Investigating the Utility of Mirror Descent in Off-policy Actor-Critic

Samuel Neumann, Jiamin He, Adam White, Martha White

Keywords: mirror descent, off-policy, actor-critic, Soft Actor-Critic, Greedy Actor-Critic, Maximum A-Posteriori Policy Optimization

Summary

Many policy gradient methods prevent drastic changes to policies during learning. This is commonly achieved through a Kullback-Leibler (KL) divergence term. Recent work has established a theoretical connection between this heuristic and Mirror Descent (MD), offering insight into the empirical successes of existing policy gradient and actor-critic algorithms. This insight has further motivated the development of novel algorithms that better adhere to the principles of MD, alongside a growing body of theoretical research on policy mirror descent. In this study, we examine the empirical feasibility of MD-based policy updates in off-policy actor-critic. Specifically, we introduce principled MD adaptations of three widely used actor-critic algorithms and systematically evaluate their empirical effectiveness. Our findings indicate that, while MD-style policy updates are not significantly advantageous over conventional approaches to actor-critic, they can somewhat mitigate sensitivity to step size selection with widely used deep-learning optimizers.

Contribution(s)

1. We derive novel Mirror Descent variants of Soft Actor-Critic (SAC), Greedy Actor-Critic (GreedyAC), and Maximum A-Posteriori Policy Optimization (MPO) based on the Functional Mirror Descent (FMD) perspective.
   Context: A growing body of work on Policy Mirror Descent (PMD) has theoretically motivated the benefits of using Mirror Descent (MD) to update policies (Xiao, 2022; Johnson et al., 2023; Fatkhullin & He, 2024; Vieillard et al., 2020a; Lan, 2023). Much of this theory is for the tabular setting with exact policies. Recent work went one step further and re-derived several policy gradient algorithms for function approximation by introducing a functional MD (FMD) perspective (Vaswani et al., 2022). Such a perspective has yet to be brought to off-policy actor-critic methods that use approximate action-values and alternative losses for the actor. We are not claiming to have introduced the FMD perspective, nor that our derivations are complex, but they produce new algorithms.
2. We show these new MD variants exhibit no significant performance advantage over SAC, GreedyAC, and MPO across a variety of small problems and MuJoCo tasks.
   Context: It is possible there could be a difference in different environments.
3. We find that these MD algorithms provide (1) minor improvement in sensitivity to actor step size, (2) no improvement in sensitivity to entropy regularization parameter, and (3) no improvement with increasing replay ratio for actor updates; even though all three potential benefits are suggested by the theory.
   Context: Recent work suggests that policy gradient algorithms often encounter cliffs in the gradient direction, limiting step size magnitudes and explaining sensitivity (Jordan et al., 2024; Sullivan et al., 2022). Since MD updates account for policy-space distances, they should be more robust to step sizes. The KL in MD updates may already prevent policy collapse by regulating policy changes, hence these algorithms should be less sensitive to entropy regularization. Finally, MD updates prevent the algorithm from changing the policy too much, in probability space, and thus the amount of replay per step can be increased. One actor update corresponds to an approximate MD step; increasing the number of actor updates better approximates an exact MD step, which theoretically should perform better.

Investigating the Utility of Mirror Descent in Off-policy Actor-Critic

Samuel Neumann, Jiamin He, Adam White[†], Martha White[†]
{sfneuman,jiamin12,amw8,whitem}@ualberta.ca

Department of Computing Science, University of Alberta, Canada
Alberta Machine Intelligence Institute (Amii)
[†]CIFAR AI Chair

Abstract

Many policy gradient methods prevent drastic changes to policies during learning. This is commonly achieved through a Kullback-Leibler (KL) divergence term. Recent work has established a theoretical connection between this heuristic and Mirror Descent (MD), offering insight into the empirical successes of existing policy gradient and actor-critic algorithms. This insight has further motivated the development of novel algorithms that better adhere to the principles of MD, alongside a growing body of theoretical research on policy mirror descent. In this study, we examine the practicality of MD-based policy updates in off-policy actor-critic. Specifically, we introduce principled MD adaptations of three widely used actor-critic algorithms and systematically evaluate their empirical effectiveness. Our findings indicate that, while MD-style policy updates are not significantly advantageous over conventional approaches to actor-critic, they can somewhat mitigate sensitivity to step size selection with widely used deep-learning optimizers.

1 Introduction

Many policy optimization methods incorporate a term that prevents the policy from changing too much. This typically takes the form of a Kullback-Leibler (KL) divergence to the previous policy, either as a regularization penalty or constraint in the update. These include the widely-used algorithms TRPO (Schulman et al., 2015), PPO (Schulman et al., 2017) and MPO (Abdolmaleki et al., 2018b;a). Initial work proved that policy improvement could be guaranteed by preventing the policy from changing too much on each step (Kakade & Langford, 2002), inspiring follow-up work in the deep setting to explicitly constrain the policy update using KL divergences (Schulman et al., 2015; 2017; Mei et al., 2019).

These algorithms have since been framed as mirror descent (MD) algorithms for policy optimization (Neu et al., 2017; Geist et al., 2019; Vieillard et al., 2020a; Tomar et al., 2022; Vaswani et al., 2022). MD generalizes the typical Euclidean distance between parameter vectors in the gradient descent update to allow for any Bregman divergence, such as the KL divergence.[1] MD has been well-motivated in machine learning, because the updates better reflect the underlying problem geometry (Raskutti & Mukherjee, 2015; Gunasekar et al., 2021). The connection between these RL algorithms and MD motivated the development of new algorithms that more closely adhere to the requirements

Code available at https://github.com/samuelfneumann/MirrorDescentRL

[1]RL algorithms have used either a KL penalty or a KL constraint. The KL penalty form is the one that corresponds to a mirror descent update, and interestingly has been shown to be more effective than the constraint form (Lazić et al., 2021).

of a true MD update and there is a growing literature of theoretical results motivating these MD updates in reinforcement learning (Abbasi-Yadkori et al., 2019; Vieillard et al., 2020a;b; 2022; Zhu et al., 2023; Johnson et al., 2023; Alfano et al., 2023; Xiao, 2022; Fatkhullin & He, 2024; Lan, 2023; Xiong et al., 2024). Despite these compelling theoretical justifications, much less is known empirically about the utility of true MD updates for policy optimization.

In this work, we focus on the empirical practicality of MD in three off-policy actor-critic algorithms: SAC (Haarnoja et al., 2018; 2019), MPO (Abdolmaleki et al., 2018b), and GreedyAC (Neumann et al., 2023). We use the functional mirror descent perspective (Vaswani et al., 2022) and introduce principled MD variants of the actor updates in these algorithms. For MPO we remove the KL constraint, which intuitively plays a role similar to MD but does not actually give an MD update, and instead derive a more faithful MD update. We select three algorithms with different base updates to more broadly understand when, or if, MD can be beneficial in deep RL.

We contribute a set of empirical results including: a bakeoff-style comparison of MD variants against their standard non-MD configurations, followed by a sequence of targeted experiments designed to uncover if MD-based actor-critic methods achieve the practical benefits the theory suggests (Vaswani et al., 2022; Xiao, 2022; Alfano et al., 2023; Vieillard et al., 2020a; Geist et al., 2019). Across several continuous state and action environments, where function approximation is required, there appears to be little benefit to using these MD approaches. We then explore three settings where MD updates with function approximation could be beneficial: (1) mitigating the step-size cliff in policy gradient updates (Jordan et al., 2024; Sullivan et al., 2022), (2) reducing sensitivity to hyperparameter choices, (3) and increasing the actor replay ratio. We found MD updates do indeed reduce the step-size cliff; allowing for a larger range of step-size parameter values that avoid divergence. Unfortunately, in the other two settings, the results were mixed. Taken together, these results suggest MD and standard gradient descent algorithms appear to be empirically comparable.

2 Problem Formulation and Background

The agent-environment interaction is formalized as a Markov Decision Process (MDP) defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, \infty)$ is the transition dynamics, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. The agent starts in state $S_0 \sim d_0$, where $d_0 : \mathcal{S} \to [0, \infty)$ is the distribution of starting states. At each timestep $t = 1, 2, 3, \ldots, T$ the agent samples an action A_t from its policy $\pi(\cdot \mid S_t)$ where $\pi : \mathcal{S} \to \Delta_{\mathcal{A}}$ is a function mapping states to probability distributions over actions, $\Delta_{\mathcal{A}}$. After taking action A_t in state S_t, the agent transitions to a new state $S_{t+1} \sim \mathcal{P}(\cdot \mid S_t, A_t)$ and receives a scalar reward $R_{t+1} = r(S_t, A_t, S_{t+1})$. The return is the cumulative, future, discounted reward $G_t \doteq \sum_{k=t}^{T} \gamma^k r_{k+1}$. The agent's goal is to learn a policy π which maximizes $\tilde{J}(\pi) \doteq \mathbb{E}[G_0]$, where the expectation is with respect to d_0, $\mathcal{P}$, and π.

Typically, a parameterized policy $\pi_{\boldsymbol{\theta}}$ with parameters in $\boldsymbol{\theta} \in \mathbb{R}^n$ is learned by adapting $\boldsymbol{\theta}$ to optimize $J(\boldsymbol{\theta}) \doteq \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[G_0]$. Policy gradient methods like REINFORCE (Williams, 1992) obtain unbiased samples of $\nabla J(\boldsymbol{\theta})$, which involves obtaining full episode trajectories. These gradient estimates are expensive to sample and can be high variance. In this work we consider (off-policy) actor-critic algorithms that learn and use action-value functions. The action-value function of π is the expected return when taking action a in state s and following π thereafter, $q_\pi(s, a) \doteq \mathbb{E}_\pi [G_t | S_t = s, A_t = a]$. This function is approximated with a parametric critic $q_{\boldsymbol{w}}(s, a) \approx q_\pi(s, a)$. Such algorithms are typically off-policy, using replay buffers of past data to update the critic with one-step bootstrap updates, followed by different ways to update the policy to be greedy with respect to $q_{\boldsymbol{w}}$. We discuss these kinds of policy updates, and how to use mirror descent (MD) for them, in the next section. First, we give intuition about how to use MD in the simpler setting with REINFORCE.

REINFORCE makes gradient descent updates with stochastic samples of $\nabla J(\boldsymbol{\theta})$ by executing the parameterized policy $\pi_{\boldsymbol{\theta}}$ for an episode to obtain the trajectory $s_0, a_0, r_1, s_1, a_1, \ldots, s_T$, terminating at some variable time T. The stochastic gradient corresponds to $g_t \doteq \sum_{t=0}^{T-1} \gamma^t G_t \nabla \ln \pi(a_t | s_t)$

(Sutton et al., 1999). Because we can get unbiased, stochastic samples of the gradient, we can leverage standard optimization techniques like stochastic MD, explicitly shown by Vaswani et al. (2022). The standard gradient update corresponds to using

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &= \arg\min_{\boldsymbol{\theta}} \; -J(\boldsymbol{\theta}_t) + \langle -\nabla J(\boldsymbol{\theta}_t), \boldsymbol{\theta} - \boldsymbol{\theta}_t \rangle + \tfrac{1}{2\lambda}\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 \\ &= \arg\min_{\boldsymbol{\theta}} \langle -\nabla J(\boldsymbol{\theta}_t), \boldsymbol{\theta} \rangle + \tfrac{1}{2\lambda}\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2 = \boldsymbol{\theta}_t + \lambda \nabla J(\boldsymbol{\theta}_t)\end{aligned}$$

where the first line uses a first-order Taylor series approximation around $\boldsymbol{\theta}_t$, the second drops the constant and the third equality simply solves this quadratic minimization in closed-form.

The idea in MD is to replace the ℓ_2 distance with one that is more meaningful for policies. The above finds the best nearby parameters, under the local Taylor series approximation, where nearby is defined by Euclidean distance $\|\boldsymbol{\theta} - \boldsymbol{\theta}_t\|_2^2$. However, small changes in the policy parameters could actually result in large changes to the policy, and vice versa. What we really want is to find the best nearby policy in distribution space. In MD, we can replace the distance with any Bregman divergence in ϕ, D_ϕ, to get $\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \langle -\nabla J(\boldsymbol{\theta}_t), \boldsymbol{\theta} \rangle + \frac{1}{\lambda} D_\phi(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$, where ϕ is a (typically strictly) convex function called the *mirror map*. The KL divergence between parameters is a valid Bregman divergence. Unfortunately, in the general case no Bregman divergence *on parameters* $D_\phi(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$ can provide a KL divergence *on policies*.

Fortunately, a recent paper introduced a functional mirror descent (FMD) perspective (Vaswani et al., 2022) that makes it clear how we can obtain a KL divergence on the policies. The idea is to consider an MD step in policy space Π directly, also called the *functional* space:

$$\pi_{t+1} = \arg\min_{\pi \in \Pi} \langle -\nabla \tilde{J}(\pi_t), \pi \rangle + \tfrac{1}{\lambda} D_\phi(\pi, \pi_t)$$

where λ is a stepsize, π_t is the previous learned policy and the inner product $\langle , \rangle$ is over the space of states and actions. Since Π is just the parameterized policy space, we can remove the constraint and directly optimize over the parameters

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}} \langle -\nabla \tilde{J}(\pi_{\boldsymbol{\theta}_t}), \pi_{\boldsymbol{\theta}} \rangle + \tfrac{1}{\lambda} D_\phi(\pi_{\boldsymbol{\theta}}, \pi_t). \tag{1}$$

In practice, we do not completely solve the above optimization problem, and instead update $\boldsymbol{\theta}$ with $M > 1$ SGD steps on Equation (1).

A key subtlety to notice here is that we use $\nabla \tilde{J}(\pi_{\boldsymbol{\theta}_t})$ instead of $\nabla J(\boldsymbol{\theta}_t)$. The first is the gradient with respect to the policy directly, whereas the second is the gradient w.r.t. to $\boldsymbol{\theta}$. In the standard gradient descent update, we would use $\nabla J(\boldsymbol{\theta}_t)$. For MD, we have to reason about $\nabla \tilde{J}(\pi_{\boldsymbol{\theta}_t})$ instead. We will see this when deriving the MD updates for the off-policy actor-critic algorithms in the Section 3.

Remark: Because KL is not symmetric, we could also have chosen the KL in the other direction, $\text{KL}(\pi_{\boldsymbol{\theta}_t} || \pi_{\boldsymbol{\theta}})$, the mode-covering, forward KL (FKL). Vaswani et al. (2022) prove that using the above mode-seeking, reverse KL (RKL) divergence provides monotonic policy improvement guarantees for direct policy representations, under some conditions, but the FKL is preferable for softmax policy parameterizations. We provide derivations with both divergences in Section 3.[2]

3 Mirror Descent for Off-policy Actor-Critic Methods

In this section, we introduce the off-policy algorithms and develop their FMD variants.

[2]In RL, it is common to call the mode-seeking KL the *reverse KL* and the mode-covering KL the *forward KL*. In the optimization literation, unfortunately, it is exactly the opposite. To avoid confusion, we will periodically use the longer, more descriptive names.

