Improving Actor-Critic Reinforcement Learning via Hamiltonian Monte Carlo Method

Duo Xu, Faramarz Fekri
Department of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
`dxu301@gatech.edu, faramarz.fekri@ece.gatech.edu`

Abstract

The actor-critic RL is widely used in various robotic control tasks. By viewing the actor-critic RL from the perspective of variational inference (VI), the policy network is trained to obtain the approximate posterior of actions given the optimality criteria. However, in practice, the actor-critic RL may yield suboptimal policy estimates due to the amortization gap and insufficient exploration. In this work, inspired by the previous use of Hamiltonian Monte Carlo (HMC) in VI, we propose to integrate the policy network of actor-critic RL with HMC, which is termed as *Hamiltonian Policy*. As such we propose to evolve actions from the base policy according to HMC, and our proposed method has many benefits. First, HMC can improve the policy distribution to better approximate the posterior and hence reduce the amortization gap. Second, HMC can also guide the exploration more to the regions of action spaces with higher Q values, enhancing the exploration efficiency. Further, instead of directly applying HMC into RL, we propose a new leapfrog operator to simulate the Hamiltonian dynamics. Finally, in safe RL problems, we find that the proposed method can not only improve the achieved return, but also reduce safety constraint violations by discarding potentially unsafe actions. With comprehensive empirical experiments on continuous control baselines, including MuJoCo and PyBullet Roboschool, we show that the proposed approach is a data-efficient and easy-to-implement improvement over previous actor-critic methods.

1 Introduction

In continuous control, actor-critic RL algorithms are widely used in solving practical problems. However, searching optimal policies can be challenging due to instability and poor asymptotic performance. Specifically, most previous methods of actor-critic RL, such as KL regularization [37, 38] and maximum policy entropy [29, 12], essentially solve RL in the framework of variational inference (VI) [24], which infers a policy that yields high expected return while satisfying prior policy constraints. However, from this perspective, the policy network essentially performs amortized optimization [13, 24]. It means that most actor-critic RL algorithms, such as soft actor-critic (SAC) [16], optimize a network to directly output the parameters of policy distribution which approximate the posterior given the input state and optimality. While these schemes have improved the efficiency of VI by encoder networks [21, 33, 28], the output distribution of learned policy can be sub-optimal and far away from the target posterior, due to the insufficient expressivity of the policy network [9, 19]. This suboptimality is typically defined as the amortization gap [9], resulting into a gap in the RL objective.

The Hamiltonian Monte Carlo (HMC) has been used to improve VI in statistics [3, 46]. In this work, by leveraging the advantages of both VI and HMC [35, 45], we propose to initialize Hamiltonian dynamics (HD) with samples from an optimized variational distribution, so that we can break the expressive limitation of the variational distribution and hence fill in the amortization gap. Specifically, we propose to use HD to evolve the actions sampled from the policy network, so as to better approximate the target posterior and sample the actions with higher Q values, improving the efficiency of the exploration. We call this new policy integrated with HD as *Hamiltonian policy*. The proposed method offers several benefits. First, the gradient information in Hamiltonian policy can make the exploration more directionally informed, avoiding sampling too many actions in opposite directions. Moreover, the randomness of momentum vectors in HD can help sampled actions to jump over the local optima and make the agent to explore more unknown parts of the state space. Further, the proposed leapfrog operator in Hamiltonian policy, which generalizes HMC via gated neural networks, can also increase the expressivity of the base policy network and adapt to the changing target distribution defined by Q function. Finally, in safe RL tasks, we find that the Hamiltonian policy can not only improve the achieved return by boosting the exploration, but also reduce the safety constraint violations by discarding potentially unsafe actions according to Lyapunov constraints [4, 5].

Using empirical experiments, we evaluated the proposed method across a variety of benchmark continuous control tasks such as OpenAI Gym using the MuJoCo simulator [43] and the realistic PyBullet Roboschool tasks [8]. We show that the proposed method improves upon representative previous methods such as SAC [14] and SAC with normalizing flow policy [27], achieving both a better convergence rate and expected return. Additionally, we also empirically verify the advantage of our method in safe RL problems.

