# MDPO-Auto: Enabling Hyperparameter Auto-tuning in Off-policy Mirror Descent Policy Optimization

Qianxi Li
University of Alberta
qianxi@ualberta.ca

**Abstract**

Mirror Descent Policy Optimization (MDPO), is a deep reinforcement learning (RL) algorithm that uses the idea of Mirror Descent (MD) to enforce a trust-region constraint in a soft way when optimizing for the policy. Researchers examine the performance of MDPO and notice it outperforms several popular deep RL algorithms, such as TRPO, PPO and SAC. In the second paper of SAC, the authors extend the vanilla SAC algorithm, they design and implement an approach to allow one critical hyperparameter - entropy coefficient to be tuned automatically during the deep RL training. The entropy coefficient set the relative importance of the entropy term when optimizing the policy and it is usually selected manually and empirically by the researchers, which is extremely time-consuming.

In this work, we will combine the work of the off-policy MDPO algorithm and the entropy coefficient auto-tuning approach to obtain a new variant of the off-policy MDPO algorithm, which we name it MDPO-Auto. We derive the update rule for the entropy coefficient and modify the off-policy MDPO algorithm to be the MDPO-Auto algorithm. Besides, we conduct experiments on an OpenAI Gym simulated environment and MDPO-Auto outperforms off-policy MDPO. The GitHub link to this work is at https://github.com/liqianxi/auto_tune_MDPO

## 1 Background

Tomar et al. [2020] proposes a new reinforcement learning (RL) algorithm called mirror descent policy optimization (MDPO) inspired by the mirror descent (MD) approach in the area of constrained convex optimization. The main idea behind this algorithm is that differs from the typical optimization objectives in RL, which attempt to maximize the expected sum of all the returns and obtain the optimal policies, instead, they add a second term after the gradient term to penalize the two adjacent policies for the large distance between them, while the distance can be defined as different variants of the Bregman divergence, such as KL divergence.

In each iteration of the MDPO algorithm, the policy update is formulated as a trust-region problem. However, to update the policy, in lieu of solving the following optimization problems, they perform multiple SGD steps to approximate the target policy.

For the experiment part, they compare the on-policy MDPO with two popular RL algorithms, TRPO and PPO, and for the off-policy MDPO, they set up MDPO with different distance measuring metrics, such as Tsallis and KL. They compare these variants with the SAC algorithm. They also investigate how the value of SGD steps affects the performance of MDPO. As a result, for the on-policy case, MDPO performs better than or equal to TRPO and strictly better than PPO. For the off-policy case, one of the MDPO variants - MDPO-KL performs on par with SAC on all the tasks, and another variant MDPO-Tsallis outperforms SAC.

Haarnoja et al. [2018b] extends the previous work on Soft Actor-Critic (SAC) algorithm Haarnoja et al. [2018a] and suggests a new approach to auto-tuning the hyper-parameter - entropy temperature ($\alpha$) such that the expected entropy term can be adjusted over the visited states, which means the agent will be encouraged to have different levels of exploration throughout the training process. The paper also evaluates the performance of different variants of SAC on several real-world tasks in robotics.

The idea behind auto-tuning $\alpha$ is a primal-dual problem. In maximum entropy reinforcement learning, the original total reward maximization optimization problem is constrained by forcing the total entropy to exceed a threshold (as shown in Eq 1), encouraging the agent to learn an optimal policy with diversities.

$$\max_{\pi_{0:T}} E_{\rho_\pi}\left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] s.t. E_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[-\log\left(\pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\right] \geq \mathcal{H}\ \forall t \quad \text{(Eq 1)}$$

They convert (Eq 1) into its dual problem, with $\alpha$ as the new variable, instead of the policy ($\pi$). And they derive the formula to optimize for $\alpha$ and the algorithm uses SGD to approximate the target at every gradient step, thus achieving the goal of automatically tuning $\alpha$.

