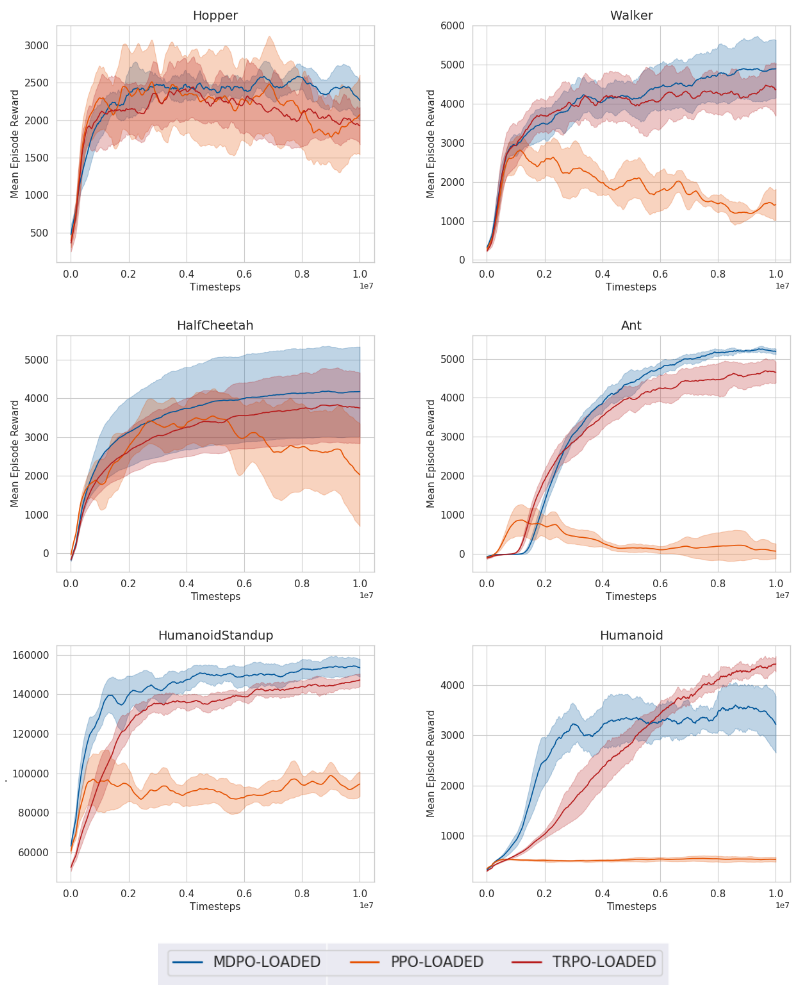


Figure 4: Performance of MDPO-LOADED+GAE, compared against loaded implementations (including GAE) of PPO and TRPO (PPO-LOADED+GAE, TRPO-LOADED+GAE) on six MuJoCo tasks.

D Off-policy Results

Here, we report the results for all off-policy algorithms, i.e. MDPO and SAC. We have two variants here, **1)** the *minimal version*, i.e. MDPO-M and SAC-M, which uses the standard neural network and batch sizes (64) and **2)** the *loaded version*, i.e. MDPO-LOADED and SAC-LOADED, which uses a neural network and batch size of 256. We see that the overall performance increases in most cases as compared to the minimal versions, i.e. SAC-M, MDPO-M. However, the trend in performance between these algorithms remains consistent to the main results.

	MDPO *KL*	MDPO *Tsallis*	SAC
Hopper-v2	**1385 (±648)**	**1108 (±417)**	**1501 (±414)**
Walker2d-v2	**873 (±180)**	**1151 (±218)**	635 (±137)
HalfCheetah-v2	8098 (±428)	**8477 (±450)**	**9298 (±371)**
Ant-v2	1051 (±284)	**2348 (±338)**	378 (±33)
Humanoid-v2	2258 (±372)	**4426 (±229)**	3598 (±172)
HumanoidStandup-v2	**131702 (±7203)**	**138157 (±8983)**	**142774 (±4864)**