In experiments, we conduct ablation study of the proposed leapfrog and sensitivity analysis on hyper-parameters. Additionally, we also compare the proposed method with iterative amortization policy optimization [26]. And the action distribution of Hamiltonian policy is also visualized to show the improvement of expressivity.

2 Preliminary

In this section, we are going to introduce reinforcement learning (RL) as an Markov Decision Process (MDP). Then the constrained MDP and the solution based on Lagrangian method are introduced. We also formulate the RL problem in the framework of variational inference. Finally we briefly review the Soft Actor-Critic (SAC) [16] and Hamiltonian Monte Carlo (HMC) [31] as building blocks of the proposed method.

2.1 Markov Decision Process

We consider Markov decision processes (MDP) as $(\mathcal{S}, \mathcal{A}, p_{\text{env}}, r)$, where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are the state and action at time step t, with the corresponding reward $r_t = r(s_t, a_t)$. The state transition of the environment is governed by $s_{t+1} \sim p_{\text{env}}(s_{t+1}|s_t, a_t)$, and the action is sampled from the policy distribution, given by the policy network $\pi_\theta(a_t|s_t)$ with parameters θ. The discounted sum of rewards is denoted as $\mathcal{R}(\tau) = \sum_t \gamma^t r_t$, where $\gamma \in (0, 1]$ is the discounted factor, and $\tau = (s_1, a_1, \dots)$ is a trajectory. Thus, the distribution over the trajectory is

$$p(\tau) = \rho(s_1) \prod_{t=1}^{T} p_{\text{env}}(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{1}$$

where the initial state is drawn from the distribution $\rho(s_1)$. The objective of RL is to maximize the expected discounted return $\mathbb{E}_{p(\tau)}[\mathcal{R}(\tau)]$. At a given time step t, one can optimize this objective by estimating the accumulated future returns in the summation using an action-value network [28, 16], denoted as $Q_\pi(s, a)$ in terms of a policy π.

2.2 Constrained MDP and Lagrangian Method

Safety is an important issue in RL problems. We use constrained MDP (CMDP) to model RL problems in which there are constraints on the cumulative cost. The CMDP extends MDP by introducing a safety cost function and the associated constrained threshold, which is defined as $(\mathcal{S}, \mathcal{A}, p_{\text{env}}, r, c, d_0)$ where $c(s) \in [0, C_{\max}]$ is a state-dependent cost function and $d_0 \in \mathbb{R}_{>0}$ is an upper-bound on the expected cumulative safety cost in one episode. In addition to Q_π, we use another action-value network to approximate the accumulated future safety costs $\mathcal{C}(\gamma) := \mathbb{E}[\sum_t \gamma^t c_t]$, denoted as $Q_{C,\pi}$.

The Lagrangian method is a straightforward method to solve CMDP, by transforming it to a penalty form, i.e., $\max_\theta \min_\lambda \mathbb{E}[\sum_t \gamma^t (r(s_t, a_t) - \lambda c(s_t)|\pi_\theta, s_0]$. The parameters θ and λ are jointly optimized to a saddle-point. The policy parameters θ are optimized by a policy gradient algorithm, while the multiplier λ is updated iteratively as $\lambda \longleftarrow [\lambda + \eta(J_C^\pi - d_0)]_+$, where J_C^π is the discounted sum (or average sum) of safety costs in previous episodes.

2.3 Reinforcement Learning via Variational Inference

Recently a surge of works have formulated reinforcement learning and control as probabilistic inference [10, 44, 42, 2, 24]. In these works, the agent-environment interaction process is formulated as a probabilistic graphical model, then reward maximization is converted into maximum marginal likelihood estimation, where the policy resulting the maximal reward is learned via probabilistic inference. This conversion is accomplished by introducing one or more binary optimality variables $\mathcal{O}$. Since calculating the likelihood of optimality $\mathcal{O}$ requires intractable integral over all the possible trajectories, variational inference (VI) is adopted to lower bound the objective, where a variational distribution $q(\tau|\mathcal{O})$ is learned to approximate the posterior of trajectory given the optimality, yielding the evidence lower bound (ELBO) [24]. The ELBO of the likelihood of optimality $\mathcal{O}$ can be written as below,