For the experiment, they compare the updated SAC algorithm with novel off-policy and on-policy algorithms on several continuous control tasks. The result shows that the SAC with auto-tuning outperforms the state-of-the-art in terms of sample efficiency and performance. And they also offer the automatic temperature tuning method works well across different environments, indicating that the method is stable.

# 2 Introduction

The problem we are trying to solve is to allow auto-tuning of the entropy coefficient ($\lambda$) in the entropy-regularized version of the **off-policy, soft version** of the mirror descent policy optimization method. In the context of deep reinforcement learning, a model that is entropy-regularized means the policies we learned consecutively are encouraged to be different from each other and will do more exploration, thus improving the diversities of policies of the agents. The multiplier $\lambda$ is to control the strength of the entropy. In the original work of Mirror Descent Policy Optimization (MDPO) Tomar et al. [2020], the authors introduce the off-policy version of the MDPO algorithm and state the fact that the vanilla algorithm can be converted into entropy-regularized MDPO. For the experiments, they use entropy-regularized MDPO with the hyperparameter $\lambda = 0.2$. However, $\lambda$ can be set automatically in the program for different tasks without human knowledge to set the value manually. In the Soft Actor-Critic (SAC) paper Haarnoja et al. [2018b], in order to enable auto-tuning of the entropy temperature hyperparameter $\alpha$ for the Soft Actor-Critic (SAC) algorithm, the authors formulate the problem as a constraint optimization problem, with the entropy term becoming the inequality condition, as shown below.

$$\max_{\pi_{0:T}} E_{\rho_\pi}\left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \text{ s.t. } E_{(\mathbf{s}_t, \mathbf{a}_t)\sim\rho_\pi}\left[-\log\left(\pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\right] \geq \mathcal{H}\ \forall t$$

They then convert the problem to its dual problem, which now involves the minimization of a function with its dual variable $\alpha$. With the help of the neural network, they can then approximate the optimal value of $\alpha$ for different tasks.

Our approach is to find such a dual optimization problem for entropy-regularized off-policy MDPO, we will get an optimization function with a dual variable of $\lambda$, and to find the optimal value of it, we use stochastic gradient descent to update and approximate the value of $\lambda$ in every training step. Meanwhile, the original MDPO work only provides an off-policy algorithm without entropy regularization. Our plan is to provide a complete MDPO algorithm incorporating entropy regularization and $\lambda$ auto-tuning so that people can implement the algorithm easier. Our implementation of the full algorithm will also be provided.

Since $\lambda$ is set to auto-tuning, we expect to see a performance improvement in the new MDPO model. Experiments will be conducted using several different reinforcement learning simulated environments and among different variants of the off-policy MDPO and SAC models.

# 3 Automating Entropy Adjustment for MDPO

## 3.1 Notations

Assume the agent's interaction with the environment is modelled as a $\gamma$-discounted Markov decision process (MDP), denoted by $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mu)$, where S and A are the state and action spaces; P is the transition kernel; R is the reward function; $\gamma \in (0, 1)$ is the discount factor, $\mu$ is the initial state distribution. $\pi : \mathcal{S} \to \mathcal{A}$, it maps from the state space to the action space. $V^\pi$ and $Q^\pi$ are value function and action-value functions. The difference between them is the advantage function $A$, $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

## 3.2 Entropy Regularized MDP

Many researchers propose adding a regularizer to the reward of the MDP to encourage exploration since finding an optimal policy in an MDP requires solving non-linear equations and the optimal policy may be deterministic. The reward function under such regularized setting can be expressed as $\gamma_\lambda(s, a) = \gamma(s, a) + \lambda H(\pi(\cdot|s))$, where $\lambda$ is the regularization parameter and $H$ is an entropy related term, such as Tsallis entropy (Lee et al. [2017]), Shannon entropy(Fox et al. [2015]) or relative entropy (Azar and Kappen [2010]). In this report, we use 'soft' or 'entropy-regularized' to refer to the regularized MDP interchangeably. To restore back to the 'hard' or 'non-entropy-regularized' version of MDP, we simply set $\lambda$ to 0.