3.1 A Generic Form of Mirror Descent for Off-policy Actor-Critic Methods

Actor-Critic methods alternate between updating an action-value critic and updating the actor. We consider a mirror descent variant for the actor update, where the loss decomposes across states s and changes as the action-values q change. Formally, $\ell_q(\boldsymbol{\theta}) \doteq \mathbb{E}_{s\sim d}[\ell_{q,s}(\boldsymbol{\theta})]$ for some weighting over states d, and $\ell_{q,s}(\boldsymbol{\theta})$ is defined differently for each actor-critic (AC) algorithm. For example, for vanilla AC, we have

$$\ell_{q,s}(\boldsymbol{\theta}) = -\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[q(s,a)\right]$$

with gradient (using the log-likelihood trick)

$$\nabla\ell_{q,s}(\boldsymbol{\theta}) = -\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[q(s,a)\nabla\ln\pi_{\boldsymbol{\theta}}(a\mid s)\right] \qquad \text{where } \nabla\ell_q(\boldsymbol{\theta}) = \mathbb{E}_{s\sim d}[\nabla\ell_{q,s}(\boldsymbol{\theta})].$$

In place of the standard gradient descent update, we can use a mirror descent update on this loss, using the same functional mirror descent argument as above. We define the loss on the policy directly $\tilde{\ell}_{q,s}(\pi) = -\mathbb{E}_{a\sim\pi(\cdot|s)}\left[q(s,a)\right]$ with $\tilde{\ell}_q(\pi) = \mathbb{E}_{s\sim d}[\tilde{\ell}_{q,s}(\pi)]$. The functional mirror descent update starts from the current π_t and obtains the next π_{t+1} using mirror descent update

$$\pi_{t+1} = \arg\min_{\pi\in\Pi} \langle\nabla\tilde{\ell}_q(\pi_t),\pi\rangle + \tfrac{1}{\lambda}D_\phi(\pi,\pi_t) \tag{2}$$

where for vanilla AC, $\frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(a|s)}(\pi_t) = -q(s,a)$ and $\langle\nabla\tilde{\ell}_q(\pi_t),\pi\rangle = -\mathbb{E}_{s\sim d,\ a\sim\pi(\cdot|s)}[q(s,a)]$. We can again rewrite the update in terms of $\boldsymbol{\theta}$, getting

$$\boldsymbol{\theta}_{t+1} = \arg\min_{\boldsymbol{\theta}}\ f_q(\boldsymbol{\theta}) \qquad \text{for surrogate loss } f_q(\boldsymbol{\theta}) \doteq \langle\nabla\tilde{\ell}_q(\pi_{\boldsymbol{\theta}_t}),\pi_{\boldsymbol{\theta}}\rangle + \tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}},\pi_t).$$

We typically cannot solve this optimization in closed-form. Instead, we approximate the solution by taking multiple gradient descent steps on this surrogate loss. For vanilla AC, we use the gradient

$$g_t = \nabla_{\boldsymbol{\theta}} f_q(\boldsymbol{\theta}) = -\mathbb{E}_{s\sim d,\ a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[q(s,a)\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(a|s)] + \tfrac{1}{\lambda}\nabla_{\boldsymbol{\theta}}D_\phi(\pi_{\boldsymbol{\theta}},\pi_t). \tag{3}$$

Or, alternatively, we can use SGD steps, with mini-batches of transitions from $s\sim d$ on loss

$$f_{q,s}(\boldsymbol{\theta}) \doteq \langle\nabla\tilde{\ell}_{q,s}(\pi_{\boldsymbol{\theta}_t}),\pi_{\boldsymbol{\theta}}(\cdot|s)\rangle + \tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s),\pi_t(\cdot|s)) \quad \text{where } f_q(\boldsymbol{\theta}) \doteq \mathbb{E}_{s\sim d}[f_{q,s}(\boldsymbol{\theta})]. \tag{4}$$

For vanilla AC, we get a surprising coincidence: the gradient in functional space $\nabla_{\boldsymbol{\theta}}\langle\nabla\tilde{\ell}_q(\pi_{\boldsymbol{\theta}_t}),\pi_{\boldsymbol{\theta}}\rangle$ is actually the same as in parameter space $\nabla_{\boldsymbol{\theta}}\langle\nabla\ell_q(\boldsymbol{\theta}_t),\boldsymbol{\theta}\rangle$. Hence, we could have obtained the same update by swapping the Euclidean distance with a KL, without taking this functional perspective. In other words, the MD update for vanilla AC uses the same gradient as usual, but adds a KL-penalty term $\lambda^{-1}D_\phi(\pi_{\boldsymbol{\theta}},\pi_t)$. This correspondence may help explain why several previous derivations did not use the functional view, and instead simply jumped to the KL form. An example of this is MDPO (Tomar et al., 2022), where the MD update is given in terms of the policy directly, and the switch to policy parameters is done without considering the potential discrepancy. Again, this happens to be correct for vanilla AC—there is no discrepancy—and MDPO (correctly) uses Equation (3) above. In general, though, we will not have such an equality, and using MD for other AC algorithms will not correspond to simply adding a KL-penalty term to the gradient descent update.

3.2 Functional Mirror Descent for Soft Actor-Critic

The previous section laid the ground-work for deriving the FMD updates for SAC; we simply need to define our surrogate loss. More specifically, we need to define $\tilde{\ell}_{q,s}$ and obtain its gradient to get both the surrogate loss $f_{q,s}$ and its gradient to get our multiple MD updates.

SAC minimizes a mode-seeking KL to the Boltzmann distribution over q with entropy scale $\tau > 0$,

$$\ell_{q,s}(\boldsymbol{\theta}) = \tau\mathrm{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s)\ ||\ \mathcal{B}^\tau q(\cdot\mid s)) \qquad \text{for Boltzmann } \mathcal{B}^\tau q(a\mid s)\propto\exp\left(\frac{q(s,a)}{\tau}\right).$$

The loss is scaled by τ to adjust the increasing magnitude for larger entropy scales. The gradient for this loss is $\nabla\ell_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[(-q(s,a)+\tau\ln\pi_{\boldsymbol{\theta}}(a|s))\nabla\ln\pi_{\boldsymbol{\theta}}(a|s)]$.

Proposition 1. *On time step t with current action-values q and policy π_t, the surrogate objective for Soft Actor-Critic with a direct functional representation and any Bregman divergence D_ϕ, is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[-q(s,a) + \tau \ln \pi_t(a \mid s)] + \tfrac{1}{\lambda} D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \tag{5}$$

with gradient

$$\nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[(-q(s,a) + \tau \ln \pi_t(a \mid s))\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)] + \tfrac{1}{\lambda}\nabla_{\boldsymbol{\theta}} D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)).$$

Proof. We first define the loss in policy space, $\tilde{\ell}_{q,s}(\pi) \doteq \tau \mathrm{KL}(\pi(\cdot|s) \mid\mid \mathcal{B}^\tau q(\cdot \mid s))$. Now we differentiate it

$$\frac{\partial \tilde{\ell}_{q,s}}{\partial \pi(a \mid s)}(\pi_t) = -q(s,a) + \tau(\ln \pi_t(a \mid s) + 1). \tag{6}$$

Taking the inner product with $\pi_{\boldsymbol{\theta}}(\cdot|s)$, we get

$$\left\langle \frac{\partial \tilde{\ell}_{q,s}}{\partial \pi(a|s)}(\pi_t), \pi_{\boldsymbol{\theta}}(\cdot|s) \right\rangle = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[-q(s,a) + \tau(\ln \pi_t(a|s) + 1)] = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[-q(s,a) + \tau \ln \pi_t(a|s)]$$

where the 1 disappears because $\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[1] = 1$ and we ignore constants. We can plug this into Equation (4) to get Equation (5). Taking the gradient is straightforward, since neither $q(s,a)$ nor $\ln \pi_t(a \mid s)$ depend on $\boldsymbol{\theta}$. Again using the log likelihood trick, we get $\nabla \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[-q(s,a) + \tau \ln \pi_t(a \mid s)] = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[(-q(s,a) + \tau \ln \pi_t(a \mid s))\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)]$. □

MD-SAC with a mode-seeking KL penalty (Reverse KL)

Corollary 1. *On time step t with current action-values q and policy π_t, the surrogate objective for Soft Actor-Critic with a mode-seeking KL, namely $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = KL(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$, is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[-q(s,a) + \left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{7}$$

with gradient

$$\nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left(-q(s,a) + \left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right)\nabla_{\boldsymbol{\theta}} \ln(\pi_{\boldsymbol{\theta}}(a \mid s))\right].$$

Proof. We need to plug $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = \mathrm{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$ into Equation (5). We have

$$\tfrac{1}{\lambda}\mathrm{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s)) = \tfrac{1}{\lambda}\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln \pi_{\boldsymbol{\theta}}(a|s) - \ln \pi_t(a|s)\right].$$

Combining this $-\frac{1}{\lambda}\ln \pi_t(a|s)$ with the $\tau \ln \pi_t(a|s)$ term from the main loss, we get Equation (7).

To derive the gradient, we primarily need to compute the gradient for the entropy term $\nabla \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[\ln \pi_{\boldsymbol{\theta}}(a|s)]$. For completeness, we derive the gradient of the entropy in Lemma 1. Using the log-likelihood trick with Lemma 1, yields the gradient above. □

This update is similar to an actor-critic variant of the MD policy update employed by DAPO-KL (Xiong et al., 2024), but derived from the FMD perspective. We can contrast this with the standard SAC update, which uses $\nabla \ell_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[(-q(s,a) + \tau \ln \pi_{\boldsymbol{\theta}}(a|s))\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)]$. For the very first MD step, the updates are actually the same! The reason is that on MD iteration $k = 0$, we start from $\pi_{\boldsymbol{\theta}_0} = \pi_t$ and so the KL divergence does not play a role. After the first step, however, $\pi_{\boldsymbol{\theta}_k}$ no longer equals π_t and we have a different update. We can also see the update is different from simply adding a KL penalty to the SAC update, which would provide the gradient

$$\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}[(-q(s,a) + \left(\tau + \tfrac{1}{\lambda}\right)\ln \pi_{\boldsymbol{\theta}}(a \mid s) - \tfrac{1}{\lambda}\ln(\pi_t(a \mid s)))\nabla_{\boldsymbol{\theta}} \ln(\pi_{\boldsymbol{\theta}}(a \mid s))]. \tag{8}$$

MD-SAC with a mode-covering KL penalty (Forward KL) Using the mode-covering KL in Proposition 1 produces an algorithm that looks like TRPO, albeit with entropy regularization. Namely we would get a surrogate loss $\mathbb{E}_{a\sim\pi_t}[(-q(s,a)+\tau\ln\pi_t(a\mid s))\frac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_t(a|s)}-\frac{1}{\lambda}\ln\pi_{\boldsymbol{\theta}}(a\mid s)]$. However, unlike the mode-seeking KL, the mode-covering KL is not a Bregman divergence. So it is not a technically valid choice for mirror descent, which is restricted to using Bregman divergences.

Instead, to get the mode-covering KL on the policy, we actually need to consider a Bregman divergence on the logits of the policy. This choice results in using a mode-covering KL on the policy itself. Vaswani et al. (2022) showed this result for a REINFORCE update; we prove this result for SAC below. Assume we have a softmax parameterization, $\pi_{\boldsymbol{\theta}}(a\mid s)\propto\exp(z_{\boldsymbol{\theta}}(a,s))$. Our network outputs $z_{\boldsymbol{\theta}}(a,s)$, and we consider the functional space over z instead.

Proposition 2. *Assume we have a finite number of actions, $|\mathcal{A}|$, and use a softmax policy parameterization, $\pi_{\boldsymbol{\theta}}(a|s)=\frac{\exp(z_{\boldsymbol{\theta}}(a,s))}{\sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(a,s))}$. On time step t with current action-values q and policy π_t and logits z_t, the surrogate objective for Soft Actor-Critic with the log-sum-exp mirror map*

$$f_{q,s}(\boldsymbol{\theta})=\mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t(a,s)}-\tfrac{1}{\lambda})\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

with gradient

$$\nabla f_{q,s}(\boldsymbol{\theta})=\mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t(a,s)}-\tfrac{1}{\lambda})\nabla\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

where $v(s)\doteq\sum_{j=1}^{|\mathcal{A}|}q(s,j)\pi(j\mid s)$ and $\varsigma_{\pi_t}(a,s)=-q(s,a)+v(s)+\tau\ln\pi_t(a\mid s)+\tau\mathcal{H}(\pi_t(\cdot\mid s))$

The proof is given in Section 8.2 in the Supplementary Material. A key difference from Equation (7) is that actions are sampled from π_t instead of $\pi_{\boldsymbol{\theta}}$. The reason for this is that the mode-covering KL for the penalty requires sampling from π_t and the SAC objective—which uses a mode-seeking KL—requires sampling from $\pi_{\boldsymbol{\theta}}$. To avoid sampling actions twice, we should choose one of these policies and use importance sampling for one of the terms. We opted for sampling from π_t, because this better matches existing algorithms like TRPO and because the gradient when sampling from $\pi_{\boldsymbol{\theta}}$ is much messier.

3.3 Functional Mirror Descent for MPO

Maximum A-Posteriori Policy Optimization (MPO) (Abdolmaleki et al., 2018b) is an off-policy actor-critic algorithm that incorporates KL regularization in two ways. The original explanation builds on relative entropy policy search (Peters et al., 2010), with a relatively complex derivation, but recent work recast it in simpler terms (Vieillard et al., 2020a). The idea is as follows. Ideally, we would extract the following policy after updating the critic q_t on time step t, for all s

$$\pi^{\text{KL}}(\cdot|s)=\arg\min_p\ -\mathbb{E}_{a\sim p}[q_t(s,a)]+\kappa\text{KL}(p\mid\mid\pi(\cdot|s))\propto\pi(a|s)\exp\left(\frac{q_t(s,a)}{\kappa}\right).$$

This closed-form solution for the policy maximizes the vanilla AC objective, under a constraint to stay close to the current policy, and can be interpreted as a mirror descent or natural gradient update in policy space (Xiao, 2022; Agarwal et al., 2021). However, this policy may not be in the policy class. MPO approximates this target policy by minimizing a mode-covering KL divergence to it:

$$\ell_{q,s}(\boldsymbol{\theta})=\text{KL}(\pi^{\text{KL}}(\cdot|s)\mid\mid\pi_{\boldsymbol{\theta}}(\cdot|s))=\mathbb{E}_{a\sim\pi^{\text{KL}}(\cdot|s)}[\ln\pi_{\boldsymbol{\theta}}(a|s)]=\mathbb{E}_{a\sim\pi_t(\cdot|s)}[\tfrac{1}{\kappa}q_t(s,a)\ln\pi_{\boldsymbol{\theta}}(a|s)].\quad(9)$$

This objective already prevents the new policy from moving too far from the current policy, by encouraging the policy to match π^{KL}. However, it is implicit and it is not obvious precisely how much it will deviate from the current policy. MPO, therefore, also adds a mode-seeking KL constraint to the optimization: $\text{KL}(\pi_{\boldsymbol{\theta}}\mid\mid\pi_{\boldsymbol{\theta}_{t-1}})<\epsilon$.

We can revisit this algorithm in light of the FMD perspective. We consider Equation (9) to be the loss for MPO, and derive the functional mirror descent update for this loss, instead of using the added constraint. For space, here we provide the losses and include the gradient and derivations in

Section 8 of the Supplementary Material. The generic form for the MD update for MPO uses the following surrogate loss:

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[-e^{\frac{q(s,a)}{\kappa}}\right] + \tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)).$$

The derivation is in Proposition 6. Using a mode-seeking KL for the Bregman divergence, we get (see Corollary 3)

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[-e^{\frac{q(s,a)}{\kappa}} - \tfrac{1}{\lambda}\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right].$$

For the softmax representation with a log-sum-exp mirror map (mode-covering, forward KL), we obtain surrogate loss (see Proposition 7)

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\left(1 - \exp\left(\frac{q(s,a)}{\kappa}\right) - \tfrac{1}{\lambda}\right)\ln\pi_{\boldsymbol{\theta}}(a|s)\right].$$

3.4 Functional Mirror Descent for Greedy Actor-Critic

Greedy Actor-Critic (GreedyAC) is a recently proposed off-policy actor-critic algorithm (Neumann et al., 2023) inspired by the cross-entropy method. GreedyAC defines a percentile policy on q, that increases the probability of actions in the top ρ percentile and zeros the probability (density) for all other actions. Like MPO, this target policy π^ρ is difficult to directly represent, but it is feasible to minimize a (forward) KL divergence to it because we can sample from π^ρ. The GreedyAC algorithm also incorporated entropy, giving us the combined loss

$$\ell_{q,s}(\boldsymbol{\theta}) = \mathrm{KL}(\pi^\rho(\cdot \mid s) \mid\mid \pi_{\boldsymbol{\theta}}(\cdot \mid s)) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}[-\ln\pi_{\boldsymbol{\theta}}(a|s)] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[\ln\pi_{\boldsymbol{\theta}}(a|s)]. \quad (10)$$

The generic form for the FMD update for GreedyAC uses the following surrogate loss

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\tfrac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_t(a|s)}\right] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t(a|s)\right] + \tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)).$$

Notice now that we need to sample the primary loss with the percentile policy and the entropy separately. The GreedyAC algorithm does not have a closed-form for the percentile policy—namely we can sample from it but cannot compute its probabilities—so we cannot use importance sampling to make the two match. The derivation is given in Proposition 4. When using a mode-seeking KL for the Bregman divergence, we get (see Corollary 2)

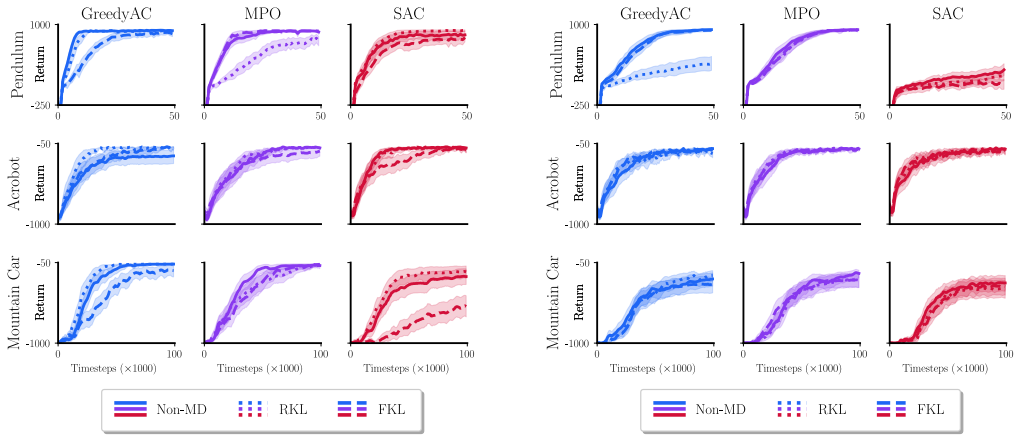
$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\tfrac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_t(a|s)}\right] + \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a|s)) + \tfrac{1}{\lambda}\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

For the softmax representation with a log-sum-exp mirror map (mode-covering, forward KL), we obtain surrogate loss (see Proposition 5)

$$f_{q,s}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{a\sim\pi^\rho(\cdot|s)}[-\ln\pi_{\boldsymbol{\theta}}(a \mid s)] + \mathop{\mathbb{E}}_{a\sim\pi_t(\cdot|s)}\left[\left(\tau\ln\pi_t(a|s) + \tau\mathcal{H}(\pi_t(\cdot|s)) + 1 - \tfrac{1}{\lambda}\right)\ln\pi_{\boldsymbol{\theta}}(a|s)\right].$$

4 Empirical Study

In this section we empirically investigate MD variants of GreedyAC, SAC, and MPO. We used several continuous- and discrete-action classic control environments: Pendulum (Degris et al., 2012), Acrobot (Sutton & Barto, 2018), and Mountain Car (Sutton & Barto, 2018). Discrete actions included the extreme continuous actions and zero. Experiments always consisted of 100,000 steps except for Pendulum which used 50,000 steps. We also used HalfCheetah-v4, Walker2D-v4, and Hopper-v4 from the MuJoCo suite (Towers et al., 2023) and used 1 million steps for each. In all plots, solid lines denote mean performance with shaded regions denoting 95% bootstrap confidence intervals. All learning curves are smoothed for readability.