$$\begin{aligned}
&\log p(\mathcal{O} = 1) \\
&\quad \geq \int q(\tau|\mathcal{O}) \left[\log p(\mathcal{O} = 1|\tau) + \log \frac{p(\tau)}{q(\tau|\mathcal{O})} \right] d\tau \\
&\quad = \mathbb{E}_q[\mathcal{R}(\tau)/\alpha] - D_{\text{KL}}(q(\tau|\mathcal{O}) \| p(\tau))) \qquad (2)
\end{aligned}$$

where $D_{\text{KL}}(\cdot\|\cdot)$ denotes the KL divergence. Only model-free RL is considered here. We can simplify the EBLO in (2) by cancelling the probability of environmental dynamics. Then we can get the objective of policy optimization as below [24],

$$\mathcal{J}(q, \theta) = \mathbb{E}_{(s_t, r_t) \in \tau, a_t \sim q} \left[\sum_{t=1}^{T} \gamma^t r_t - \alpha \log \frac{q(a_t|s_t, \mathcal{O})}{\pi_\theta(a_t|s_t)} \right] \qquad (3)$$

Specifically, at time step t, this objective can be written as

$$\mathcal{J}(q, \theta) = \mathbb{E}_q[Q_q(s_t, a_t)] - \alpha D_{\text{KL}}(q(a_t|s_t, \mathcal{O}) \| \pi_\theta(a_t|s_t)) \qquad (4)$$

Hence, with π_θ as action prior, policy optimization in the framework of VI [16, 24] is to find optimal q maximizing the objective $\mathcal{J}(q, \theta)$ in (4).

2.4 Soft Actor-Critic

Soft Actor-Critic (SAC) [16] is a state-of-art off-policy RL algorithm widely used in many applications, especially in robotic problems with continuous actions and states. SAC can also be formulated from the perspective of variational inference. When using uniform distribution $\mathcal{U} = (-1, 1)$ as the action prior π_θ in (4), the objective of SAC can be formulated as the state-action value function regularized with a maximum entropy,

$$\mathcal{L}(q) = \mathbb{E}_{s_t \sim \rho_q} \big[\mathbb{E}_{a_t \sim q} Q_q(s_t, a_t) - \alpha \log q(a_t|s_t) \big]. \qquad (5)$$

where q is the variational distribution of action. Here ρ_q is the state distribution induced by policy q, and α is the temperature parameter which is introduced to improve the exploration. In this work, we are going to build the proposed method upon SAC. The optimal solution of (5) is $\bar{p}_\alpha(a|s) \propto \exp(Q(s, a)/\alpha)$ which is also the target policy distribution.

2.5 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a popular Markov chain Monte Carlo (MCMC) method for generating sequence of samples, which converge to being distributed according to the target distribution [31]. Inspired by physics, the key idea of HMC is to propose new points by simulating the dynamics of a frictionless particle on a potential energy landscape $U(x)$ induced by a desired target distribution $p(x)$, where $p(x) \propto \exp(-U(x))$. This simulation is done in the formulation of *Hamiltonian dynamics* (HD). Specifically, HD is a reformulation of physical dynamics whose states can be described by a pair (x, v) of d-dimensional vectors, where x is the position vector and

v is the momentum vector. The dynamics of the system over time, i.e., the HD, is described by the Hamiltonian equations:

$$\frac{dx}{dt} = \frac{dH}{dx}, \qquad \frac{dv}{dt} = -\frac{dH}{dv} \tag{6}$$

where $H(x, v)$ is the Hamiltonian of the system, defined as the total energy of the system. In the physical context of HMC, the motion of the frictionless particle is governed by the potential energy $U(x)$ and kinetic energy $K(v)$. Since the Hamiltonian is the total energy here, we have $H(x, v) = U(x) + K(v)$, which is independent of time step due to the conservation of energy. The kinetic energy can be described as $K(v) = \beta v^T v/2$ where β is the mass of the particle, and the momentum vector is distributed as $p(v) \propto \exp(-\beta v^T v/2)$ [45].