### 3.3 Automating Entropy Adjustment Rule Derivation

Inspired by Haarnoja et al. [2018b], in order to allow the entropy coefficient $\lambda$ to adjust automatically, it first formulates a constrained optimization problem, then covert the problem using a dynamic programming approach, then solves the dual-problem of each sub-problem, simplifies and gets the final optimization problem iteratively. We will do the same thing in this work. In the original work of MDPO, the algorithm provide the 'hard' version of its on-policy and off-policy algorithm to keep the vanilla MDPO algorithm simple and clean. However, they use the 'soft' version of MDPO in the code implementation since it gives better performance. **Our work below will extend the off-policy, soft version of the MDPO algorithm**.

First, we formulate a constrained optimization problem

$$\max_{\pi_{0:T}} E_{\rho_\pi}\left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \text{ s.t. } E_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[-\log\left(\pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\right] \geq \mathcal{H}\ \forall t \tag{1}$$

Where $\mathcal{H}$ is the minimum expected entropy level we want to achieve. Although in this case, we introduce a new hyperparameter $\mathcal{H}$, in practice, it can be decided much easier than $\lambda$, the value is usually set to $-dim(ActionSpace)$ of an environment. The reason why there's no Bregman divergence term in this formula is that the way MDPO enforces trust region only for the policy, so you will only see the Bregman divergence term in the formula of optimizing policy, it is not relevant to the optimization of $\lambda$.

Since the policy at time t will only affect the future objective value, we can employ an (approximate) dynamic programming approach, solving for the policy backward through time.

$$\max_{\pi_0}(E[r(s_0, a_0)] + \max_{\pi_1}(E[...] + \max_{\pi_T}[E[r(s_T, a_T)]]) \tag{2}$$

Here it is formulated in a dynamic programming way, we see that most inside expressions can be converted into its dual problem

$$\max_{\pi_T} E_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r\left(\mathbf{s}_T, \mathbf{a}_T\right)\right] = \min_{\lambda_T\geq 0}\max_{\pi_T} E\left[r\left(\mathbf{s}_T, \alpha_T\right) - \lambda_T \log\pi\left(\mathbf{a}_T \mid \mathbf{s}_T\right)\right] - \lambda_T\mathcal{H} \tag{3}$$

The optimal policy $\pi_T^*$ is corresponding to the temperature $\lambda_T$, to solve for the optimal dual variable $\lambda_T^*$ since $r(s_T, a_T)$ and $-\lambda_T\mathcal{H}$ are constant values, we can remove the first $r(s_T, a_T)$ and move $-\lambda_T\mathcal{H}$ into the expect:

$$\arg\min_{\lambda_T} E_{\mathbf{s}_t,\mathbf{a}_t\sim\pi_t^*}\left[-\lambda_T \log\pi_T^*\left(\mathbf{a}_T \mid \mathbf{s}_T; \alpha_T\right) - \lambda_T\mathcal{H}\right] \tag{4}$$

Here's the recursive definition of the soft Q-function:

$$Q_t^*\left(\mathbf{s}_t, \mathbf{a}_t; \pi_{t+1:T}^*, \alpha_{t+1:T}^*\right) = E\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] + E_{\rho_\pi}\left[Q_{t+1}^*\left(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}\right) - \lambda_{t+1}^* \log\pi_{t+1}^*\left(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}\right)\right] \tag{5}$$