(a) Continuous-Action

(b) Discrete-Action

Figure 1: Learning curves on the classic control suite. Dashed and dotted lines indicate MD-style updates with FKL and RKL penalties respectively. Solid lines denote non-MD updates.

Summary and some Implementation Details Hereon, we will label the FMD variants of SAC, MPO, and GreedyAC using MD-SAC, MD-MPO and MD-GreedyAC with RKL or FKL to indicate the variant. Standard algorithm variants are sometimes referred to using *non-MD*. Typically, we approximated the closed-form MD update with $M = 10$ SGD updates, performing all M SGD updates on the same environment step. For computational feasibility in Section 4.1, we performed an MD update (i.e. M SGD updates) every M environment steps, ensuring that the number of SGD updates was equal for MD and non-MD algorithms. For all other experiments, we performed an MD update (i.e. M SGD updates) on each environment step.

All algorithms optimize updates using a log-likelihood form for consistency, a natural choice for GreedyAC and MPO. While SAC typically uses reparameterization, this is challenging for MD-SAC with the FKL, so we apply the log-likelihood form to all SAC variants. Using a baseline $v(s)$ for advantage estimates $q(s, a) - v(s)$ was crucial, rather than relying solely on an action-value critic. We approximated $v(s)$ with a sample average of 30 action-values from policy-sampled actions. This approach aligns with the original SAC proposal (Haarnoja et al., 2018), though later versions (Haarnoja et al., 2019) omitted it. In the continuous-action setting, MD-GreedyAC with the RKL exhibited unstable learning due to the likelihood ratio in its gradient. We found clipping to be crucial for stability (see Supplementary Material, Section 7).

We used the Adam optimizer (Kingma & Ba, 2014) with neural networks for both actor and critic, each consisting of 2 hidden layers with 256 units for MuJoCo, 64 units for continuous-action classic control, and 32 units for discrete-action classic control per layer. Continuous-action policies were parameterized as Squashed Gaussian distributions with a separate network predicting location and log-scale parameters, sharing inputs. We clipped the log-scale parameter to $\pm 10^3$ and exponentiated for positivity. For discrete-actions, we used softmax policies. All algorithms used a single action-value critic. Hyperparameter sweeps and tuned values are listed in Section 7 in the Supplementary.

4.1 Mirror Descent Bake-Off

We begin with a bake-off style analysis, comparing each algorithm to its two MD variants, sweeping hyperparameters and reporting performance over 50 runs[3] (30 for MuJoCo). Since hyperparameter tuning is often impractical in real-world applications, we assess algorithm sensitivity using a cross-environment hyperparameter tuning procedure (Neumann et al., 2023), tuning across the continuous-action classic control, discrete-action classic control, and MuJoCo suites separately.

[3]For continuous-action MD-GreedyAC with RKL, we only used 30 runs.

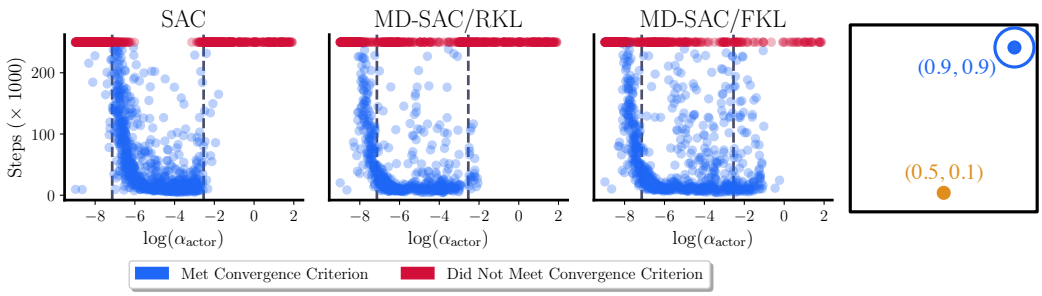


Figure 3: (**Left**) The number of steps for SAC and MD-SAC to learn a near-optimal policy for at least 1,500 runs. Each point represents a single agents with a randomly sampled $\log(\alpha_{\text{actor}}) \in [-9, 2]$. An algorithm which is robust to the step size cliff would exhibit a wide, contiguous region of points along the x-axis which are low on the y-axis, indicating that a wide range of step sizes induce convergence to a near-optimal policy. Dashed lines indicate smallest/largest inlier working step sizes for SAC. (**Right**) The environment.

Classic Control Figures 1a and 1b show the learning curves on the continuous- and discrete-action classic control suite respectively. We did not find a consistent trend between MD and non-MD algorithms, both performing similarily.

MuJoCo Gymnasium Suite Despite the negative result, it is possible that things change in more difficult environments. Figure 2 shows the learning curves of SAC and its MD variants on the MuJoCo suite. Again, MD-style updates provided no consistent benefit. MD-SAC with FKL learned a better policy than SAC on Hopper but learned a worse policy than SAC on HalfCheetah. In all cases, SAC and MD-SAC with RKL performed similarly.

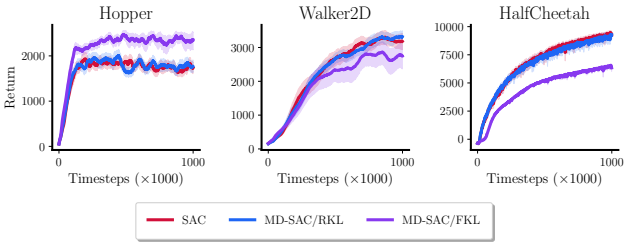


Figure 2: Mujoco Suite

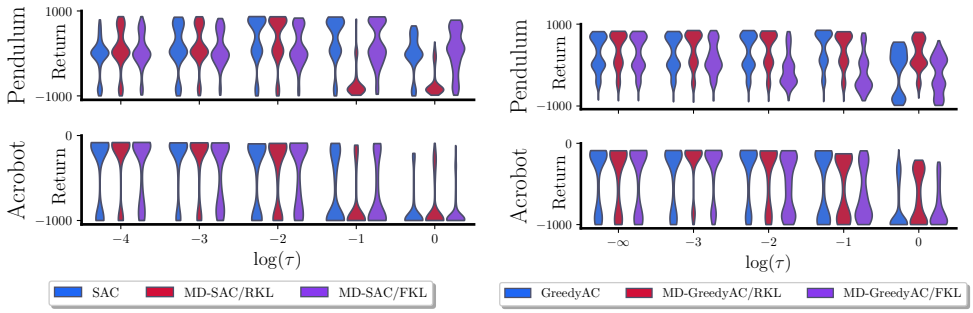
4.2 Further Analysis of the Functional Mirror Descent Update

The previous section found no definitive advantage to using the MD-style actor-critic algorithms in terms of learning speed. However, this does not preclude the possibility that MD offers other benefits. Our previous analysis may have focused on the wrong aspects, overlooking scenarios where MD could be advantageous. In this section, we conduct a more in-depth examination of the MD update to identify conditions under which it may provide meaningful improvements.

4.2.1 Mitigating the Step Size Cliff

Large actor step sizes can severely degrade policies (Jordan et al., 2024; Sullivan et al., 2022), known as the *step size cliff*. This cliff can be avoided by using small actor step sizes but such a strategy slows learning. Step sizes must be chosen to balance learning speed and stability. Since MD better accounts for policy changes, it may enhance robustness to the step size cliff.

To test this, we considered a continuous-state, discrete-action MDP (Figure 3, Right). States consisted of the agent's (x, y) position, between $[0, 0]$ (lower left) and $[1, 1]$ (upper right). The agent, starting at $(0.5, 0.1)$, took actions either moving it nowhere or moving it 0.05 units along the four cardinal directions or four diagonals. The goal was a 0.1 radius circle at $(0.9, 0.9)$, with rewards of -0.01 per step and an additional $+10$ for entering the goal, which induced episode termination. We truncated episodes at 1,000 steps (without termination), set $\gamma = 1$, and ran experiments for



(a) SAC (b) GreedyAC

Figure 4: Performance distributions across entropy scales.

250,000 steps. The maximum and minimum attainable return was $G_{\text{max}} = 9.85$ and $G_{\text{min}} = -10$ respectively for each (potentially cutoff) episode.

To quantify how actor step sizes affect learning, we measured performance as the number of environment steps required for SAC and MD-SAC to maintain performance above a threshold ϱ. We set $M = 10$ and updated every environment step. Following Jordan et al. (2024), we estimated an approximate lower bound on $\tilde{J}(\pi)$ using the last n returns without pausing learning. Concretely, we let $\tilde{J}_{e,n} = \bar{G}_{e-n+1:e} - 3\frac{\sigma(\bar{G}_{e-n+1:e})}{\sqrt{n}}$, where $\bar{G}_{e-n+1:e}$ and $\sigma(\bar{G}_{e-n+1:e})$ are the sample mean and standard deviation of the last n returns, and terminate when $\tilde{J}_{e,n} \geq \varrho$. We set $n = 100$ and $\varrho = G_{\text{min}} + 0.95(G_{\text{max}} - G_{\text{min}})$.

Figure 3 shows convergence times across at least 1,500 randomly sampled $\log(\alpha_{\text{actor}}) \in [-9, 2]$ per algorithm using the Adam optimizer (Kingma & Ba, 2014). We fixed $\tau = \alpha_{\text{critic}} = 10^{-3}$, ensuring SAC could learn well across many actor step sizes, and $\lambda = 10^{-1}$ for MD-SAC variants. Occasionally, extreme α_{actor} yielded convergence. We applied a local outlier factor algorithm to identify the largest/smallest inlier α_{actor} leading to convergence and show these as dashed lines in Figure 3 (see Supplementary Material, Section 10.1).

A broader range of actor step sizes resulted in convergence for MD-SAC compared to SAC. Within the lower limit of SAC's working step size range, MD-SAC variants often converged faster. Further, MD-SAC with FKL was most robust to large step sizes. Even with good step sizes, each algorithm occasionally failed to learn a near-optimal policy within the duration of the experiment, especially noticeable for MD-SAC. Notably, many of these good—yet unsuccessful—step sizes produced reasonable performance but did not meet our convergence criterion. Furthermore, λ influenced the range of good step sizes for MD-SAC (see Supplementary Material, Section 10.1), generally increasing the range of good step sizes as λ decreased (higher KL penalty).

4.2.2 Sensitivity to Entropy Scale

Actor-Critics are notoriously sensitive to the entropy scale hyperparameter (Eimer et al., 2023; Neumann et al., 2023). Intuitively, MD should mitigate this sensitivity by restricting policy changes in policy space. Further, the FKL penalty form provides additional protection since the update samples actions from π_t rather than from $\pi_{\boldsymbol{\theta}}$. We analyzed performance distributions across different entropy scales τ for SAC and GreedyAC. Using the experimental setup in Section 4.1, we conducted 50 additional runs of MD-SAC and MD-GreedyAC for $\log \tau \in \{-4, 0\}$ (MD-SAC) and $\log \tau \in \{-3, 0\}$ (MD-GreedyAC). Figure 4 shows the performance distribution of each algorithm with λ set to the tuned values from Section 4.1. We see that MD-style updates did not substantially improve robustness to τ for either algorithm, though a moderate improvement was observed for MD-SAC (FKL) and MD-GreedyAC (RKL) on Pendulum.

4.2.3 Increasing the Actor Replay Ratio

In this section we test if increasing M can improve the MD update, because doing so could give a better approximation of the closed-form MD update in Equation 2. On each step, we increase the number of actor updates, though this still only corresponds to an approximation to one MD update. For a fair comparison, we also increase the number of actor updates for the non-MD variants, though the interpretation in that case is different. Each actor update is another standard gradient descent update. We expect that increasing the number of actor updates for the MD algorithms should be beneficial, but may actually be harmful for the non-MD algorithms.

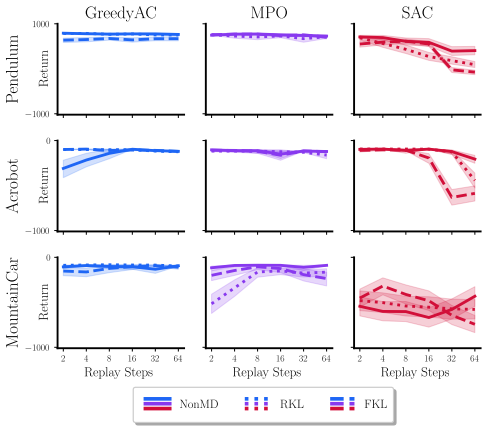


Figure 5: Performance vs replay ratio over 60 runs with continuous-actions.

Using the tuned hyperparameters from Section 4.1, we evaluated each algorithm on the classic control suite with actor replay ratios of 2, 4, 8, 16, 32, or 64 gradient updates for the actor per environment step over 60 runs, setting M to the number of actor updates per step and updating on each environment step. Our goal was to test the policy update, so we fixed the critic updates to 1 per step.[4] Figure 5 shows the results for the continuous-action setting. Discrete-action results are shown in Section 10.2 in the Supplementary Material.

We found no consistent relationship between actor replay ratio and performance. While large actor replay ratios sometimes degraded performance as expected, MD-style algorithms did not consistently benefit from increased updates and, in some cases, performed worse. Moreover, MD-style methods were not typically more robust to the actor replay ratio than non-MD variants, though they exhibited slightly greater robustness in the continuous-action setting than in the discrete-action setting.

5 Discussion and Conclusion

Our empirical analysis did not provide evidence for a significant difference between MD and NonMD algorithm variants. In this section we discuss potential reasons for this and propose further avenues for investigation. The potential reasons can be grouped into two categories: difficulty in approximately solving the MD optimization problem (Equation 4) and difficulty in optimizing an actor-critic algorithm's objective with MD.

Key to the performance of MD algorithms is how well the optimization problem in Equation 4 is approximated. If the approximation is too coarse, then the MD algorithm variants may be quite similar to their Non-MD counterparts. In a similar vein, if this optimization problem is difficult to solve and not much progress is made for any number of inner updates M, the resulting algorithm would again be quite similar to a Non-MD algorithm. In fact, we observed some evidence of this, since we found that the actor step size α_{actor} was more crucial for performance than λ. In reality, if our approximation to the MD optimization problem was good enough, then α_{actor} should have had a smaller impact on performance since its role in optimizing the policy is secondary to λ. If we were able to exactly solve the MD optimization problem, then α_{actor} should play no role at all in the MD update.

The performance of actor-critic algorithms heavily rests on the kind of optimizer used, and modern optimizers are crucial for successfully training neural networks with actor-critics. In contrast, SGD with fixed step sizes is typically much less effective practically. In this work, we used the Adam

[4]Note that increasing the replay ratio for the action-value has been shown to degrade performance (D'Oro et al., 2023; Nikishin et al., 2022). But here we keep the number of critic updates at 1, and so those results do not directly apply.

optimizer (Kingma & Ba, 2014) for training neural networks. But for MD, the hyperparameter λ is well though of as a step size controlling how updates are made in policy space. Although in this work we tuned the MD step size λ, it was nevertheless held constant after tuning. Because of this, the MD algorithms could have been at a significant disadvantage. We likely need an optimizer for the MD step size.

We propose a few avenues for future research. The first being to verify whether the surrogate MD optimization problem is approximately solved in these MD-style actor-critic and policy gradient algorithms, and to develop improved methods for approximating the MD optimization problem if not. This could include different optimization strategies beyond Adam, such as second order methods, line search, or meta-learning. Next, we suggest developing new adaptive step size strategies for adjusting λ rather than keeping it constant, and potentially considering joint tuning of α_{actor} and λ. Finally, we suggest a more thorough investigation of MD and actor-critic components to better understand the interplay between MD and actor-critic objectives, especially pertaining to how actor and critic step size affect the MD update and performance.

