Mirror Descent Policy Optimization

Aditya Shirwatkar (21232) Aman Singh (22027)
Vishwas B C (19828)

Robert Bosch Centre for Cyber Physical Systems

April 17, 2023

Table of Contents

Introduction
Motivation

- **Trust-region based Algorithms** are a class of RL algos which use an extra term in their policy optimization(PO) to constrain the consecutive policies to remain close to each other eg. TRPO, PPO.
  - Stable learning since policy updates are close
- **Mirror-Descent (MD)** is a first-order trust-region optimization method for solving constrained convex problems.
  - MD has been investigated for PO in RL, previoulsy.
  - But the trust-region problems for policy update in RL cannot be solved in closed-form
- **Goal** is to derive a new algorithm, to solve the problem of the non-existence of closed-form solution

Introduction
Mirror Descent

Mirror Descent (MD) is a first order trust-region method for constrained convex optimization.

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

- In each iteration, MD minimizes a sum of two terms:
  1) A linear approximation of the objective function f at the previous estimate x_k, and
  2) A proximity term that measures the distance between the updated x_{k+1} and current x_k estimates.
- MD is considered a trust-region method since the proximity term keeps the updates x_k and x_{k+1} close to each other.

Introduction
Mirror Descent

MD offers the following iterative update rule for the above problem:

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

where $B_\psi\left(x, x_k\right) := \psi(x) - \psi\left(x_k\right) - \left\langle \nabla \psi\left(x_k\right), x - x_k \right\rangle$ is the Bregman divergence associated with a strongly convex potential function ψ, and t_k is a step-size determined by the MD analysis.

- When $\psi = \frac{1}{2}\|\cdot\|_2^2$, the Bergman divergence is the Euclidean distance $B_\psi\left(x, x_k\right) = \frac{1}{2}\left\|x - x_k\right\|_2^2$, and (2) becomes the *projected gradient descent algorithm*.

Mirror Descent in RL

The goal in RL is to find an optimal policy π^*, and two common notions of optimality, are:

1. $\pi^*(\cdot \mid s) \in \arg\max_{\pi} V^{\pi}(s), \quad \forall s \in \mathcal{S}$
   - Value function optimized over the entire state space $\mathcal{S}$
   - Mainly used in value function based RL algorithms
2. $\pi^* \in \arg\max_{\pi} \mathbb{E}_{s\sim\mu}\left[V^{\pi}(s)\right]$
   - A scalar i.e the value function at the initial state $(s \sim \mu)$ is optimized
   - More common in policy optimization
   - MDPO uses this optimal criteria

Mirror Descent in RL

Unlike the MD optimization problem, the objective function is not convex in π in either of the previous two RL optimization problems.

Despite this, previous works have derived MD-style RL algorithms for policy optimization with the update rule at each iteration k as,

$$\pi_{k+1} \leftarrow \arg\max_{\pi \in \Pi} \mathbb{E}_{s \sim \rho_{\pi_k}} \left[\mathbb{E}_{a \sim \pi} [A^{\pi_k}(s,a)] - \frac{1}{t_k} \mathrm{KL}(s; \pi, \pi_k) \right] \qquad (3)$$

where, Π is the policy space, $\rho_{\pi_k} \equiv (1-\gamma)\mathbb{E}\left[\sum_{t\geq 0} \gamma^t \mathbb{I}\{s_t = s\} \mid \mu, \pi\right]$ is the state frequency induced by the current policy π_k, $A^{\pi_k}(s,a) := Q^{\pi_k}(s,a) - V^{\pi_k}(s)$ is the advantage function, and KL(.) is the Kullback–Leibler divergence.

Mirror Descent Policy Optimization
Contribution

Problem

Trust region update rule (3) cannot be solved in closed form.

Solution

Replace the update rule (3) with multiple steps of Stochastic Gradient Descent (SGD) on the objective function. This is crucial for approximately solving each MD iterate

From now on we will define, Π to be the policy space of smoothly parameterized stochastic policies, i.e

$$\Pi = \{\pi(\cdot|s;\theta) : s \in \mathcal{S}, \theta \in \Theta\}$$

where θ is the policy parameter that will be optimized.