Now, with the help of the recursive definition of the soft Q-function, to solve the dynamic programming problem using dual problem, we start from the inside part of (2):

$$\begin{aligned}&\max_{\pi_{T-1}}\left(E\left[r\left(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}\right)\right] + \max_{\pi_T} E\left[r\left(\mathbf{s}_T, \mathbf{a}_T\right)\right]\right)\\ &= \min_{\lambda_{T-1}\geq 0}\max_{\pi_{T-1}}\left(E\left[Q_{T-1}^*\left(\mathbf{s}_{T-1}, \mathbf{a}_{T-1}\right)\right] - E\left[\lambda_{T-1}\log\pi\left(\mathbf{a}_{T-1} \mid \mathbf{s}_{T-1}\right)\right] - \lambda_{T-1}\mathcal{H}\right) + \lambda_T^*\mathcal{H}\end{aligned} \tag{6}$$

We can then do it backwards in time and optimize (1) recursively. Note that the optimal policy at time t is a function of the dual variable $\lambda_t$. Similarly, we can solve the optimal dual variable $\lambda_t^*$ after solving for $Q_t^*$ and $\pi_t^*$:

$$\lambda_t^* = \arg\min_{\lambda_t} E_{\mathbf{a}_t\sim\pi_t^*}\left[-\lambda_t \log\pi_t^*\left(\mathbf{a}_t \mid \mathbf{s}_t; \alpha_t\right) - \lambda_t\overline{\mathcal{H}}\right] \tag{7}$$

In practice, we need to use function approximators and SGD to approximate the final result of (7), instead of solving it. Here's the final loss function we will use to approximate the optimal entropy coefficient $\lambda$:

$$L(\lambda) = E_{\mathbf{a}_t\sim\pi_t}\left[-\lambda\log\pi_t\left(\mathbf{a}_t \mid \mathbf{s}_t\right) - \lambda\overline{\mathcal{H}}\right] \tag{8}$$

The final algorithm of MDPO-Auto is simply adding the logic for optimizing (8) after we optimize for policy network $\pi$ in each training step.

# 4 Experiments

## 4.1 Set Up for Training

The experiments are conducted among MDPO, SAC and our new MDPO-Auto algorithms. The variant of MDPO we are using is MDPO-Tsallis as it gives the best performance in the original paper, we simply refer to this variant as MDPO throughout this report. In Tomar et al. [2020] the authors illustrate that the off-policy MDPO outperforms popular algorithms TRPO and PPO in several experiments, so their result should be irrelevant to our comparison here. Since we use the auto-tuning idea from SAC, we should also include SAC in the experiments. The experiments are done using a continuous control tasks simulated environment on OpenAI Gym called Walker2d-v2. All experiments are run across 5 random seeds and outliers are removed. The plot will report the empirical mean of the random runs while the shaded region represents a 95% confidence interval (empirical mean$\pm$1.96$\times$empirical standard deviation $/n^{0.5}$).

Here's the hyperparameter list for the experiments: Number of hidden units per layer: 256; minibatch size: 256; entropy coefficient: ($\lambda$); adam step size: 3 x $10^{-4}$; replay buffer size: $10^6$; target value function smoothing coefficient: 0.005; the number of hidden layers: 2; discount factor 0.99; #runs used for plot averages: 5; confidence interval for plot runs 95%; Bregman step size $\frac{1}{t_k}$: 0.4. We use the same settings as in Tomar et al. [2020] to make sure we can reproduce the result of MDPO and SAC as close to the original work as possible. The training process is designed to be run on a device with GPU. We recently obtained access to Compute Canada but couldn't resolve some of the Python library conflicts on it. Thus all the experiments were trained on a MacBook using CPUs only.

The reason for using $\lambda = 0.2$ for MDPO is that Tomar et al. [2020] selects this value as it gives the best-performing value for all tasks. The design of MDPO should have multiple gradient steps at each iteration, but due to the high computational cost for the off-policy case, we only run one gradient step but fixing $\pi_{\theta_k}$ for m number of gradient steps to mimic the effect of multiple gradients steps at each iteration.