References

Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellert Weisz. POLITEX: Regret Bounds for Policy Iteration using Expert Prediction. In *Proceedings of the International Conference on Machine Learning*, 2019.

Abbas Abdolmaleki, Jost Tobias Springenberg, Jonas Degrave, Steven Bohez, Yuval Tassa, Dan Belov, Nicolas Heess, and Martin Riedmiller. Relative Entropy Regularized Policy Iteration. *arXiv preprint arXiv:1812.02256*, 2018a.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a Posteriori Policy Optimisation. In *Proceedings of the International Conference on Learning Representations*, 2018b.

Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research*, 2021.

Carlo Alfano, Rui Yuan, and Patrick Rebeschini. A Novel Framework for Policy Mirror Descent with General Parameterization and Linear Convergence. *Proceedings of the International Conference on Neural Information Processing Systems*, 2023.

Navid Azizan, Sahin Lale, and Babak Hassibi. Stochastic Mirror Descent on Overparameterized Nonlinear Models. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

Amir Beck. *First-Order Methods in Optimization*. Society for Industrial and Applied Mathematics, 2017.

Sébastien Bubeck. Convex Optimization: Algorithms and Complexity. *Foundations and Trends in Machine Learning*, 2015.

Thomas Degris, Patrick M. Pilarski, and Richard S. Sutton. Model-Free Reinforcement Learning with Continuous Action in Practice. In *Proceedings of the American Control Conference*, 2012.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G. Bellemare, and Aaron Courville. Sample-Efficient Reinforcement Learning by Breaking the Replay Ratio Barrier. In *Proceedings of the International Conference on Learning Representations*, 2023.

Theresa Eimer, Marius Lindauer, and Raileanu Roberta. Hyperparameters in Reinforcement Learning and How to Tune Them. In *Proceedigs of the International Conference on Machine Learning*, 2023.

Ilyas Fatkhullin and Niao He. Taming nonconvex stochastic mirror descent with general Bregman divergence. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2024.

Matthieu Geist, Bruno Scherrer, and Olivier Pietquin. A theory of regularized markov decision processes. In *Proceedings of the International Conference on Machine Learning*, 2019.

Suriya Gunasekar, Blake Woodworth, and Nathan Srebro. Mirrorless Mirror Descent: A Natural Derivation of Mirror Descent. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2021.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proceedings of the International Conference on Machine Learning*, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications, 2019.

Emmeran Johnson, Ciara Pike-Burke, and Patrick Rebeschini. Optimal Convergence Rate for Exact Policy Mirror Descent in Discounted Markov Decision Processes. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2023.

Scott M Jordan, Samuel Neumann, James E Kostas, Adam White, and Philip S Thomas. The Cliff of Overcommitment with Policy Gradient Step Sizes. *Reinforcement Learning Conference*, 2024.

Sham Kakade and John Langford. Approximately Optimal Approximate Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, 2002.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, 2014.

Guanghui Lan. Policy mirror descent for reinforcement learning: Linear convergence, new sampling complexity, and generalized problem classes. *Mathematical Programming*, 2023.

Nevena Lazić, Botao Hao, Yasin Abbasi-Yadkori, Dale Schuurmans, and Csaba Szepesvári. Optimization Issues in KL-Constrained Approximate Policy Iteration. *arXiv preprint arXiv:2102.06234*, 2021.

Jincheng Mei, Chenjun Xiao, Ruitong Huang, Dale Schuurmans, and Martin Müller. On Principled Entropy Exploration in Policy Optimization. In *International Joint Conference on Artificial Intelligence*, 2019.

Gergely Neu, Anders Jonsson, and Vicenç Gómez. A Unified View of Entropy-Regularized Markov Decision Processes. *arXiv preprint arXiv:1705.07798*, 2017.

Samuel Neumann, Sungsu Lim, Ajin Joseph, Yangchen Pan, Adam White, and Martha White. Greedy Actor-Critic: A New Conditional Cross-Entropy Method for Policy Improvement. In *Proceedings of the International Conference on Learning Representations*, 2023.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The Primacy Bias in Depp Reinforcement Learning. In *Proceedings of the International Conference on Machine Learning*, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

Jan Peters, Katharina Mulling, and Yasemin Altun. Relative Entropy Policy Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

Garvesh Raskutti and Sayan Mukherjee. The Information Geometry of Mirror Descent. *IEEE Transactions on Information Theory*, 2015.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the International Conference on Machine Learning*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ryan Sullivan, Jordan K Terry, Benjamin Black, and John P Dickerson. Cliff Diving: Exploring Reward Surfaces in Reinforcement Learning Environments. In *Proceedings of the 39th International Conference on Machine Learning*, 2022.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Proceedings of the International Conference on Neural Information Processing Systems*, 1999.

Manan Tomar, Lior Shani, Yonathan Efroni, and Mohammad Ghavamzadeh. Mirror Descent Policy Optimization. In *Proceedings of the International Conference on Learning Representations*, 2022.

Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, 2023. URL https://zenodo.org/record/8127025.

Sharan Vaswani, Olivier Bachem, Simone Totaro, Robert Mueller, Shivam Garg, Matthieu Geist, Marlos C. Machado, Pablo Samuel Castro, and Nicolas Le Roux. A General Class of Surrogate Functions for Stable and Efficient Reinforcement Learning. In *International Conference on Artificial Intelligence and Statistics*, 2022.

Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Rémi Munos, and Matthieu Geist. Leverage the Average: An Analysis of Regularization in RL. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020a.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen Reinforcement Learning. In *Proceedings of the International Conference on Neural Information Processing Systems*, 2020b.

Nino Vieillard, Marcin Andrychowicz, Anton Raichuk, Olivier Pietquin, and Matthieu Geist. Implicitly Regularized RL with Implicit Q-Values. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, 2022.

Ronald J Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 1992.

Lin Xiao. On the Convergence Rates of Policy Gradient Methods. In *Journal of Machine Learning Research*, 2022.

Zhihan Xiong, Maryam Fazel, and Lin Xiao. Dual Approximation Policy Optimization. In *Proceedings of the ICML Workshop on Aligning Reinforcement Learning Experimentalists and Theorists*, 2024.

Lingwei Zhu, Zheng Chen, Matthew Schlegel, and Martha White. Generalized Munchausen Reinforcement Learning using Tsallis KL Divergence. *arXiv preprint arXiv:2301.11476*, 2023.

Supplementary Materials

The following content was not necessarily subject to peer review.

6 Mirror Descent

Mirror Descent (MD) is an iterative optimization algorithm which generalizes (sub)gradient descent; in contrast to gradient descent, mirror descent better considers the geometry of the input space (Raskutti & Mukherjee, 2015; Bubeck, 2015; Beck, 2017). Further, by clearly distinguishing primal and dual spaces, MD addresses the philosophical issue of combining vectors in different spaces during gradient updates. Although the derivation of mirror descent can be found in a multitude of other sources, we provide a brief derivation and discussion here for completeness, focusing on smooth optimization alone. This section is organized as follows. We first briefly summarize mirror descent. We refer the reader to Bubeck's comprehensive textbook on convex optimization (Bubeck, 2015) – from which we extensively adapt the proceeding sections – for a more detailed discussion of mirror descent. We then derive mirror descent on the simplex with a negative entropy mirror map. Finally, we explore the connections between the *lift-step-project* form of mirror descent and its commonly used *proximal* form.

Let us first define some terminology. In the following sections, we refer to an input $\boldsymbol{x}$ as being in some *primal* space. Following the notation of Bubeck (2015), we use a *hatted* symbol (e.g. $\hat{\boldsymbol{x}}$) to refer to the same point after transforming it to the associated dual space. For some set $\mathcal{S}$, denote $\partial\mathcal{S}$ as the boundary of $\mathcal{S}$ and define the closure of $\mathcal{S}$ as $\overline{\mathcal{S}} \doteq \mathcal{S} \cup \partial\mathcal{S}$. Denote as $\mathbb{R}^n_{++}$ the subset of $\mathbb{R}^n$ consisting of vectors whose elements are positive real numbers:

$$\mathbb{R}^n_{++} \doteq \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n,\ x_i > 0,\ \forall i \leq n \in \mathbb{Z}\}$$

We similarly defined $\mathbb{R}^n_{+} = \mathbb{R}^n_{++} \cup \{\mathbf{0}\}$ as the set of non-negative real numbers.

Consider an objective function which we would like to optimize $J : \mathcal{X} \to \mathbb{R}$ such that $\mathcal{X} \subseteq \mathbb{R}^n$. For gradient-descent-style optimization algorithms, an optimization step at iteration t is taken by subtracting from input $\boldsymbol{x}_t$ a scaled gradient of J evaluated at $\boldsymbol{x}_t$:

$$\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t - \lambda \boldsymbol{g}_t \quad \text{for some } \lambda \in \mathbb{R}$$

where $\boldsymbol{g}_t = \nabla J(\boldsymbol{x}_t)$. The gradient $\boldsymbol{g}_t$ does not necessarily lie in the same space as the input $\boldsymbol{x}_t$. While $\boldsymbol{x}_t \in \mathcal{X}$, $\boldsymbol{g}_t$ is in the dual space (we direct the reader to Beck's textbook for further explanation (Beck, 2017)). As an example, consider the case when $\mathcal{X}$ is the simplex and $J(\boldsymbol{x}) = \langle \boldsymbol{x}, \ln(\boldsymbol{x}) \rangle$. Then $\nabla J(\boldsymbol{x}) = \ln(\boldsymbol{x}) - 1$ which may not lie in $\mathcal{X}$.

The Lift Procedure The *lift* procedure of MD consists of *lifting* the input to the dual space, the same space that the gradient of J lies is. To do this, we introduce a *mirror map*. Let $\mathcal{D}$ be an open, convex set such that $\mathcal{X} \subset \overline{D}$ and $\mathcal{X} \cap \mathcal{D} \neq \varnothing$. We say that $\phi : \mathcal{D} \to \mathbb{R}$ is a *mirror map* if it satisfies the following conditions (Bubeck, 2015):

- **Convexity and Differentiability**: ϕ is strictly convex and differentiable
- **Gradient Surjectivity**: $\nabla\phi : \mathcal{D} \to \mathbb{R}^n$ is surjective
- **Gradient Divergence**: $\lim_{\boldsymbol{x} \to \partial\mathcal{D}} \|\nabla\phi(\boldsymbol{x})\| = \infty$

This is the common characterization of mirror maps (Bubeck, 2015) but note that we will later require $\nabla\phi$ to be injective on its domain.

The *lift* procedure then consists of the transformation:

$$\hat{\boldsymbol{x}}_t \leftarrow \nabla\phi(\boldsymbol{x}_t) \tag{11}$$

The Step Procedure The *step* procedure consists of taking a step in the dual space:

$$\hat{\boldsymbol{y}}_{t+1} \leftarrow \hat{\boldsymbol{x}}_t - \lambda \boldsymbol{g}_t \tag{12}$$

where $\lambda \in \mathbb{R}$ is a stepsize parameter.

The Project Procedure The *project* procedure consists of transforming $\hat{\boldsymbol{y}}_{t+1}$ back to the primal set $\mathcal{X}$. To begin, $\hat{\boldsymbol{y}}_{t+1}$ is transformed back to $\mathcal{D}$ using the inverse gradient of the mirror map. That is

$$\boldsymbol{y}_{t+1} \leftarrow (\nabla\phi)^{-1}(\hat{\boldsymbol{y}}_{t+1}) \tag{13}$$

This procedure is often computed using the Legendre-Fenchel transform. For a function $f : \mathcal{X} \to \mathbb{R}$, the Legendre-Fenchel transform (or *convex conjugate*) of f is defined to be

$$f^*(\boldsymbol{x}^*) \doteq \sup_{\boldsymbol{x}\in\mathcal{X}} \left(\langle \boldsymbol{x}^*, \boldsymbol{x}\rangle - f(\boldsymbol{x})\right) \tag{14}$$

A useful property of the Legendre-Fenchel transformation of f is that its gradient is the inverse of the gradient of f, that is:

$$\begin{aligned} \nabla f^*(\nabla f(\boldsymbol{x})) &= \boldsymbol{x} \\ \nabla f(\nabla f^*(\boldsymbol{x})) &= \boldsymbol{x} \end{aligned} \tag{15}$$

Therefore, if we are able to compute the gradient of the Legendre-Fenchel transform of ϕ, we can use it to compute Equation 13:

$$\boldsymbol{y}_{t+1} \leftarrow \nabla\phi^*(\hat{\boldsymbol{y}}_{t+1}) \tag{16}$$

To project $\boldsymbol{y}_{t+1} \in \mathcal{D}$ back to the constraint set $\mathcal{X}$ we employ a projection operator $\Pi_\phi^{\mathcal{X}} : \mathcal{D} \to \mathcal{X}$:

$$\Pi_\phi^{\mathcal{X}}(\boldsymbol{y}) \doteq \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}} \; D_\phi(\boldsymbol{x}, \boldsymbol{y}) \tag{17}$$

where D_ϕ is the Bregman divergence in ϕ. To complete the mirror descent update, we simply apply this projection to $\boldsymbol{y}_{t+1}$:

$$\boldsymbol{x}_{t+1} \in \Pi_\phi^{\mathcal{X}}(\boldsymbol{y}_{t+1}) \tag{18}$$

General Form Based on the preceding discussion, mirror descent has the following general form:

$$\begin{aligned} &\boldsymbol{y}_{t+1} = \nabla\phi^*(\nabla\phi(\boldsymbol{x}_t) - \lambda \boldsymbol{g}_t) \text{ where } \boldsymbol{g}_t \in \partial J(\boldsymbol{x}_t) && \text{from (11, 12, 13, 16)} \\ &\boldsymbol{x}_{t+1} \in \Pi_\phi^{\mathcal{X}}(\boldsymbol{y}_{t+1}) && \text{from (18)} \end{aligned} \tag{19}$$

where $\boldsymbol{x}_1 = \arg\min_{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}} \phi(\boldsymbol{x})$.

6.1 Deriving Mirror Descent on the Simplex

Given a mirror map ϕ, the *lift* and *step* operations are immediately defined. For the *projection* step, we must derive the inverse of the *lift* step. This requires deriving:

1. Inverse of mirror map gradient: $(\nabla\phi)^{-1}$
2. Projection operator: $\Pi_\phi^{\mathcal{X}}(\boldsymbol{y}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathcal{D}} \; D_\phi(\boldsymbol{x}, \boldsymbol{y})$ for some $y \in \mathcal{D}$

We will use the negative entropy mirror map, defined as:

$$\phi(\boldsymbol{x}) = \langle \boldsymbol{x}, \ln \boldsymbol{x}\rangle = \sum_{i=1}^{n} x_i \ln(x_i) \tag{20}$$

where $\mathcal{D} = \mathbb{R}_+^n$. We use the convention that $0 \log 0 = 0$.

In Section 6.1.1, we derive the Legendre-Fenchel transform of the negative entropy mirror map and use it to compute the inverse of the mirror map gradient for Point (1) above. Next, in Section 6.1.2 and 6.1.3, we derive the projection operator (2).