Mirror Descent Policy Optimization
Update Rule

Thus the optimization problem is, $\theta_{k+1} \leftarrow \arg\max_{\theta \in \Theta} \Psi(\theta, \theta_k)$

- **On-policy MDPO**:

$$\Psi(\theta, \theta_k) = \mathbb{E}_{s \sim \rho_{\theta_k}} \left[\mathbb{E}_{a \sim \pi_\theta} [A^{\theta_k}(s, a)] - \frac{1}{t_k} \mathrm{KL}(s; \pi_\theta, \pi) \right] \tag{4}$$

- **Off-policy MDPO**:

$$\Psi(\theta, \theta_k) = \mathbb{E}_{s \sim \mathcal{D}} \left[\mathbb{E}_{a \sim \pi_\theta} [A^{\theta_k}(s, a)] - \frac{1}{t_k} \mathrm{KL}(s; \pi_\theta, \pi) \right] \tag{5}$$

Proposed Update Rule

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

On-Policy MDPO

Algorithm

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

On-Policy MDPO
Key Takeaways

- MDPO performs better than or on par to TRPO, without requiring to enforce a hard constraint
- MDPO is more efficient than TRPO due to a reduced wall clock time
- MDPO consistently beats PPO by a considerable margin across all tasks

Table: Comparison of On Policy MDPO

	MDPO	TRPO	PPO
Hopper-v2	1964 ($\pm$217)	**2382 ($\pm$445)**	1281 ($\pm$353)
Walker2d-v2	**2948 ($\pm$298)**	2454 ($\pm$171)	424 ($\pm$92)

Off-Policy MDPO

Policy updates: Modify the objective function in (5) by the same reparametrization trick as SAC,

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

where, $\widetilde{a}_\theta(\epsilon,s)$ is the action generated by sampling the noise from a zero-mean normal distribution $\mathcal{N}$.

For a Gaussian Policy, $\widetilde{a}_\theta(\epsilon,s) = \tanh(\text{mean}_\theta(s) + \text{sigma}_\theta(s)\odot\xi)$; $\xi\sim\mathcal{N}(0,I)$

Value and Q network updates: Q network update is done in a TD(0) fashion, while the Value network update involves fitting the value function to the Q estimate of the current policy.

Off-Policy MDPO

Algorithm

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

Off-Policy MDPO

Key Takeaways

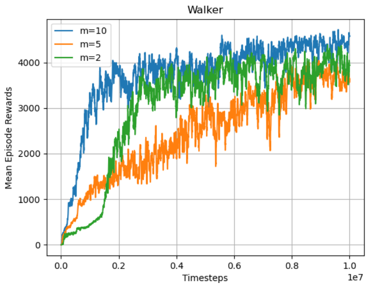
- MDPO offers more generality to viewing SAC
- MDPO performs better than or on par with SAC
- Off-policy MDPO has a better sample efficiency than on-policy MDPO
- Off-policy MDPO has a much higher wall clock time than on-policy MDPO

Table: Comparison of Off Policy MDPO

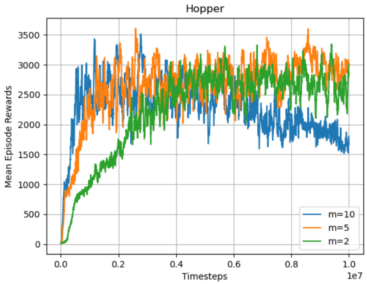
	MDPO-KL	SAC
Hopper-v2	1385 (±648)	**1501** (±414)
Walker2d-v2	873 (±180)	635 (±137)

Results

Reproduced Results: GitHub@aditya-shirwatkar/model-based-mdpo



(a) Walker



(b) Hopper

Figure: Performance of on-policy MDPO for different values of m (gradient steps)

Future Work

Current Literature

So far MD-style algorithms have been of model-free learning type

Novelty

We propose to learn model-ensemble to generate dynamics rollout, which will be then used to update the policy parameters with MDPO

Motivation[1]

- Model-free methods tend to suffer from high sample complexity, and Model-based methods reduce sample complexity, but tend to require careful tuning
- An ensemble of models maintains the model uncertainty and regularizes the learning process

[1] Kurutach, T. et al. Model-ensemble trust-region policy optimization.

Future Work

Algorithm Model Ensemble Mirror Descent Policy Optimization

```
1:  Initialize a policy $\pi_\theta$, all models $\hat{f}_{\phi_1}, \hat{f}_{\phi_2}, ..., \hat{f}_{\phi_K}$
2:  Initialize an empty dataset $\mathcal{D}$;
3:  repeat
4:      Collect samples from real system $f$ using $\pi_\theta$ and add them to $D$;
5:      Train all models using $\mathcal{D}$;
6:      # Optimize $\pi_\theta$ using all models
7:      repeat
8:          Collect fictitious samples from $\{\hat{f}_{\phi_i}\}_{i=1}^{K}$ using $\pi_\theta$.
9:          Update the policy using MDPO (On Policy) on the fictitious
            samples;
10:         Estimate the expected return for $i = 1, ..., K$;
11:     until the expected return stops improving;
12: until the policy performs well in real environment $f$;
```

Thank You!