The analytic solutions of HD (6) can determine three important properties of HMC algorithm, i.e., *reversibility, volume preservation* and *Hamiltonian conservation*. The reversibility means that the mapping T_s from the state (x_t, v_t) at time t to some future state at time $t + s(s > 0)$ is one-to-one and reversible. The volume preservation implies that the transformation based on HD conserves the volume in state space, i.e., applying T_s to some region results in another region with the same volume. Finally, the Hamiltonian $H(x, v)$ stays constant with time, i.e., $dH/dt = 0$, which is called Hamiltonian conservation.

The HD described in (6) is typically simulated by the *leapfrog operator* [23, 31], of which the single time step can be described as

$$v^{\frac{1}{2}} = v - \frac{\epsilon}{2}\partial_x U(x); \;\; x' = x + \epsilon v^{\frac{1}{2}}; \;\; v' = v^{\frac{1}{2}} - \frac{\epsilon}{2}\partial_{x'} U(x'); \tag{7}$$

which transforms (x, v) to (x', v'). We can see that transformations in (7) are all volume-preserving shear transformations, where in every step only one of variables (x or v) changes, by an amount determined by the other one. Hence the Jacobian determinant of (7) is simply 1 and the density of transformed distribution $p(x', v')$ is tractable to compute.

3 Related Work

There have been a lot of previous works on improve policy optimization in recent years. To optimize the Q-value estimator with an iterative derivative-free optimizer, Qt-opt [18] uses the cross-entropy method (CEM) [34] to train robots to grasp things. To improve the model-predictive control, CEM and related methods are also used in model-based RL [30, 6, 32, 36].

However, there are less recent works on gradient-based policy optimization [17, 39, 1, 26]. They are specifically designed for model-based RL [17, 39, 1]. Normalizing flow [14, 40, 27] is another method to improve the policy optimization, by increasing the expressivity of the policy network. But none of them include gradient information, so that exploration is not sufficient in some environments. Another significant challenge with this approach is the Jacobian determinant in the objective, which is generally expensive to compute. Previous methods make the Jacobian determinant easy-to-evaluate at the sacrifice of the expressivity of the transformation, where the determinant only depends on the diagonal [22, 40, 27], limiting the exploration in the RL process.

Another approach is to apply iterative amortization in policy optimization, which uses gradients of Q function to iteratively update the parameters of the policy distribution [15, 26]. However, especially when the estimation bias of Q functions is significant [7], directly using gradients to improve policy distributions without additional randomness may make the policy search stuck at local optima which limits the exploration, and the policy distribution therein is still Gaussian. That is why methods in [26] cannot outperform SAC in many MuJoCo environments. Finally, one more related work is Optimistic Actor Critic (OAC) [7] using gradient of value function to update actions in exploration. However, their updated policy distribution is still Gaussian which has limited expressivity and one-step update with gradient is not enough to have significant performance improvement.

4 Methodology

In this section, we first formulate the proposed policy optimization method in the framework of variational inference (VI), and then propose a new leapfrog operator to quickly adapt to the changes of the target distribution during the learning. Finally implementation considerations and its application into safe RL are introduced.

4.1 Hamiltonian Policy Optimization

We call the method of using Hamiltonian policy into RL as Hamiltonian policy optimization (HPO). A feature of HPO is that a momentum vector ρ is introduced to pair with the action a in dimension d_a, extending the Markov chain to work in a state space $(a, \rho) \in \mathbb{R}^{d_a} \times \mathbb{R}^{d_a}$. Specifically, the momentum vector ρ has Gaussian prior $\mathcal{N}(\rho|0, \beta_0^{-1}I)$, and the action a follows the uniform prior $\mathcal{U}(-1,1)^{d_a}$, where β_0 is a hyper-parameter which determines variance of ρ. According to the discussion in Section 2.3 and 2.4, the target distribution in HPO, i.e., target posterior of action and momentum vector, can be written as

$$\bar{p}_\alpha(a, \rho|s) \propto \exp\big(Q_{\pi_\theta}(s,a)/\alpha\big)\mathcal{N}(\rho|0, \beta_0^{-1}I) \tag{8}$$

Therefore, following Section 2.5, the target potential function and momentum kinetic energy can be written as $U_\theta(s,a) := -Q_{\pi_\theta}(s,a)/\alpha$ and $K(\rho) := \frac{\beta_0}{2}\rho^T\rho$.