6.1.1 Legendre-Fenchel Transform of the Negative Entropy Mirror Map

Recall the definition of the Legendre-Fenchel transform of ϕ:

$$\phi^*(\boldsymbol{x}^*) = \sup_{\boldsymbol{x}\in\mathbb{R}^n_+} \left(\langle \boldsymbol{x}, \boldsymbol{x}^*\rangle - \phi(\boldsymbol{x})\right) \tag{21}$$

$$= \sup_{\boldsymbol{x}\in\mathbb{R}^n_+} \left(\langle \boldsymbol{x}, \boldsymbol{x}^*\rangle - \langle \boldsymbol{x}, \ln(\boldsymbol{x})\rangle\right) \tag{22}$$

$$= \sup_{\boldsymbol{x}\in\mathbb{R}^n_+} f(\boldsymbol{x}) \tag{23}$$

where f is defined implicitly. Notice that

$$\begin{aligned}\nabla f(\boldsymbol{x}) &= \boldsymbol{x}^* - \nabla\phi(\boldsymbol{x}) \\ &= \boldsymbol{x}^* - \nabla\langle \boldsymbol{x}, \ln(\boldsymbol{x})\rangle \\ &= \boldsymbol{x}^* - \ln(\boldsymbol{x}) - 1\end{aligned}$$

Clearly this objective function has a unique optimum. This optimum either lies on the boundary of $\mathbb{R}^n_+$ (where gradients are undefined) or else it lies in the interior of $\mathbb{R}^n_+$. We will seek a solution in the interior of $\mathbb{R}^n_+$. Upon obtaining such a solution, the uniqueness property ensures that the optimum indeed lies in the interior of $\mathbb{R}^n_+$. Setting $\nabla f(\boldsymbol{x}) = 0$ and solving provides:

$$\boldsymbol{x} = e^{\boldsymbol{x}^*-1} \tag{24}$$

substituting $\boldsymbol{x}$ back into Equation 22 we get

$$\phi^*(\boldsymbol{x}^*) = \langle e^{\boldsymbol{x}^*-1}, \boldsymbol{x}^*\rangle - \langle e^{\boldsymbol{x}^*-1}, \boldsymbol{x}^* - 1\rangle = \mathbf{1}^\top e^{\boldsymbol{x}^*-1} \tag{25}$$

where $\mathbf{1} \in \mathbb{R}^n$ is the vector whose elements are all 1. Hence $\phi^*(\boldsymbol{x}^*) = \mathbf{1}^\top e^{\boldsymbol{x}^*-1}$ and $\nabla\phi^*(\boldsymbol{x}^*) = e^{\boldsymbol{x}^*-1}$

6.1.2 Bregman Divergence of the Negative Entropy

Similarly to above, let $\phi(x) = \langle \boldsymbol{x}, \ln \boldsymbol{x}\rangle$. The Bregman divergence of ϕ is

$$D_\phi(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x}) - \phi(\boldsymbol{y}) - \langle \nabla\phi(\boldsymbol{y}), (\boldsymbol{x} - \boldsymbol{y})\rangle \tag{26}$$

$$= \langle \boldsymbol{x}, \ln(\boldsymbol{x})\rangle - \langle \boldsymbol{y}, \ln(\boldsymbol{y})\rangle - \langle \ln(\boldsymbol{y}) + 1, \boldsymbol{x}\rangle + \langle \ln(\boldsymbol{y}) + 1, \boldsymbol{y}\rangle \tag{27}$$

$$= \sum_{i=1}^n x_i \ln\left(\frac{x_i}{y_i}\right) - \sum_{i=1}^n x_i + \sum_{i=1}^n y_i \tag{28}$$

$$\tag{29}$$

This last equation is known as the *generalized KL divergence* and is exactly equal to the KL divergence when $\boldsymbol{x}$ and $\boldsymbol{y}$ lie on the simplex.

6.1.3 Projection onto the Constraint Set

Let $\mathcal{X} \subset \mathbb{R}^n_+$ denote the probability simplex over $\mathbb{R}^n$:

$$\mathcal{X} \doteq \left\{ \boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^N,\ x_i \geq 0,\ \sum_{i=1}^n x_i = 1 \right\} \tag{30}$$

Let $y \in \mathbb{R}^n_{++}$. Let $\Pi^{\mathcal{X}}_\phi$ denote the projection operator, projecting vectors in $\mathbb{R}^n_{>0}$ onto the simplex $\mathcal{X}$ according to:

$$\Pi^{\mathcal{X}}_\phi(\boldsymbol{y}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathbb{R}^n_+} D_\phi(\boldsymbol{x}, \boldsymbol{y}) = \operatorname*{arg\,min}_{\boldsymbol{x}\in\mathcal{X}\cap\mathbb{R}^n_+} \sum_{i=1}^n x_i \ln\left(\frac{x_i}{y_i}\right) \tag{31}$$

We will now derive the closed form equation for this projection operator. To begin, recall Jensen's inequality for some convex ψ:

$$\psi\left(\frac{\sum_i a_i x_i}{\sum_i a_i}\right) \leq \frac{\sum_i a_i \psi(x_i)}{\sum_i a_i} \quad \text{for } a_i > 0 \; \forall i$$

and $\psi(\sum_i x_i) \leq \sum_i \psi(x_i)$ when $a_i = 1 \; \forall i$. Letting $f(x) = x\ln(x)$, we can rewrite the argument to the optimization problem in Equation 31 as:

$$\sum_{i=1}^{n} x_i \ln\left(\frac{x_i}{y_i}\right) = \sum_{i=1}^{n} y_i f\left(\frac{x_i}{y_i}\right)$$

Using Jensen's inequality, we get that:

$$\frac{\sum_{i=1}^{n} y_i f\left(\frac{x_i}{y_i}\right)}{\|\boldsymbol{y}\|_1} \geq f\left(\frac{\sum_{i=1}^{n} y_i \frac{x_i}{y_i}}{\|\boldsymbol{y}\|_1}\right) \implies \sum_{i=1}^{n} y_i f\left(\frac{x_i}{y_i}\right) \geq \|\boldsymbol{y}\|_1 \, f\left(\frac{\sum_{i=1}^{n} x_i}{\|\boldsymbol{y}\|_1}\right)$$

Based on the conditions of Jensen's inequality, we know that equality holds if and only if $\frac{x_j}{y_j} = \frac{x_i}{y_i} \; \forall i, j$. This condition is satisfied if $x_i = \frac{y_i}{\|\boldsymbol{y}\|_1}$, indicating that:

$$\underset{\mathcal{X}\cap\mathbb{R}^n_+}{\arg\min} \; D_\phi(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{y}}{\|\boldsymbol{y}\|_1}$$

6.2 Relationship to the Proximal Form of Mirror Descent

It may not be immediately clear how the *lift-step-project* form of mirror descent is related to the *proximal* form of mirror descent. The proximal form is defined as:

$$\underset{\boldsymbol{x}\in\mathcal{X}\cap\mathbb{R}^n_+}{\arg\min} \; \left(\langle \boldsymbol{x}, \boldsymbol{g}_t\rangle + \tfrac{1}{\lambda} D_\phi(\boldsymbol{x}, \boldsymbol{x}_t)\right) \text{ where } \boldsymbol{g}_t \in \partial J(\boldsymbol{x}_t) \tag{32}$$

Notice that we can re-write the general *lift-step-project* form of mirror descent from Equation 19 as:

$$\begin{aligned}
\boldsymbol{x}_{t+1} &= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; D_\phi(\boldsymbol{x}, \boldsymbol{y}_{t+1}) \\
&= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; \phi(\boldsymbol{x}) - \langle \nabla\phi(\boldsymbol{y}_{t+1}), \boldsymbol{x} - \boldsymbol{y}_{t+1}\rangle \\
&= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; \phi(\boldsymbol{x}) - \langle \nabla\phi(\boldsymbol{y}_{t+1}), \boldsymbol{x}\rangle \\
&= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; \phi(\boldsymbol{x}) - \langle \nabla\phi(\boldsymbol{x}_t) - \lambda \boldsymbol{g}_t, \boldsymbol{x}\rangle && \triangleright \text{ from (15)} \\
&= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; \langle \boldsymbol{x}, \lambda \boldsymbol{g}_t\rangle + D_\phi(\boldsymbol{x}, \boldsymbol{x}_t) \\
&= \underset{\mathcal{X}\cap\mathbb{R}^n_{>0}}{\arg\min} \; \langle \boldsymbol{x}, \boldsymbol{g}_t\rangle + \tfrac{1}{\lambda} D_\phi(\boldsymbol{x}, \boldsymbol{x}_t)
\end{aligned}$$

From here it is clear what exactly mirror descent is doing. It is minimizing the first order approximation of the function J while staying in close proximity (as determined by λ) to the previous point as measured by the Bregman divergence in ϕ.

As previously shown, the Bregman divergence in the negative entropy mirror map is the generalized KL divergence. Hence, the proximal form of mirror ascent as studied in this paper is defined by the iterative equation:

$$\boldsymbol{x}_{t+1} = \underset{\mathcal{X}\cap\mathbb{R}^n_+}{\arg\max} \; \langle \boldsymbol{x}, \nabla J(\boldsymbol{x})\rangle - \tfrac{1}{\lambda}\mathrm{KL}(\boldsymbol{x} \,||\, \boldsymbol{x}_t) \tag{33}$$

where KL is the generalized KL divergence on the simplex and J is some policy performance function.

7 Hyperparameter Tuning

In this section, we list the ranges of hyperparameters swept for our analysis in Section 4.1 along with the tuned hyperparameters for each algorithm. Table 1 lists the tuned hyperparameters across all continuous-action classic control environments for each algorithm tested. Table 2 lists the tuned hyperparameters across all MuJoCo environments. Finally, Table 6 lists the tuned hyperparameters across all discrete-action environments.

We swept the following hyperparameters. Replay buffers held all transitions, with batch sizes of 32 (256 for MuJoCo). We swept critic step sizes $\alpha_{\text{critic}} = 10^y$ for $y \in \{-4, -3, -2\}$ and actor stepsizes $\alpha_{\text{actor}} = 10^{\varsigma_{\text{actor}}}\alpha_{\text{critic}}, \varsigma_{\text{actor}} \in \{-1, 0, 1\}$. The likelihood ratio in the gradient for MD-GreedyAC with an RKL penalty caused numerical instabilities, hence we found it crucial to clip this likelihood ratio. We clipped between $\left[(1+\varepsilon)^{-1}, 1+\varepsilon\right]$ and set $\varepsilon = 0.7$ in all experiments as we found the value to work well. We swept entropy scales $\tau = 10^t$ for $t \in \{-4, -3, -2, -1, 0\}$. For MD updates, the two extreme entropy scales were excluded. For GreedyAC, the smallest τ was replaced with 0. Soft action-value functions were used. MPO did not use entropy regularization, instead we swept the KL policy coefficient $\kappa = 10^{t-1}$, excluding the two lowest κ values for MD-MPO. For MD updates, we swept $\lambda \in \{\pm 3, \pm 2, \pm 1, 0\}$, setting $M = 10$ unless otherwise explicitly stated. For GreedyAC, we set $\rho_{\text{actor}} = 0.1$, $\rho_{\text{proposal}} = 0.2$, and $n = 10$. Both policies used entropy regularization. We used target critic networks for learning action-value critics with a Sarsa update. The target networks were updated each timestep using a polyak average with constant $\beta = 0.01$

$$\boldsymbol{w}_{\text{target}} = (1-\beta)\boldsymbol{w}_{\text{target}} + \beta \boldsymbol{w} \tag{34}$$

where $\boldsymbol{w}$ are the parameters for the critic network and $\boldsymbol{w}_{\text{target}}$ are the parameters for the target critic network.

On the MuJoCo environments, we reduced our hyperparameter sweep and only used SAC. We swept the critic stepsize 10^y and entropy scale 10^{y+1} for $y \in \{-5, -4, \ldots, -1\}$ as we found SAC to be most sensitive to these hyperparameters. We set $\varsigma_{\text{actor}} = 10^{-1}$ and $\lambda = 10$.

Table 1: Tuned Hyperparameters across environments for continuous-action algorithms

Algorithm \ Hyper	α_{critic}	ς_{actor}	τ	κ	λ
GreedyAC	10^{-3}	1	10^{-1}	-	-
MD-GreedyAC (RKL)	10^{-3}	1	10^{-2}	-	1
MD-GreedyAC (FKL)	10^{-3}	1	0	-	1
SAC	10^{-3}	1	10^{-3}	-	-
MD-SAC (RKL)	10^{-3}	1	10^{-3}	-	10^2
MD-SAC (FKL)	10^{-3}	1	10^{-3}	-	10^1
MPO	10^{-3}	1	-	10^{-5}	-
MD-MPO (RKL)	10^{-2}	-1	0	10^{-4}	10^{-1}
MD-MPO (FKL)	10^{-3}	1	0	10^{-4}	10^{-1}

Table 2: Tuned Hyperparameters across environments for the MuJoCo suite

Algorithm \ Hyper	α_{critic}	ς_{actor}	τ	λ
SAC	10^{-3}	-1	10^{-2}	-
MD-SAC (RKL)	10^{-3}	-1	10^{-2}	10^1
MD-SAC (FKL)	10^{-3}	-1	1	10^1

Table 3: Tuned Hyperparameters for Continuous-Action Algorithms

Environment	GreedyAC			MPO			SAC		
	α_{critic}	ς_{actor}	τ	α_{critic}	ς_{actor}	κ	α_{critic}	ς_{actor}	τ
Acrobot	10^{-3}	0	10^{-2}	10^{-2}	-1	10^{-5}	10^{-3}	0	10^{-4}
MountainCar	10^{-2}	-1	10^{-1}	10^{-3}	0	10^{-5}	10^{-3}	0	10^{-3}
Pendulum	10^{-3}	0	10^{-1}	10^{-2}	-1	10^{-3}	10^{-2}	-1	10^{-2}

Table 4: Tuned Hyperparameters for Continuous-Action PMD Algorithms with an RKL Penalty

Environment	PMD GreedyAC/Clipping				PMD MPO				PMD SAC			
	α_{critic}	ς_{actor}	λ	τ	α_{critic}	ς_{actor}	λ	κ	α_{critic}	ς_{actor}	λ	τ
Acrobot	$10^{-4}/10^{-3}$	$-2/-1$	$10^{-1}/10^{2}$	$10^{-2}/10^{-2}$	10^{-2}	-1	10^{3}	10^{-3}	10^{-2}	-1	10^{1}	10^{-3}
MountainCar	$10^{-2}/10^{-3}$	$-1/0$	$10^{-1}/10^{0}$	$10^{-1}/0$	10^{-3}	0	10^{-1}	10^{-3}	10^{-3}	0	10^{2}	10^{-3}
Pendulum	$10^{-2}/10^{-3}$	$-2/0$	$10^{0}/10^{0}$	$10^{-1}/10^{-2}$	10^{-2}	-1	10^{1}	10^{-3}	10^{-2}	-1	10^{1}	10^{-2}

Table 5: Tuned Hyperparameters for Continuous-Action PMD Algorithms with an FKL Penalty

Environment	PMD GreedyAC				PMD MPO				PMD SAC			
	α_{critic}	ς_{actor}	λ	τ	α_{critic}	ς_{actor}	λ	κ	α_{critic}	ς_{actor}	λ	τ
Acrobot	10^{-2}	-1	10^{-1}	10^{-2}	10^{-2}	-1	10^{0}	10^{-3}	10^{-2}	-1	10^{1}	10^{-2}
MountainCar	10^{-3}	0	10^{0}	0	10^{-3}	0	10^{0}	10^{-3}	10^{-3}	0	10^{2}	10^{-2}
Pendulum	10^{-2}	0	10^{0}	0	10^{-2}	-1	10^{0}	10^{-2}	10^{-2}	-1	10^{1}	10^{-1}

Table 6: Tuned Hyperparameters across environments for discrete-action algorithms

Algorithm \ Hyper	α_{critic}	ς_{actor}	τ	κ	λ
GreedyAC	10^{-3}	1	10^{-3}	-	-
MD-GreedyAC (RKL)	10^{-3}	1	0	-	10^{-1}
MD-GreedyAC (FKL)	10^{-3}	1	10^{-2}	-	1
SAC	10^{-3}	1	10^{-2}	-	-
MD-SAC (RKL)	10^{-3}	1	10^{-2}	-	10^{2}
MD-SAC (FKL)	10^{-3}	1	10^{-2}	-	10^{2}
MPO	10^{-3}	1	-	10^{-2}	-
MD-MPO (RKL)	10^{-3}	1	0	10^{-2}	1
MD-MPO (FKL)	10^{-3}	1	0	10^{-3}	10^{2}

Table 7: Tuned Hyperparameters for Discrete-Action Algorithms

Environment	GreedyAC			MPO			SAC		
	α_{critic}	ς_{actor}	τ	α_{critic}	ς_{actor}	κ	α_{critic}	ς_{actor}	τ
Acrobot	10^{-3}	0	0	10^{-3}	0	10^{-2}	10^{-2}	-1	10^{-2}
MountainCar	10^{-3}	0	10^{-2}	10^{-3}	0	10^{-2}	10^{-3}	0	10^{-2}
Pendulum	10^{-2}	-1	10^{-1}	10^{-2}	-1	10^{-4}	10^{-2}	-1	10^{0}

Table 8: Tuned Hyperparameters for Discrete-Action PMD Algorithms with an RKL Penalty

	PMD GreedyAC				PMD MPO				PMD SAC			
Environment	α_{critic}	ς_{actor}	λ	τ	α_{critic}	ς_{actor}	λ	κ	α_{critic}	ς_{actor}	λ	τ
Acrobot	10^{-2}	-1	10^{1}	0	10^{-3}	0	10^{2}	10^{-1}	10^{-2}	-1	10^{-1}	10^{-2}
MountainCar	10^{-3}	0	10^{-1}	0	10^{-3}	0	10^{-1}	10^{-2}	10^{-3}	0	10^{2}	10^{-2}
Pendulum	10^{-2}	-1	10^{-1}	10^{-1}	10^{-2}	-1	10^{0}	10^{-3}	10^{-2}	-1	10^{1}	10^{-1}

Table 9: Tuned Hyperparameters for Discrete-Action PMD Algorithms with an FKL Penalty

	PMD GreedyAC				PMD MPO				PMD SAC			
Environment	α_{critic}	ς_{actor}	λ	τ	α_{critic}	ς_{actor}	λ	κ	α_{critic}	ς_{actor}	λ	τ
Acrobot	10^{-3}	0	10^{2}	0	10^{-3}	1	10^{0}	10^{-1}	10^{-2}	-1	10^{1}	10^{-2}
MountainCar	10^{-3}	0	10^{-2}	10^{-2}	10^{-3}	0	10^{-1}	10^{-1}	10^{-3}	0	10^{0}	10^{-2}
Pendulum	10^{-2}	0	10^{-1}	10^{-2}	10^{-2}	-1	10^{3}	10^{-3}	10^{-2}	-1	10^{-2}	10^{-1}

8 Functional Mirror Decent Algorithm Derivations

In this section, we provide mathematical proofs for each of the algorithm extensions to the functional mirror descent framework.