## 4.2 Experiment Results

Figure (1) plot shows the result of our experiment, we compare the average returns of MDPO, SAC and our new MDPO-Auto algorithms. The average results of the 5 runs are reported along with the 95% confidence interval. The performance of MDPO and SAC-Auto is matched with the result presented in Tomar et al. [2020]. We observe the following things from the plot. First, MDPO outperforms SAC in Walker's 2D task. Second, our MDPO-Auto performs better or on par with MDPO, and clearly better than SAC. However, the result of these three algorithms exhibits substantial variability in average return, and the confidence interval at each timestamp is very wide.

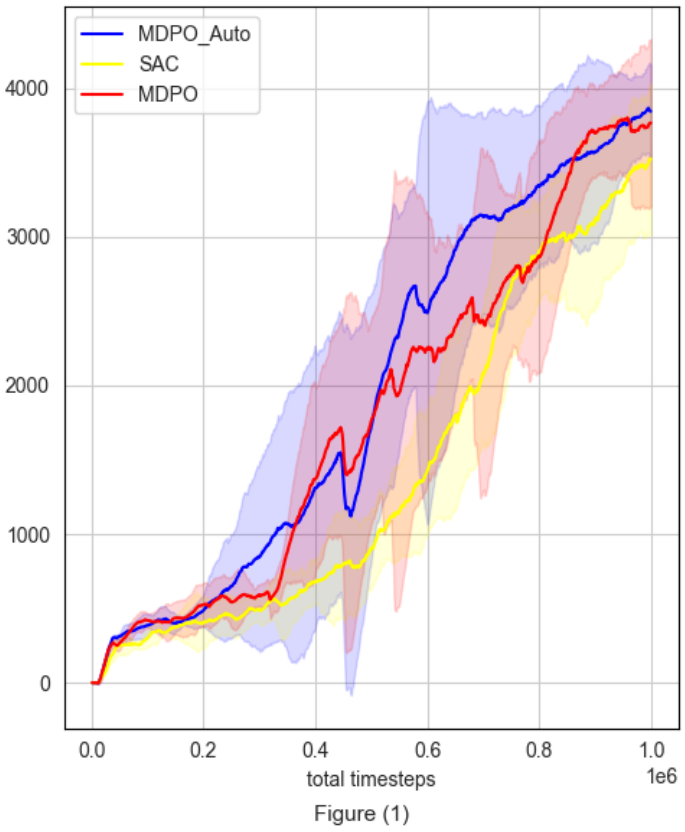

Figure (1)

# 5 Analysis

To interpret the result in figure 1 between the comparison of MDPO and MDPO-Auto, we first look at figure 2, which shows that by auto-tuning, the value of entropy coefficient $\lambda$ eventually converges to 0.2314 given enough time. Since the dual problem is always a convex optimization problem, the initial value of $\lambda$ doesn't change the final result. Recall that the optimal $\lambda$ value from Tomar et al. [2020] is manually set to 0.2 since it gives the best performance among all the $\lambda$ values an earlier experiment tested. We argue that the optimal $\lambda$ value of MDPO for this environment should actually be around 0.22 to 0.23, which is extremely close to their result. Since they were choosing the $\lambda$ value empirically from a set of values, not approximating the value gradually, their $\lambda$ may not lead to the optimal return and it takes much more human effort and computation to decide the value of $\lambda$.

Looking back to figure 1, MDPO-Auto performs better than MDPO and sometimes on par with MDPO, this shows that their manually chosen $\lambda$=0.2 is very close to our approximated optimal value 0.23 which gives MDPO a sub-optimal solution.