The core innovation of HPO is to evolve a and ρ via Hamiltonian dynamics (HD) in (6), where HD is approximated by steps of deterministic transitions (leapfrog in (7)), so that the evolved actions would likely better approximate the target posterior.

In Hamiltonian policy, given input state s, the initial action a_0 and momentum vector ρ_0 are sampled as $(a_0, \rho_0) \sim \pi_\theta(\cdot|s)\mathcal{N}(0, \beta_0^{-1}I)$ where π_θ is the base policy network. Then, by using leapfrog operator in (7) to simulate HD, since $U_\theta(s,a) := -Q_{\pi_\theta}(s,a)/\alpha$, action and momentum are evolved iteratively by the leapfrog (HMC step) as below,

$$\begin{aligned} \rho_{k+1/2} &= \rho_k + \frac{\epsilon}{2} \odot \nabla Q_{\pi_\theta}(s, a_k)/\alpha \\ a_{k+1} &= a_k + \epsilon \odot \rho_{k+1/2} \\ \rho_{k+1} &= \rho_{k+1/2} + \frac{\epsilon}{2} \odot \nabla Q_{\pi_\theta}(s, a_{k+1})/\alpha \end{aligned} \tag{9}$$

where ∇ is the differentiation taken with respect to a, $\epsilon \in \mathbb{R}$ is the learning step size, and $k = 0, \ldots, K-1$. Then by evolving initial action a_0 for K leapfrog steps, the action a_K is applied to the environment finally. The working process is shown in Figure 1(a)

Denote the k-th leapfrog step described above as $(a_k, \rho_k) := \Phi^k_{\theta,h}(a_{k-1}, \rho_{k-1})$. We can see that each leapfrog step still has unit Jacobian. Therefore, based on the change of variable formula in probability distribution, the joint distribution of action and momentum variables after K steps of leapfrog can be expressed as

$$\begin{aligned} q^K_{\theta,h}(a_K, \rho_K) &= q^0_{\theta,h}(a_0, \rho_0) \prod_{k=0}^{K-1} \big|\det \nabla \Phi^{k+1}_{\theta,h}(a_k, \rho_k)\big|^{-1} \\ &= \pi_\theta(a_0|s)\mathcal{N}(\rho_0|0, \beta_0^{-1}I) \end{aligned} \tag{10}$$

where (a_K, ρ_K) are action and momentum evolved by K HMC steps. Hence the density of output action and momentum vector becomes tractable to compute, facilitating the policy entropy regularization in SAC-style algorithms.

In the framework of VI, the policy optimization objective of HPO is the ELBO (2). Since ELBO can be written as the difference between the log of target distribution and log of variational distribution [21], we can write the ELBO for HPO as below,

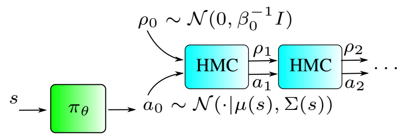
$$\mathcal{L}_{\text{ELBO}}(\theta, h; s) = \mathbb{E}_{(a_0,\rho_0)}[\log \bar{p}_\alpha(a_K, \rho_K|s) - \log q^K_{\theta,h}(a_K, \rho_K)] \tag{11}$$

The policy network parameters are denoted as θ and parameters in HMC are denoted as h.