8.1 Preliminary Results

Lemma 1 (Gradient of the Negative Shannon Entropy)**.** *The gradient of the negative Shannon entropy is* $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right] = \mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[\ln \pi_{\boldsymbol{\theta}}(a \mid s)\nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a \mid s)\right]$

Proof. We begin the proof by differentiating the argument to the expectation in integral form $\mathbb{E}_{\pi_{\boldsymbol{\theta}}}\left[-\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right] = -\int_{\mathcal{A}} \pi_{\boldsymbol{\theta}}(a \mid s) \ln(\pi_{\boldsymbol{\theta}}(a \mid s))\, da$. We get:

$$\begin{aligned}\nabla[\pi_{\boldsymbol{\theta}}(a|s) \ln \pi_{\boldsymbol{\theta}}(a|s)] &= \ln \pi_{\boldsymbol{\theta}}(a|s)\nabla\pi_{\boldsymbol{\theta}}(a|s) + \pi_{\boldsymbol{\theta}}(a|s)\nabla \ln \pi_{\boldsymbol{\theta}}(a|s) \\ &= \ln \pi_{\boldsymbol{\theta}}(a|s)\nabla\pi_{\boldsymbol{\theta}}(a|s) + \pi_{\boldsymbol{\theta}}(a|s)\frac{1}{\pi_{\boldsymbol{\theta}}(a|s)}\nabla\pi_{\boldsymbol{\theta}}(a|s) = (\ln \pi_{\boldsymbol{\theta}}(a|s) + 1)\nabla\pi_{\boldsymbol{\theta}}(a|s)\end{aligned}$$

Again using the log-likelihood trick, we get

$$\begin{aligned}\nabla\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[\ln \pi_{\boldsymbol{\theta}}(a|s)] &= \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[\ln \pi_{\boldsymbol{\theta}}(a|s)\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)] + \nabla\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[1] \\ &= \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[\ln \pi_{\boldsymbol{\theta}}(a|s)\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)].\end{aligned}$$

□

Lemma 2 (Bregman Divergence in Log-Sum-Exp)**.** *Let* ϕ *be the log-sum-exp function and* $\boldsymbol{z}$*,* $\boldsymbol{z}'$ *be the logits of two softmax distributions* p *and* p' *respectively. Then* $D_\phi(\boldsymbol{z}, \boldsymbol{z}') = KL(p' \mid\mid p)$*.*

Proof. The proof is given by Vaswani et al. (2022). □

8.2 Functional Mirror Descent for Soft Actor-Critic

In this section, we provide additional proofs for Mirror Descent Soft Actor-Critic not included in the main text.

Assume we have a softmax parameterization, $\pi_{\boldsymbol{\theta}}(a \mid s) \propto \exp(z_{\boldsymbol{\theta}}(a, s))$. Our network outputs $z_{\boldsymbol{\theta}}(a, s)$, and we consider the functional space over z instead. We prove the result for the discrete action setting, but the updates extend to the continuous action setting.

Proposition 3. *Assume we have a finite number of actions, $|\mathcal{A}|$, and use a softmax policy parameterization, $\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp(z_{\boldsymbol{\theta}}(a,s))}{\sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(a,s))}$. On time step t with current action-values q and policy π_t and logits z_t, the surrogate objective for Soft Actor-Critic with the log-sum-exp mirror map*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t(a,s)} - \tfrac{1}{\lambda})\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

with gradient

$$\nabla f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t(a,s)} - \tfrac{1}{\lambda})\nabla\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

where $v(s) \doteq \sum_{j=1}^{|\mathcal{A}|} q(s,j)\pi(j \mid s)$ and $\varsigma_{\pi_t}(a,s) = -q(s,a)+v(s)+\tau\ln\pi_t(a \mid s)+\tau\mathcal{H}(\pi_t(\cdot \mid s))$

Proof. Letting $\tilde{\ell}_{q,s}(\pi)$ equal that defined in the proof of Proposition 1, we can use the chain rule to get the partial derivatives in terms of the logits $z(a,s)$ instead.

$$\frac{\partial\tilde{\ell}_{q,s}}{\partial z(a,s)} = \sum_{j=1}^{|\mathcal{A}|}\frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(j \mid s)}\frac{\partial\pi(j \mid s)}{\partial z(a,s)} = \sum_{j=1}^{|\mathcal{A}|}(-q(s,j)+\tau(\ln\pi(j \mid s)+1))\frac{\partial\pi(j \mid s)}{\partial z(a,s)}$$

We can compute the second term considering two cases. Let $c(s) = \sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(j,s))$, with $\pi(a|s) = \exp(z(a,s))/c(s)$. If $a \neq j$, then we have

$$\frac{\partial\pi(j \mid s)}{\partial z(a,s)} = -\exp(z(j,s))c(s)^{-2}\exp(z(a,s)) = -\pi(j \mid s)\pi(a \mid s)$$

If $a = j$, then we have

$$\frac{\partial\pi(a \mid s)}{\partial z(a,s)} = \exp(z(a,s))/c(s) - \exp(z(a,s))c(s)^{-2}\exp(z(a,s)) = \pi(a \mid s)(1-\pi(a \mid s))$$

Plugging this in above, we get that

$$\sum_{j=1}^{|\mathcal{A}|}\frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(j \mid s)}\frac{\partial\pi(j \mid s)}{\partial z(a,s)} = \pi(a \mid s)(-q(s,a)+\tau\ln\pi(a \mid s))-\pi(a \mid s)\sum_{j=1}^{|\mathcal{A}|}(-q(s,j)+\tau\ln\pi(j \mid s))\pi(j \mid s) \tag{35}$$

Notice that

$$\sum_{j=1}^{|\mathcal{A}|}(-q(s,j)+\tau\ln\pi(j \mid s))\pi(j \mid s) = -\sum_{j=1}^{|\mathcal{A}|}q(s,j)\pi(j \mid s)+\tau\sum_{j=1}^{|\mathcal{A}|}\ln\pi(j \mid s)\pi(j \mid s) = -v(s)+\tau\mathcal{H}(\pi(\cdot \mid s))$$

and so finally we get that

$$(35) = \pi(a \mid s)\varsigma(a,s) \quad \text{for } \varsigma(a,s) \doteq -q(s,a)+v(s)+\tau\ln\pi(a \mid s)-\tau\mathcal{H}(\pi(\cdot \mid s))$$

Taking the inner product with $z_{\boldsymbol{\theta}}(\cdot|s)$, we get

$$\begin{aligned}\left\langle\frac{\partial\tilde{\ell}_{q,s}}{\partial z(\cdot,s)}(z_t), z_{\boldsymbol{\theta}}(\cdot,s)\right\rangle &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\varsigma(a,s)z_{\boldsymbol{\theta}}(a,s)\right] \\ &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\varsigma(a,s)\left(z_{\boldsymbol{\theta}}(a,s)-\sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(j,s))\right)\right] \\ &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\varsigma(a,s)\ln\pi_{\boldsymbol{\theta}}(a|s)\right]\end{aligned}$$

where the second line follows from the fact that $\mathbb{E}_{a\sim\pi_t(\cdot|s)}[\varsigma(a,s)] = 0$ and so we can shift the values of $z_{\boldsymbol{\theta}}(\cdot,s)$ without changing the expectation.

Finally, with a logsumexp mirror map, Lemma 2 shows that $D_\phi(\boldsymbol{z}^\pi, \boldsymbol{z}^{\pi_t}) = \text{KL}(\pi_t \mid\mid \pi)$, and so we get our surrogate objective

$$\begin{aligned} f_{q,s}(\boldsymbol{\theta}) &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\varsigma(a,s)\ln\pi_{\boldsymbol{\theta}}(a|s)\right] + \tfrac{1}{\lambda}\text{KL}(\pi_t(\cdot|s) \mid\mid \pi_{\boldsymbol{\theta}}(\cdot|s)) \\ &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[\varsigma(a,s)\ln\pi_{\boldsymbol{\theta}}(a|s) + \tfrac{1}{\lambda}\ln(\pi_t(a \mid s)) - \tfrac{1}{\lambda}\ln(\pi_{\boldsymbol{\theta}}(a \mid s))\right] \\ &= \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma(a,s) - \tfrac{1}{\lambda})\ln\pi_{\boldsymbol{\theta}}(a|s)\right] \end{aligned}$$

The gradient is simple to compute, because we are sampling from π_t,

$$\nabla f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma(a,s) - \tfrac{1}{\lambda})\nabla\ln\pi_{\boldsymbol{\theta}}(a|s)\right]$$

□

8.3 Functional Mirror Descent for Greedy Actor-Critic

In this section, we provide additional proofs for Mirror Descent Greedy Actor-Critic not included in the main text.

Greedy Actor-Critic minimizes a mode-covering KL between $\pi_{\boldsymbol{\theta}}$ and the percentile greedy policy $\pi^\rho_{\boldsymbol{\theta}}$ with entropy scale $\tau \geq 0$.

Proposition 4. *On time step t with current action-values q and policy π_t, the surrogate objective for Greedy Actor-Critic with a direct functional representation and any Bregman divergence D_ϕ, is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t\right] + \tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \tag{36}$$

with gradient

$$\begin{aligned} \nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a|s)}{\pi_t(a|s)}\right] &+ \tau\,\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t(a|s)\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(a|s)\right] + \\ &\nabla_{\boldsymbol{\theta}}\tfrac{1}{\lambda}D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \end{aligned} \tag{37}$$

where π^ρ is the percentile greedy policy in π_t.

Proof. We first define the loss in policy space, $\tilde{\ell}_{q,s}(\pi) \doteq \text{KL}(\pi^\rho(\cdot \mid s) \mid\mid \pi(\cdot \mid s)) = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}[-\ln\pi(a \mid s)] + \tau\mathbb{E}_{a\sim\pi(\cdot|s)}[\ln\pi(a \mid s)]$. Now we differentiate it

$$\frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(a \mid s)}(\pi) = \frac{-\pi^\rho(a \mid s)}{\pi(a \mid s)} + \tau(\ln\pi(a \mid s) + 1) \tag{38}$$

Taking the inner product with $\pi_{\boldsymbol{\theta}}(\cdot|s)$, we get

$$\left\langle \frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(a \mid s)}(\pi_t), \pi_{\boldsymbol{\theta}}(\cdot|s)\right\rangle = \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t(a \mid s)\right] \tag{39}$$

where the 1 disappears because $\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}[1] = 1$ and we ignore constants. Plugging this into Equation (4), we get Equation (36). Taking the gradient is straightforward. Using the log-likelihood trick we get:

$$\begin{aligned} \nabla_{\boldsymbol{\theta}}\left[\mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t\right]\right] = \\ \mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[-\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \tau\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_t(a \mid s)\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right] \end{aligned}$$

□

Corollary 2. *On time step t with current action-values q and policy π_t, the surrogate objective for Greedy Actor-Critic with a mode-seeking KL, namely $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = KL(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$, is*

$$f_{q,s}\boldsymbol{\theta} = -\mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{40}$$

with gradient

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} f_{q,s}\boldsymbol{\theta} = &-\mathbb{E}_{a\sim\pi^\rho(\cdot|s)}\left[\frac{\nabla_{\boldsymbol{\theta}}\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] \\ &+ \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left(\left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right)\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right]\end{aligned} \tag{41}$$

Proof. We simply need to plug-in $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = \mathrm{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$ into Equation (36). We have

$$\tfrac{1}{\lambda}\mathrm{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s)) = \tfrac{1}{\lambda}\mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln\pi_{\boldsymbol{\theta}}(a|s) - \ln\pi_t(a|s)\right]$$

Combining this $-\frac{1}{\lambda}\ln\pi_t(a|s)$ with the $\tau\ln\pi_t(a|s)$ term from the main loss, we get Equation (40).

The gradient is a straightforward calculation. Differentiating Equation (40) and using the log-likelihood trick with Lemma 1, yields the above gradient. □

Now let us consider the case where we have a softmax parameterization, $\pi_{\boldsymbol{\theta}}(a \mid s) \propto \exp(z_{\boldsymbol{\theta}}(a,s))$. Our network outputs $z_{\boldsymbol{\theta}}(a,s)$, and we consider the functional space over z instead.

Proposition 5. *Assume we have a finite number of actions, $\mid \mathcal{A} \mid$, and use a softmax policy parameterization, $\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp(z_{\boldsymbol{\theta}}(a,s))}{\sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(a,s))}$. On time step t with current action-values q and policy π_t and logits z_t, the surrogate objective for Greedy Actor-Critic with the log-sum-exp mirror map is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_t}\left[(\varsigma_{\pi_t^\tau} - \tfrac{1}{\lambda})\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{42}$$

with gradient

$$\nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_t}\left[(\varsigma_{\pi_t^\tau} - \tfrac{1}{\lambda})\nabla_{\boldsymbol{\theta}}\ln\pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{43}$$

where $\varsigma_{\pi^\tau(a,s)} = \frac{-\pi^\rho(a|s)}{\pi(a|s)} + 1 + \tau\ln\pi(a \mid s) + \tau\mathcal{H}(\pi(\cdot \mid s))$

Proof. Letting $\tilde{\ell}_{q,s}(\pi)$ equal that defined in the proof of Proposition 4, we can use the chain rule to get the partial derivatives in terms of the logits $z(a,s)$ instead.

$$\frac{\partial\tilde{\ell}_{q,s}}{\partial z(a,s)} = \sum_{j=1}^{|\mathcal{A}|}\frac{\partial\tilde{\ell}_{q,s}}{\partial\pi(j \mid s)}\frac{\partial\pi(j \mid s)}{\partial z(a,s)} = \sum_{j=1}^{|\mathcal{A}|}\left(\left[\frac{-\pi^\rho(j \mid s)}{\pi(j \mid s)}\right] + \tau(\ln\pi(j \mid s) + 1)\right)\frac{\partial\pi(j \mid s)}{\partial z(a,s)}$$

Similarly to Proposition 3, we can compute the second term considering two cases. Let $c(s) = \sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(j,s))$, with $\pi(a|s) = \exp(z(a,s))/c(s)$. If $a \neq j$, then we have

$$\frac{\partial\pi(j \mid s)}{\partial z(a,s)} = -\exp(z(j,s))c(s)^{-2}\exp(z(a,s)) = -\pi(j \mid s)\pi(a \mid s)$$

If $a = j$, then we have

$$\frac{\partial\pi(a \mid s)}{\partial z(a,s)} = \exp(z(a,s))/c(s) - \exp(z(a,s))c(s)^{-2}\exp(z(a,s)) = \pi(a \mid s)(1 - \pi(a \mid s))$$

Plugging this in above, we get that

$$\sum_{j=1}^{|\mathcal{A}|} \frac{\partial \tilde{\ell}_{q,s}}{\partial \pi(j \mid s)} \frac{\partial \pi(j \mid s)}{\partial z(a,s)} = \pi(a \mid s) \left(\frac{-\pi^{\rho}(a \mid s)}{\pi(a \mid s)} + \tau(\ln \pi(a \mid s) + 1) \right) - \pi(a \mid s) \sum_{j=1}^{|\mathcal{A}|} \left(\frac{-\pi^{\rho}(j \mid s)}{\pi(j \mid s)} + \tau(\ln \pi(j \mid s) + 1) \right) \pi(j \mid s) \tag{44}$$

Notice that

$$\sum_{j=1}^{|\mathcal{A}|} \left(\frac{-\pi^{\rho}(j \mid s)}{\pi(j \mid s)} + \tau(\ln \pi(j \mid s) + 1) \right) \pi(j \mid s) = -1 - \tau \mathcal{H}(\pi) + \tau$$

and so finally we get that

$$(44) = \pi(a \mid s) \varsigma_{\pi^{\tau}(a,s)} \quad \text{for } \varsigma_{\pi^{\tau}(a,s)} \doteq \frac{-\pi^{\rho}(a \mid s)}{\pi(a \mid s)} + 1 + \tau \ln \pi(a \mid s) + \tau \mathcal{H}(\pi(\cdot \mid s))$$

where π^{ρ} is the percentile greedy policy in π. Taking the inner product with $z_{\boldsymbol{\theta}}(\cdot|s)$, we get

$$\begin{aligned} \left\langle \frac{\partial \tilde{\ell}_{q,s}}{\partial z(\cdot,s)}(z_t), z_{\boldsymbol{\theta}}(\cdot,s) \right\rangle &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[\varsigma_{\pi_t^{\tau}(a,s)} z_{\boldsymbol{\theta}}(a,s) \right] \\ &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[\varsigma_{\pi_t^{\tau}(a,s)} \left(z_{\boldsymbol{\theta}}(a,s) - \sum_{j=1}^{|\mathcal{A}|} \exp(z_{\boldsymbol{\theta}}(j,s)) \right) \right] \\ &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[\varsigma_{\pi_t^{\tau}(a,s)} \ln \pi_{\boldsymbol{\theta}}(a|s) \right] \end{aligned}$$

where the second line follows from the fact that $\mathbb{E}_{a \sim \pi_t(\cdot|s)}[\varsigma_{\pi_t^{\tau}(a,s)}] = 0$ and so we can shift the values of $z_{\boldsymbol{\theta}}(\cdot, s)$ without changing the expectation.