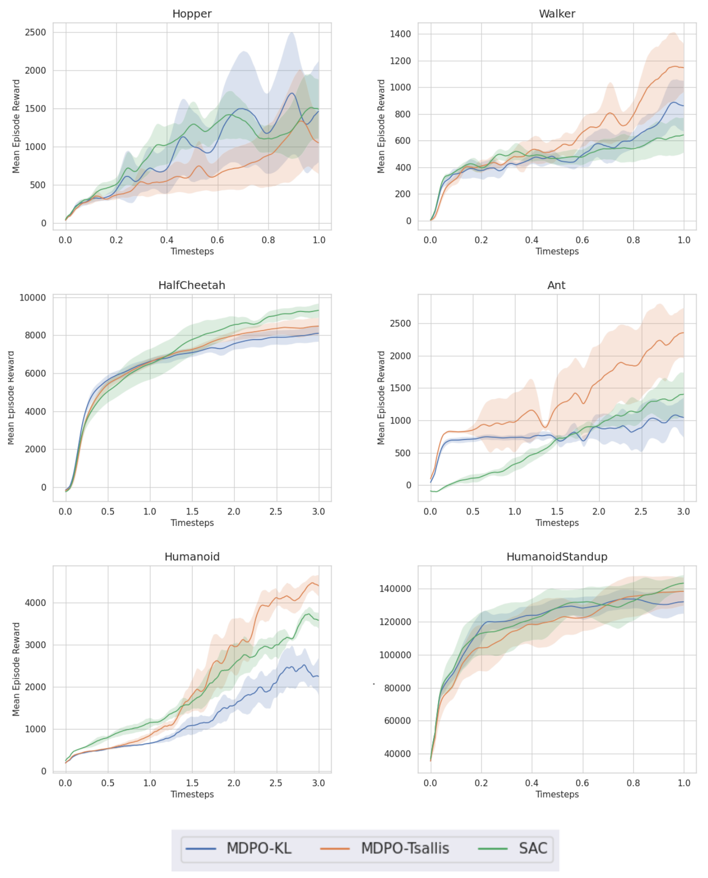


Figure 5: Performance of KL and Tsallis based versions of MDPO-M, compared with SAC-M on six MuJoCo tasks. X-axis represents time steps in millions.

	Hopper-v2	Walker2d-v2	HalfCheetah-v2	Ant-v2	Humanoid-v2	HumanoidStandup-v2
Tsallis q	0.8	0.2	0.5	2.0	0.2	0.5

Table 5: Tsallis q, used by minimal version of off-policy MDPO.

	MDPO *KL*	MDPO *Tsallis*	SAC
Hopper-v2	**2428** (±395)	**1882** (±650)	**1870** (±404)
Walker2d-v2	**3591** (±366)	**4028** (±287)	**3738** (±312)
HalfCheetah-v2	**11823** (±154)	10927 (±463)	**11928** (±342)
Ant-v2	**4434** (±749)	**5486** (±737)	**4989** (±579)
Humanoid-v2	**5323** (±348)	**5611** (±260)	**5191** (±312)
HumanoidStandup-v2	**143955** (±4499)	**165882** (±16604)	**154765** (±11721)

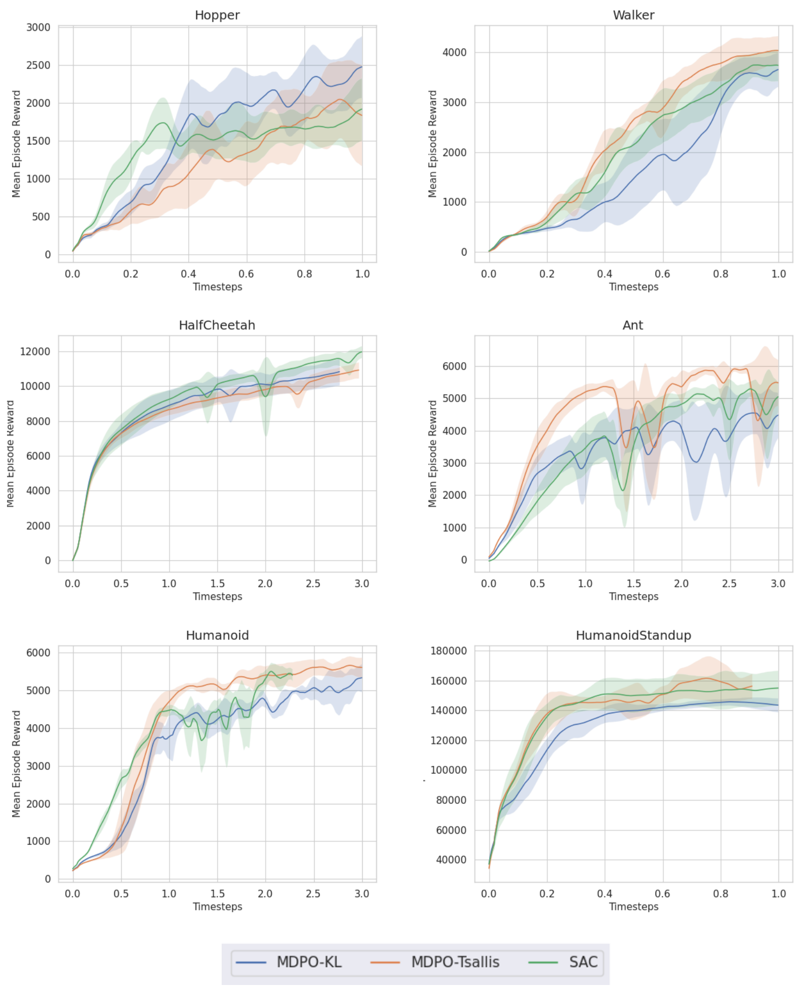


Figure 6: Performance of KL and Tsallis based versions of MDPO-LOADED, compared with SAC-LOADED on six MuJoCo tasks. X-axis represents time steps in millions. Note that although there is overlap in the performance of all methods, MDPO achieves a higher mean score in 5 out 6 domains.

	Hopper-v2	Walker2d-v2	HalfCheetah-v2	Ant-v2	Humanoid-v2	HumanoidStandup-v2
Tsallis q	0.8	0.5	0.8	2.0	0.2	1.5

Table 6: Tsallis q, used by loaded version of off-policy MDPO.