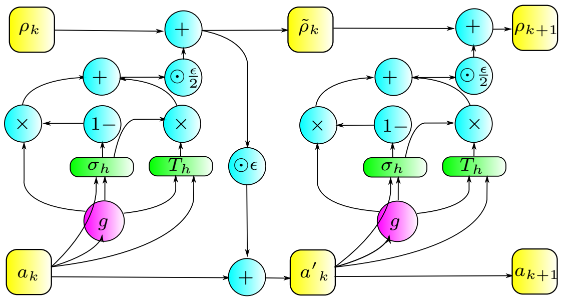
Finally, combining (8), (10) and (11) together and ignoring terms not related with θ and h, the objective of HPO, i.e., the expectation of EBLO over all the visited states, can be written as

$$\mathcal{J}(\theta, h) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[Q_{\pi_\theta}(s, a_K) - \alpha \log \pi_\theta(a_0|s) - \frac{\alpha\beta_0}{2}\rho_K^T\rho_K\right] \tag{12}$$

where ρ_{π_θ} is the state distribution induced by the policy π_θ. Note that α is the temperature parameter tuned in the same way as SAC [16]. And Q_{π_θ} is approximated by a target critic network which is not related with θ and is periodically updated in the learning process [14].


$\rho_0 \sim \mathcal{N}(0, \beta_0^{-1} I)$
HMC
ρ_1
a_1
HMC
ρ_2
a_2
s
π_θ
$a_0 \sim \mathcal{N}(\cdot|\mu(s), \Sigma(s))$

(a) Hamiltonian Policy



(b) Proposed Leapfrog

Figure 1: Diagram of the Hamiltonian policy and proposed leapfrog operator. The HMC box represents one step leapfrog operator.

4.2 Proposed Leapfrog Operator

Since HMC with conventional leapfrog (9) converges and mixes slowly, some past works proposed to use neural networks to generalize HMC [24, 25]. However, since Q networks are changing in RL, the techniques proposed in [24, 25] cannot be used here. Based on our empirical study, the direction variable, binary mask and exp operation therein [24, 25] can make the policy optimization unstable, degrading the RL performance.

Instead, we propose to use a gating-based mechanism to generalize the conventional leapfrog operator (9) and design a new leapfrog operator, which integrates gradient information via both explicit and implicit approaches. The explicit approach is to directly use the primitive gradient same as (9), whereas the implicit approach is to use an MLP T_h to transform the primitive gradient, state and action together. Then the gradient information from both explicit and implicit approaches are combined by a gate σ_h. The motivation behind is to improve the policy expressivity by MLP T_h and control the numerical stability by gate σ_h, making the policy distribution quickly adapt to the changes of Q networks during the learning process.

The inputs of T_h and σ_h include normalized gradients, action and state, where the state is optional and can be ignored in some environments. Therefore, the proposed leapfrog operation, transforming from (a_k, ρ_k) to (a_{k+1}, ρ_{k+1}), can be described as

$$\begin{aligned} \rho_{k+1/2} &= \rho_k - \frac{\epsilon}{2} \odot (\sigma_h(s, a_k, g) \odot g + (1 - \sigma_h(s, a_k, g)) \odot T_h(s, a_k, g)) \\ \rho_{k+1} &= \rho_{k+1/2} - \frac{\epsilon}{2} \odot (\sigma_h(s, a_{k+1}, g') \odot g' + (1 - \sigma_h(s, a_{k+1}, g')) \odot T_h(s, a_{k+1}, g')) \end{aligned} \quad (13)$$

where $g := \frac{-\nabla Q_\theta(s,a_k)}{\|\nabla Q_\theta(s,a_k)\|}$, $g' := \frac{-\nabla Q_\theta(s,a_{k+1})}{\|\nabla Q_\theta(s,a_{k+1})\|}$ and $a_{k+1} = a_k + \epsilon \odot \rho_{k+1/2}$. This process is shown in Figure 1(b). Obviously the proposed leapfrog operator (13) still keeps the properties of reversibility and unit Jacobian, so that the distribution of (a_K, ρ_K) in (10) is still tractable.

4.3 Implementation and Algorithms

In implementation, we build the Hamiltonian dynamics (HD) simulated by leapfrog steps on top of the policy network. Specifically, we only use one hidden layer for the base policy network π_θ, so the number of parameters of our model is much smaller than that of models in previous papers [14, 15, 26] which use two hidden layers in the policy network. The proposed RL algorithm is built on top of SAC, where the Gaussian policy is replaced by Hamiltonian policy and the policy optimization objective in (12) is used. It is termed as SAC-HPO. The process of producing actions from Hamiltonian policy is shown in Figure 1(a). The details of the algorithm are presented in Algorithm 2 and Algorithm 3 in Appendix.