Finally, with a logsumexp mirror map, Lemma 2 shows that $D_{\phi}(\boldsymbol{z}^{\pi}, \boldsymbol{z}^{\pi_t}) = \mathrm{KL}(\pi \mid\mid \pi)$, and so we get our surrogate objective

$$\begin{aligned} f_{q,s}(\boldsymbol{\theta}) &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[\varsigma_{\pi_t^{\tau}(a,s)} \ln \pi_{\boldsymbol{\theta}}(a|s) \right] + \tfrac{1}{\lambda} \mathrm{KL}(\pi_t(\cdot|s) \mid\mid \pi_{\boldsymbol{\theta}}(\cdot|s)) \\ &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[\varsigma_{\pi_t^{\tau}(a,s)} \ln \pi_{\boldsymbol{\theta}}(a|s) + \tfrac{1}{\lambda} \ln(\pi_t(a \mid s)) - \tfrac{1}{\lambda} \ln(\pi_{\boldsymbol{\theta}}(a \mid s)) \right] \\ &= \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[(\varsigma_{\pi_t^{\tau}(a,s)} - \tfrac{1}{\lambda}) \ln \pi_{\boldsymbol{\theta}}(a|s) \right] \end{aligned}$$

The gradient is simple to compute, because we are sampling from π_t,

$$\nabla f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a \sim \pi_t(\cdot|s)} \left[(\varsigma_{\pi_t^{\tau}(a,s)} - \tfrac{1}{\lambda}) \nabla \ln \pi_{\boldsymbol{\theta}}(a|s) \right]$$

□

8.4 Functional Mirror Descent for Maximum A-Posteriori Policy Optimization

In this section, we provide additional proofs for Mirror Descent Maximum A-Posteriori Policy Optimization not included in the main text.

MPO minimizes a mode-seeking KL between $\pi_{\boldsymbol{\theta}}$ and π^{KL}:

$$\ell_{q,s}(\boldsymbol{\theta}) = \mathrm{KL}(\pi^{\mathrm{KL}} \mid\mid \pi_{\boldsymbol{\theta}}(\cdot|s)) = -\mathbb{E}_{a \sim \pi(\cdot|s)} \left[\exp\left(\frac{q(s,a)}{\kappa} \right) \ln \pi_{\boldsymbol{\theta}}(a \mid s) \right] + \text{ constant}$$

$$\text{where } \pi^{\mathrm{KL}}(a \mid s) \propto \pi(a \mid s) \exp\left(\frac{q(s,a)}{\kappa} \right)$$

for some π, typically the previously learned policy $\pi_{\boldsymbol{\theta}_t}$. The constant is typically dropped. We can incorporate entropy regularization by subtracting $\tau\, \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\ln \pi_{\boldsymbol{\theta}}(a \mid s)]$ to the objective above, for entropy scale τ.

Proposition 6. *On time step t with current action-values q and policy π_t, the surrogate objective for Maximum A-Posteriori Policy Optimization with a direct functional representation and any Bregman divergence D_ϕ, is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[-\exp\left(\frac{q(s,a)}{\kappa}\right) + \tau \ln \pi_t(a|s)\right] + \tfrac{1}{\lambda} D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \tag{45}$$

gradient

$$\nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\left(-\exp\left(\frac{q(s,a)}{\kappa}\right) + \tau \ln \pi_t(a|s)\right) \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a|s)\right] + \tfrac{1}{\lambda} D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \tag{46}$$

Proof. The proof is a straightforward extension of the proof for Proposition 4, replacing the percentile policy in π_t, π^ρ, with the KL policy in π_t, π^{KL}. Following the exact same steps, we get

$$\mathbb{E}_{a\sim\pi^{\text{KL}}(\cdot|s)}\left[-\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] + \tau \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}(\cdot|s)}\left[\ln \pi_t\right] + \tfrac{1}{\lambda} D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) \tag{47}$$

Evaluating the expectation in the first term, we get

$$\mathbb{E}_{a\sim\pi^{\text{KL}}(\cdot|s)}\left[-\frac{\pi_{\boldsymbol{\theta}}(a \mid s)}{\pi_t(a \mid s)}\right] = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[-\exp\left(\frac{q(s,a)}{\kappa}\right)\right]$$

Substituting this back into Equation (47) and combining terms results exactly in Equation (45).

The gradient is once again a simple calculation which involves using the log-likelihood trick to derive the gradient of the expectation in Equation (45). □

Corollary 3. *On time step t with current action-values q and policy π_t, the surrogate objective for Maximum A-Posteriori Policy Optimization with a mode-seeking KL, namely $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = KL(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$, is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[-\exp\left(\frac{q(s,a)}{\kappa}\right) + \left(\tau - \tfrac{1}{\lambda}\right)\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{48}$$

with gradient

$$\nabla_{\boldsymbol{\theta}} f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_{\boldsymbol{\theta}}}\left[\left(-\exp\left(\frac{q(s,a)}{\kappa}\right) + (\tau - \tfrac{1}{\lambda})\ln(\pi_t(a \mid s)) + \tfrac{1}{\lambda}\ln \pi_{\boldsymbol{\theta}}(a \mid s)\right) \nabla_{\boldsymbol{\theta}} \ln \pi_{\boldsymbol{\theta}}(a \mid s)\right] \tag{49}$$

Proof. We simply need to plug-in $D_\phi(\pi_{\boldsymbol{\theta}}(\cdot|s), \pi_t(\cdot|s)) = \text{KL}(\pi_{\boldsymbol{\theta}}(\cdot|s) \mid\mid \pi_t(\cdot|s))$ into Equation (45), re-arrange, and combine terms to get Equation (48).

Using the log-likelihood trick with Lemma 1, yields the gradient. □

Now let us consider the case where we have a softmax parameterization, $\pi_{\boldsymbol{\theta}}(a \mid s) \propto \exp(z_{\boldsymbol{\theta}}(a,s))$. Our network outputs $z_{\boldsymbol{\theta}}(a,s)$, and we consider the functional space over z instead.

Proposition 7. *Assume we have a finite number of actions, $\mid \mathcal{A} \mid$, and use a softmax policy parameterization, $\pi_{\boldsymbol{\theta}}(a|s) = \frac{\exp(z_{\boldsymbol{\theta}}(a,s))}{\sum_{j=1}^{|\mathcal{A}|}\exp(z_{\boldsymbol{\theta}}(a,s))}$. On time step t with current action-values q and policy π_t and logits z_t, the surrogate objective for Maximum A-Posteriori Policy Optimisastion with the log-sum-exp mirror map is*

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t^\tau(a,s)} - \tfrac{1}{\lambda})\ln \pi_{\boldsymbol{\theta}}(a|s)\right] \tag{50}$$

with gradient

$$\nabla f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t^\tau(a,s)} - \tfrac{1}{\lambda})\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)\right] \tag{51}$$

where

$$\varsigma_{\pi^\tau(a,s)} = -\exp\left(\frac{q(s,a)}{\kappa}\right) + 1 + \tau \ln \pi(a \mid s) + \tau\mathcal{H}(\pi(\cdot \mid s))$$

Proof. Similarly to Proposition 4, this proof is a straightforward extension of the corresponding proof for GreedyAC in Proposition 5. We simply replace the percentile policy in π_t, π^ρ, with the KL policy in π_t, π^{KL}. Following the exact same steps, we get

$$f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{\pi_t}\left[(\varsigma_{\pi_t^\tau} - \tfrac{1}{\lambda}) \ln \pi_{\boldsymbol{\theta}}(a \mid s)\right]$$

where $\varsigma_{\pi_t^\tau(a,s)} = \frac{-\pi^{\text{KL}}(a|s)}{\pi_t(a|s)} + 1 + \tau \ln \pi_t(a \mid s) + \tau\mathcal{H}(\pi_t(\cdot \mid s))$. But notice that

$$\frac{-\pi^{\text{KL}}(a \mid s)}{\pi_t(a \mid s)} = -\exp\left(\frac{q(s,a)}{\kappa}\right) \tag{52}$$

Substituting this into the equation for $\varsigma_{\pi_t^\tau(a,s)}$ results in the surrogate above.

The gradient is a straightforward computation, since sampling is performed according to π_t:

$$\nabla f_{q,s}(\boldsymbol{\theta}) = \mathbb{E}_{a\sim\pi_t(\cdot|s)}\left[(\varsigma_{\pi_t^\tau(a,s)} - \tfrac{1}{\lambda})\nabla \ln \pi_{\boldsymbol{\theta}}(a|s)\right]$$

□

9 Closed Form Mirror Descent Updates on the Simplex

In this section, we discuss closed-form mirror descent updates on the probability simplex. Such a case is possible when policies are tabular, a setting we will further empirically consider in Appendix 10.4.

The general framework for utilizing closed-form MD updates with stepsize $\lambda > 0$ is outlined in Algorithm 1. In blue, we show the update equations for the negative entropy mirror map and simplex policies.

Algorithm 1: Closed-Form Mirror Descent for Policy Improvement

input : mirror map $\phi : \mathcal{D} \subseteq \mathbb{R}^n \to \mathbb{R}$, policy π_i, $\lambda > 0$, $\boldsymbol{s}_t \in \mathcal{S}$

$\boldsymbol{x}_i \leftarrow \pi_i(\cdot|\boldsymbol{s}_t)$

$\hat{\boldsymbol{x}}_i \leftarrow \nabla\phi(\boldsymbol{x}_i) = \ln(\boldsymbol{x}_i) + 1$

$\hat{\boldsymbol{y}}_{i+1} \leftarrow \hat{\boldsymbol{x}}_i - \lambda \boldsymbol{g}_i$ `// For gradient` $\boldsymbol{g}_i$

$\boldsymbol{y}_{i+1} \leftarrow \nabla\phi^* = e^{\hat{\boldsymbol{y}}_{i+1}-1}$

$\boldsymbol{x}_{i+1} \leftarrow \Pi_\phi^{\mathbb{S}^{|\mathcal{A}|}}(\boldsymbol{y}_{i+1}) =$
$\arg\min_{\boldsymbol{x}\in\mathbb{S}^{|\mathcal{A}|}\cap\mathcal{D}}(\boldsymbol{x}, \boldsymbol{y}_{i+1}) = \frac{\boldsymbol{y}_{i+1}}{\|\boldsymbol{y}_{i+1}\|_1}$

$\pi_{i+1}(\cdot \mid \boldsymbol{s}_t) \leftarrow \boldsymbol{x}_{i+1}$

We will use the following notations. Let $\boldsymbol{M}(i,j) = \boldsymbol{M}^{(j,i)}$ be the element of $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ at row j and column i. Define $\boldsymbol{M}(i,\cdot) = \boldsymbol{M}^{(\cdot,i)} \in \mathbb{R}^n$ as the vector composed of the i-th column of $\boldsymbol{M}$ and $\boldsymbol{M}(\cdot,j) = \boldsymbol{M}^{(j,\cdot)^\top}$ as the column vector composed of the j-th row of $\boldsymbol{M}$. The indicator matrix for (s,a), denoted as $\mathbb{1}(s,a) \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|}$, is the matrix of zeros everywhere and a 1 at $\mathbb{1}^{(a,s)}$. The indicator matrix for column s, denoted as $\mathbb{1}(s,\cdot) \in \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|}$, is the matrix of zeros everywhere and a column of $\mathbf{1}$ at $\mathbb{1}^{(\cdot,s)}$.

Let $\mathbb{M} \subset \mathbb{R}^{|\mathcal{A}|\times|\mathcal{S}|}$ denote the space of probability matrices. Similarly to the previous sections, let $\mathbb{S}^n$ be the simplex in $\mathbb{R}^n$, $\mathcal{S}$ be the state space and $\mathcal{A}$ the action space. For $\boldsymbol{S} \in \mathbb{M}$, each column has the simplex restriction, namely that $\boldsymbol{S}^{(j,i)} \in [0,1]$ and $\mathbf{1}^\top\boldsymbol{S}(i,\cdot) = 1$ for $i \leq \mid \mathcal{A} \mid \in \mathbb{Z}$, $j \leq \mid \mathcal{S} \mid \in \mathbb{Z}$. For policy $\pi_{\boldsymbol{\theta}}$ parameterized by $\boldsymbol{\theta} \in \mathbb{M}$, $\pi_{\boldsymbol{\theta}}(a \mid s) = \boldsymbol{\theta}^{(a,s)}$ denotes the probability of selecting action a in state s.

In Section 10, we analyze the closed-form MD-GreedyAC with RKL algorithm in more depth. For completeness, we here provide the algorithms for the variants of GreedyAC, SAC, and MPO which use closed-form MD updates. We omit gradient derivations as each is a simple derivative calculation for the corresponding algorithm's loss function with tabular function approximation. We introduce all algorithms with entropy regularization for entropy scale hyperparameter τ.

Mirror Descent Greedy Actor-Critic Let q be an action-value function estimate. Closed-Form Mirror Descent Greedy Actor-Critic with an RKL penalty (CMD-GreedyAC with RKL) on the simplex is exactly Algorithm 1 with the following gradient in state $s \in \mathcal{S}$:

$$\boldsymbol{g} = \tau \ln(\boldsymbol{\theta}(s, \cdot)) - \mathbb{1}(s, a^*) \frac{1}{\boldsymbol{\theta}(s, a^*)} \qquad \text{for } a^* = \arg\max_{a \in \mathcal{A}} q(s, a) \tag{53}$$

Mirror Descent Soft Actor-Critic Let $b : \mathcal{S} \to \mathbb{R}^n$ arbitrary and define $\mathbb{A}_t(s, a) = q(s, a) - b(s)$. $\mathbb{A}$ is the advantage estimate of action a in state s when b is an approximation to the policy state-value function. Closed-Form Mirror Descent Soft Actor-Critic with an RKL penalty (CMD-SAC with RKL) on the simplex is exactly Algorithm 1 with the gradient:

$$\boldsymbol{g} = \tau \ln(\boldsymbol{\theta}) - \mathbb{A}(s) \qquad \text{where } \mathbb{A}(s) = \begin{bmatrix} \mathbb{A}(s, a_1) \\ \mathbb{A}(s, a_2) \\ \vdots \\ \mathbb{A}(s, a_{|\mathcal{A}|}) \end{bmatrix} \tag{54}$$

in state $s \in \mathcal{S}$

Mirror Descent Maximum A-Posteriori Policy Optimisation Let $\pi_{\text{KL}}(a \mid s)$ be the KL policy in π as in Section 8. Closed-Form Mirror Descent Maximum A-Posteriori Policy Optimisation with an RKL penalty (CMD-MPO with RKL) on the simplex is exactly Algorithm 1 with the gradient in state $s \in \mathcal{S}$:

$$\boldsymbol{g} = \tau \ln(\boldsymbol{\theta}(s, \cdot)) - \mathbb{1}(s, \cdot) \frac{\pi_{\text{KL}}(s, \cdot)}{\boldsymbol{\theta}(s, \cdot)} \tag{55}$$

unlike the version of MPO introduced by Abdolmaleki et al. (2018b), MD-MPO with the negative entropy mirror map is equivalent to using a KL penalty rather than a constraint.

10 Further Experimental Results

In this section, we provide further empirical analysis of the MD-style algorithms considered.

10.1 Step Size Cliff

In this section, we provide additional details and analysis regarding our experiments in Section 4.2.1.

In our experiments in Section 4.2.1, a few of the extreme step sizes tested resulted in convergence for SAC, according to our convergence metric mentioned in Section 4.2.1. To determine these outliers, we used a Local Outlier Factor algorithm from ScikitLearn (Pedregosa et al., 2011) using 30 neighbours and the $L2$-norm. We conducted the outlier detection based on the features $(\log(\alpha_{\text{actor}}), y)$ where α_{actor} was the randomly sampled step size, $y = 1$ if the sampled α_{actor} resulted in convergence according to our metric, and $y = -1$ otherwise.

Figure 6 shows the results of similar experiments to those described in Section 4.2.1, except for $\log(\lambda) \in \{-2, -1, 1, 2\}$. Again, we set $\log(\tau) = \log(\alpha_{\text{critic}}) = -3$ and provide plots for at least 1,500 random seeds. We see a clear trend for MD-SAC with FKL here. As $\log(\lambda)$ increases, fewer actor step sizes work – the algorithm becomes more susceptible to the step size cliff. Again, we mention that many of the unsuccessful step sizes in this figure still produced reasonable performance but did not meet our criterion for convergence. Results for $\log(\tau) = -4$ are quantitatively similar, but fewer points met our convergence criterion.