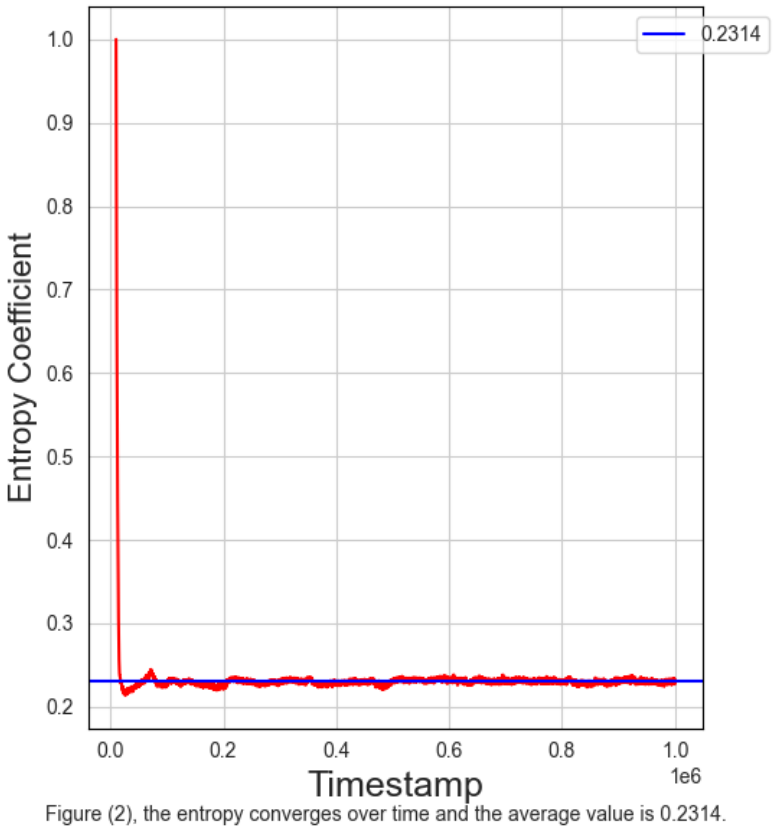

Figure (2), the entropy converges over time and the average value is 0.2314.

# 6 Novelty

The novelty of this work is two-fold:

- The approach of formulating a hyperparameter tuning problem as a constrained optimization comes from SAC and this is the first time it has been applied to an algorithm that outperforms SAC.
- Auto-tuning can save researchers a wealth of time and computational cost of empirically choosing hyperparameters.

# 7 Conclusion

In this report, we present our work MDPO-Auto which is the **off-policy, soft version** of the mirror descent policy optimization (MDPO) algorithm with enabling auto-tuning for one of the key hyperparameters, $\lambda$. The key finding includes:

- Extend **off-policy, soft MDPO** to allow $\lambda$ auto-tuning, which uses neural networks as the function approximator to approximate the optimal value of $\lambda$ over time.
- Provide code implementation for MDPO-Auto, empirically examine the performance of MDPO-Auto on a popular simulated environment and achieve better or on-par performance compared to the baseline.

# 8 Future Work

- Due to the high computational cost and time limitation, we only evaluate the performance of MDPO, MDPO-Auto and SAC-Auto on one of Mujoco's environments, the performance of these algorithms can be further examined in more environments to demonstrate the result.

- In Haarnoja et al. [2018b] the authors examine the average return of the SAC algorithm 3 million timestamps and more. Experiments can be done to test whether the advantage of auto-tuning still holds when training for more timestamps.

# References


Mohammad Gheshlaghi Azar and Hilbert J. Kappen. Dynamic policy programming. *CoRR*, abs/1004.2027, 2010. URL `http://arxiv.org/abs/1004.2027`.

Roy Fox, Ari Pakman, and Naftali Tishby. G-learning: Taming the noise in reinforcement learning via soft updates. *CoRR*, abs/1512.08562, 2015. URL `http://arxiv.org/abs/1512.08562`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018a.

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

Kyungjae Lee, Sungjoon Choi, and Songhwai Oh. Sparse markov decision processes with causal sparse tsallis entropy regularization for reinforcement learning. *CoRR*, abs/1709.06293, 2017. URL `http://arxiv.org/abs/1709.06293`.

Manan Tomar, Lior Shani, Yonathan Efroni, and Mohammad Ghavamzadeh. Mirror descent policy optimization. *arXiv preprint arXiv:2005.09814*, 2020.