4.4 Safe Reinforcement Learning with Hamiltonian Policy

In addition to regular RL, we also find the proposed method can be used in safe RL to reduce the safety violations. It is based on Lyapunov constraint [4, 5]. Under this constraint, based on their assumptions, the updated policy is provably guaranteed to satisfy the safety constraint. It transforms the trajectory-wise safety constraint to a state-wise constraint [5]. Specifically, the Lyapunov constraint is expressed as

$$Q_{C,\pi_B}(s,a) - Q_{C,\pi_B}(s,\pi_B(s)) < \tilde{\epsilon} \tag{14}$$

where

$$\tilde{\epsilon} = (1-\gamma)\cdot(d_0 - Q_{C,\pi_B}(s_0,\pi_B(s_0))) \tag{15}$$

where π_B is the reference policy which is the updated policy in last iteration, Q_{C,π_B} is the accumulated safety costs in terms of policy π_B, s_0 is the initial state and γ is the discounting factor of the MDP.

However, in previous work [4, 5], the sampled actions are made to satisfy the Lyapunov constraints based on the linear approximation of the cost critic Q_{C,π_B}, which is inaccurate in practice since the action may not have a linear relationship with the future costs. In this paper, we propose to use Hamiltonian policy to iteratively sample actions until the Lyapunov constraint is satisfied, so that potentially unsafe actions can be discarded in this process.

When interacting with the environment, for any sampled action, the agent first uses Lyapunov-constraint to predict its safety violation. If the Lyapunov-constraint is satisfied, it is applied into the environment. Otherwise, the sampled action will be updated by the leapfrog operator to get the next sampled actions, until the sampled actions satisfy the Lyapunov constraint. So, HMC here can not only boost the exploration by randomness and gradient information, but also improve safety in exploration by sampling actions iteratively until the safety (Lyapunov) constraint is satisfied. The application of Hamiltonian policy in safe RL is summarized in Algorithm 1. Note that compared with regular RL, a difference of Hamiltonian policy in safe RL is that the random noise is injected into momentum variables in every leapfrog step, rather than initial step only.

Algorithm 1: Hamiltonian Policy in safe RL; $s, \pi_\theta, \beta_0, \epsilon, K, d_0$

```
1 Sample $a_0 \sim \pi_\theta(\cdot|s), \rho_0 \sim \mathcal{N}(0, \beta_0^{-1} I)$;
2 for $k = 0, \dots, K-1$ do
3     Transform $(a_k, \rho_k)$ to $(a_{k+1}, \rho_{k+1})$ by the proposed leapfrog (13);
4     if $(a_{k+1}, \rho_{k+1})$ satisfies Lyapunov constraint (14) then
5         Return $a_{k+1}, \rho_{k+1}$;
6     end
7     Sample $\tilde{\rho} \sim \mathcal{N}(0, \beta_0^{-1} I)$, and update $\rho_{k+1} \longleftarrow \rho_{k+1} + \tilde{\rho}$;
8 end
9 Return $a_K, \rho_K$;
```

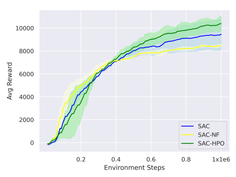
5 Experiment

In experiments, the Hamiltonian policy is applied into soft actor critic (SAC), so the proposed method is denoted as "SAC-HPO". The environments in our experiments are diverse, ranging from OpenAI Gym MuJoCo [43] to the realistic Roboschool PyBullet suit [8, 11]. We empirically evaluate the proposed method from many perspectives. First, SAC-HPO is compared with the primitive SAC [16] and SAC-NF [27] to show our advantage over classical SAC and normalizing flow policy. Second, we show the advantage of Hamiltonian policy in two MuJoCo environments with safety constraints, comparing with SAC-Lagrangian [41, 5].