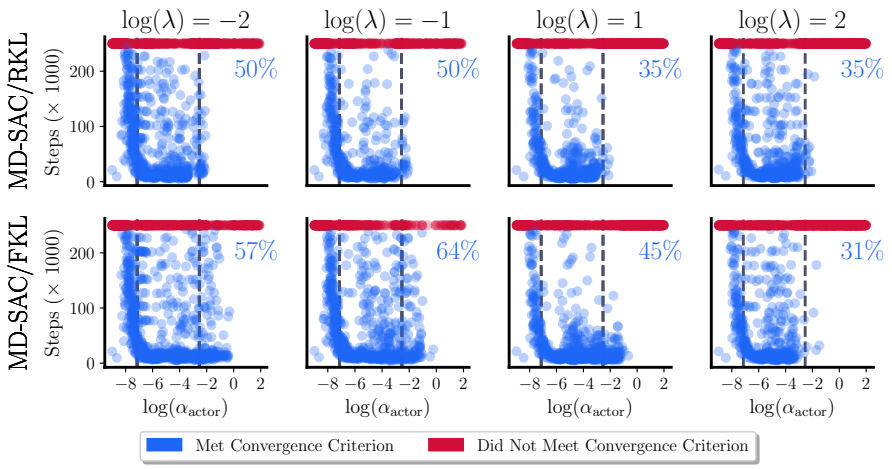


Figure 6: Steps for MD-SAC to learn a near-optimal policy for different $\log(\lambda)$ with RKL (top) and FKL (bottom) penalties. Each point represents a single agent with a randomly sampled $\log(\alpha_{\text{actor}}) \in [-9, 2]$. Dashed lines indicate the smallest/largest working step sizes for SAC using outlier detection as in Figure 3. Inset blue text indicates the percentage of blue points.

10.2 Replay Ratio

In this section, we present additional results for our replay ratio analysis in the discrete-action. All experiments follow the same setup as in Section 4.2.3.

Figure 7 shows the sensitivity of each algorithm to the actor replay ratio in the discrete-action setting, across classic control problems. Similarly to the continuous-action results presented in Section 4.2.3, we found no consistent relationship between replay ratio and performance. Again, MD-style algorithms did not consistently benefit from increased number of actor updates. Instead, often increasing the number of actor updates per environment step was detrimental to performance – as in the case of MPO.

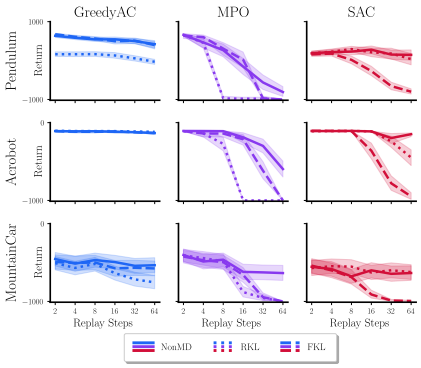


Figure 7: Performance vs replay ratio over 60 runs with discrete-actions.

10.3 Return Landscape for Mirror Descent Hyperparameters

The hyperparameters λ and M determine both the step size of the MD update and the degree to which the MD objective is approximated, where an increasing M indicates a better approximate solution to the MD objective, with appropriately chosen actor step size. We were interested in determining if and how these hyperparameters affect the performance of algorithms.

We therefore analyzed the return landscape as a function of (λ, M) across two domains, the tabular Cliffworld environment (Sutton & Barto, 2018) and discrete-action Acrobot. We chose Acrobot because each algorithm could learn well on it, but it is not as easy as Pendulum and not as hard as MountainCar. On Cliffworld, we used tabular function approximation, while for Acrobot we used neural networks. We used the MD algorithms developed in the main text: on Cliffworld, we did not use the tabular, closed-form variants of these algorithms which were developed in Appendix 9. To analyze the return landscape, we randomly sampled $\log(\lambda) \in [-5, 5]$ and $M \in [3, 500]$ ($M \in [3, 250]$ for Acrobot). We performed all M SGD updates every environment step. We then plotted scatter plots of (λ, M) with colour denoting average episodic return.

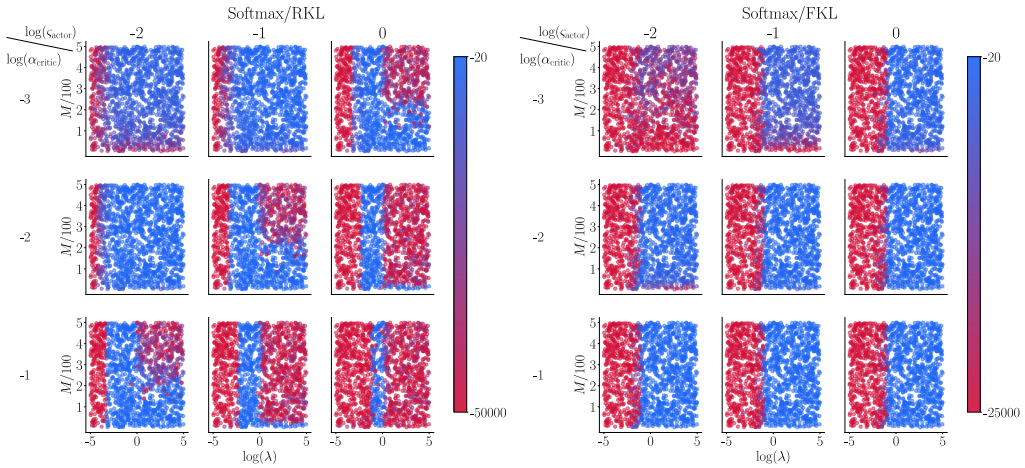


Figure 9: Scatter plot of randomly sampled $(\log(\lambda), M)$ on Cliffworld for GreedyAC with softmax policies over 1,500 random seeds. Colour denotes average episodic return on a linear scale. Each row/column corresponds to a different critic/actor stepsize.

We used GreedyAC which we found to be the least sensitive algorithm to hyperparameters. The entropy scale was fixed to 0. For Acrobot, all hyperparameters and experimental details followed those outlined in Setion 4. For the tabular Cliffworld environment, We swept critic step sizes $\alpha_{\text{critic}} \in \{-1, -2, -3\}$ and actor step sizes $\alpha_{\text{actor}} = 10^{\varsigma_{\text{actor}}} \times \alpha_{\text{critic}}$ with $\varsigma_{\text{actor}} \in \{-2, -1, 0\}$. On Cliffworld, we used both tabular softmax policies (MD-GreedyAC with FKL and with RKL) and tabular simplex policies (MD-GreedyAC with RKL). Experiments were run for 25,000 steps.

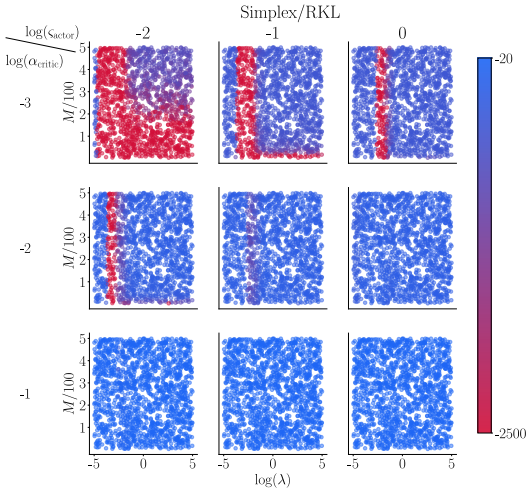


Figure 8: Randomly sampled $(\log(\lambda), M)$ on Cliffworld for GreedyAC (simplex policies) over 1,500 random seeds. Colour denotes episodic return on a linear scale. Rows/columns denote critic/actor stepsizes.

We will first consider the tabular experiment on Cliffworld. Figure 8 shows a scatter plot of 1,500 random $(\log(\lambda), M)$ pairs for simplex policies with an RKL penalty on Cliffworld. Colour denotes episodic return on a linear scale. The figure indicates that for simplex policies, performance was primarily influenced by actor/critic stepsizes rather than λ and M. Furthermore, we found that minuscule ($\alpha_{\text{actor}} = 10^{-5}$) actor step sizes generally required larger M, as expected, but did not often result in higher return.

Figure 9 shows scatter plots of 1,500 randomly sampled $(\log(\lambda), M)$ pairs for RKL (left) and FKL (right) penalties and softmax policies on Cliffworld. Compared to simplex policies, softmax policies induced higher sensitivity to $(\log(\lambda), M)$, indicating that policy parameterization affects sensitivity. We explicitly point out the difference in scales in the colour bars in Figures 8 and 9, which makes this relationship quite clear. On the other hand, we observed that with minuscule actor step sizes ($\alpha_{\text{actor}} = 10^{-5}$), the softmax policy parameterization tended to work better than the simplex policy parameterization. Again we note that larger values of M tended to result in higher return in this case. An FKL penalty induced lower sensitivity to $(\log(\lambda), M)$ compared to the RKL penalty in the softmax case. Softmax policy parameterizations seemed to be more sensitive to the actor and critic stepsizes than to either λ or M.

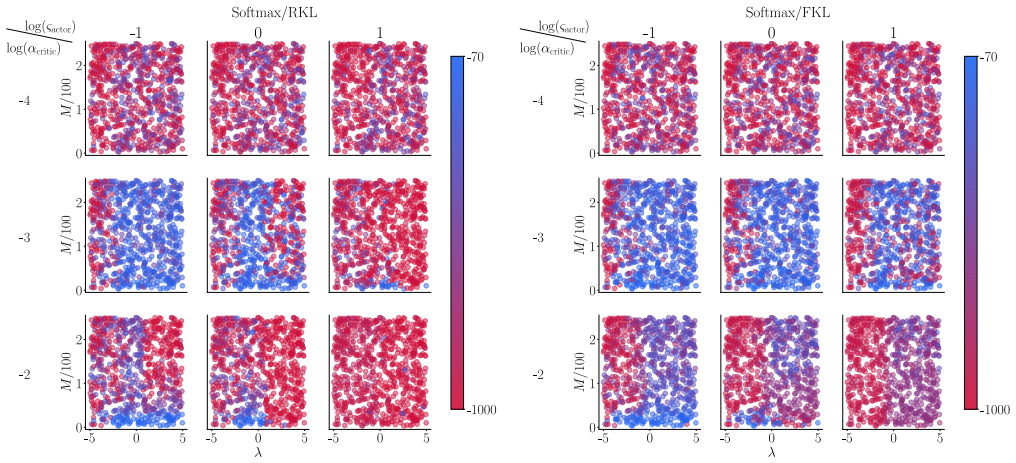


Figure 10: Scatter plot of randomly sampled $(\log(\lambda), M)$ on Acrobot for GreedyAC with softmax policies and neural networks over 750 random seeds. Colours denote average episodic return. Each row/column corresponds to a different critic/actor stepsize.

Overall, we found low sensitivity to λ and M in the tabular case on Cliffworld. It is worth noting that for a well-tuned actor and critic step size, a wide range of values for λ and M induced similar performance. This could indicate that the MD objective is quite easy to solve on this problem or perhaps that the MD objective is difficult to solve, but the environment too easy to show a big difference in performance across multiple values of M.

Figure 10 shows the corresponding scatter plot of 750 randomly sampled $(\log(\lambda), M)$ pairs on discrete-action Acrobot with neural networks. Compared to the tabular case, the patterns of performance sensitivity with neural networks were much less obvious. In both RKL and FKL cases, we observed higher sensitivity to λ and M. With an FKL penalty, we found a similar pattern to the tabular case: larger values of λ tended to induce higher performance. Furthermore, we noticed a slight trend with regards to α_{actor} and M. For large values of M, performance tended to increase with decreasing α_{actor}, perhaps indicating the benefits of a better approximation to the MD optimization in Equation 2.

10.4 Noisy Rewards

Previous work has suggested that mirror descent can enhance robustness to reward noise (Vieillard et al., 2020a; Lazić et al., 2021). We analyzed GreedyAC's performance in a noisy reward environment, focusing on the tabular setting where MD is analytically tractable. We designed a small MDP where MD updates dramatically improve over conventional ones. Figure 11 (right) shows a tabular tree MDP where the agent starts at the root and moves to leaf nodes via left or right actions. Rewards are -10, except for one transition (blue edge) with a reward of -1 (probability $\frac{274}{275}$) or -100 (probability $\frac{1}{275}$). The optimal policy ($\gamma = 1$) is to always go right. We ran GreedyAC for 25,000 steps, including both the closed-form MD updates developed in Appendix 9 and the approximate MD updates considered in the main text. In this section, we will refer to MD-GreedyAC with RKL as the approximate MD update and denote the number of gradient steps made on the surrogate objective, M, in parentheses. We refer to CMD-GreedyAC with RKL as the closed-from MD variant with an RKL penalty.

We swept the following hyperparameters and report performance across 50 runs. Critic step sizes were swept in 2^x for $x \in \{-1, -2, -3\}$. Actor step sizes were swept in 2^x for $x \in \{-11, -10, \ldots, -1\}$ for CMD-GreedyAC and standard GreedyAC. For MD-GreedyAC with RKL we instead swept $x \in \{-11, -9, -7, -5\}$. For MD-GreedyAC, we also swept $\lambda = 2^x$ for $x \in \{-15, -14, -13, \ldots, -1\}$. We tested $M \in \{250, 100, 10\}$. Simplex policies were initial-

ized uniformly and used a negative entropy mirror map. We did not use entropy regularization. We used simplex policies, initialized uniformly, with an RKL penalty for the MD update.

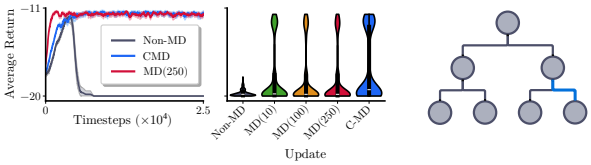


Figure 11: (**left**) Learning curves on the noisy reward environment. (**middle**) Performance distributions. (**right**) Environment.

GreedyAC is unstable. It initially learned the optimal policy but quickly switched to a suboptimal one. In contrast, MD-GreedyAC with RKL consistently learned the optimal policy for a range of values of M – faster than other update type, shown in Figure 11 (Left). The closed-form MD update was less sensitive to hyperparameters (Middle) but did not learn as quickly as the approximate MD update. This finding suggests that the performance improvements of the approximate MD updates noted here may not completely stem from accurately approximating a closed-form update. Despite significant effort we could not extend this finding to large environments with function approximation: adding noise and partial observability to classic control domains did not yield significant advantages for MD updates.

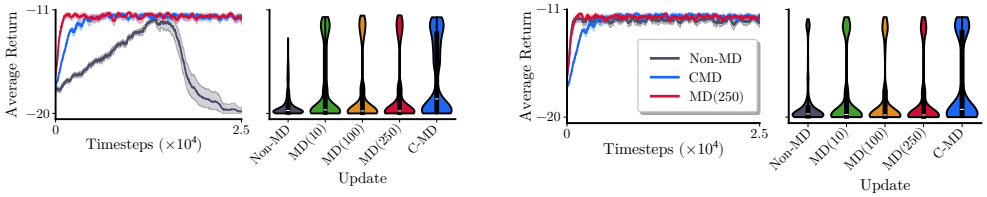


Figure 12: Learning curves and performance distributions when (**left**) tuning over lower actor step sizes and (**right**) adding entropy regularization. Solid lines denote mean performance over 50 runs with shaded regions denoting 95% percentile bootstrap confidence intervals.

One could ask how GreedyAC could be improved to match the performance of either MD-GreedyAC CMD-GreedyAC. Further reducing the actor step size inhibits learning and slows – but does not prevent – convergence to the suboptimal policy. Figure 12 (left) illustrates this phenomenon for smaller actor step sizes tuned over the additional values of $\alpha_{\text{actor}} = 2^x$ for $x \in \{-13, -12\}$. The figure shows that while a low actor step size slow convergence to the suboptimal policy, it did not prevent it. Further reducing the actor step size to 2^{-15} enabled GreedyAC to learn the optimal policy across all runs by the experiment's end, but convergence remained slow, requiring the full 25,000 steps. We conjecture that, if the experiment's length were extended, GreedyAC would have eventually collapsed to the suboptimal policy even with this miniscule actor step size of 2^{-15}. The reason for this collapse is an exploding gradient, and a small, fixed actor step size cannot alone prevent this.

Entropy regularization enabled GreedyAC to learn the optimal policy. Figure 12 (Right) shows the learning curves of GreedyAC when tuning the entropy scale $\tau = 2^x$ for $x \in \{-10, -8, \ldots, -2\}$. In this case, GreedyAC benefited from a significant improvement in learning and no longer converged to the suboptimal policy. Even so, the MD-style updates exhibited improved hyperparameter sensitivity compared to the GreedyAC update. This result perhaps indicates that MD style optimization induces some form of entropy regularization, as also suggested by results in the literature (Lazić et al., 2021; Azizan et al., 2022).