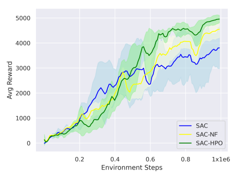
In addition, we conduct ablation study on the proposed leapfrog operator and the sensitivity analysis of hyper-parameters in Section C. The effect of random momentum variables in HMC is also evaluated in Section C.3. Finally, in Section C.4, we also analyze the shape of action distribution after leapfrog steps, verifying its non-Gaussianity and improvement of expressivity.

5.1 Continuous Control Tasks

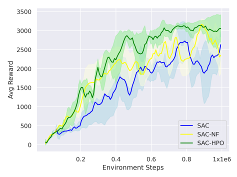
We compare SAC-HPO with SAC and SAC-NF on eight continuous control tasks. SAC is chosen because it is a fundamental learning method in actor-critic RL. SAC-NF is selected since it is a representative and widely-used method which adopts normalizing flow policy to improve the exploration. We use the official implementation of SAC [16]. And we try our best to implement SAC-NF according to [27], where the policy network is one-layer MLP with 256 hidden units and ReLU activation and radial normalizing flow is adopted. The learning curves are shown in Figure 2, where first five tasks, corresponding from Figure 2(a) to Figure 2(e), are from the MuJoCo suite and the other three are from Roboschool PyBullet.



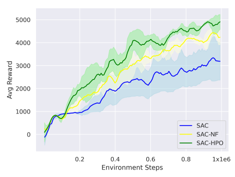
(a) HalfCheetah-v2



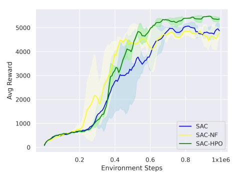
(b) Walker2d-v2



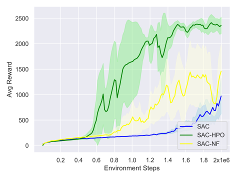
(c) Hopper-v2



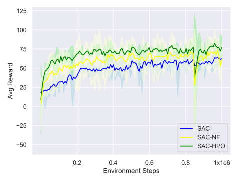
(d) Ant-v2



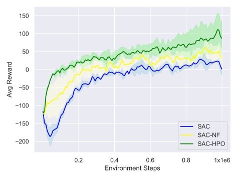
(e) Humanoid-v2



(f) Humanoid PyBullet



(g) Flagrun PyBullet



(h) Flagrun Harder PyBullet

Figure 2: The learning performance comparison over 8 tasks. All the curves are averaged over 5 random seeds, where shadowed regions are standard deviations.

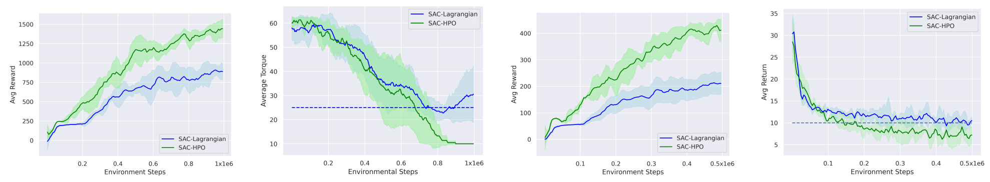
All the methods use the same architecture for Q networks, hyper-parameters, and tuning scheme for the temperature α. The critic (Q) networks follows the same architecture as [16], i.e., two-layer fully-connected neural networks with 256 units and ReLU activation in each layer, where two Q networks are implemented and trained by bootstrapping. All networks are updated by Adam optimizer [20] with the learning rate of 3e-4. The batch size for updating policies and critics is 256, and the size of replay buffer is 10^6. In SAC, the policy network consists of two fully-connected hidden layers with 256 units and ReLU activation.

In SAC-HPO, actions are evolved by HD simulated by leapfrog operations (13) for $K \in \{1, 2, 3\}$ steps. The base policy only has one hidden layer with 256 units and ReLU activation. Neural networks in proposed leapfrog (T_h and σ_h) are simple MLPs having one hidden layer with h_n hidden units and ELU activation. The variances of the momentum vector (β_0) should be different for training and exploration, denoted as β_0^{tr} and β_0^{exp} respectively. In most experiments, we find the variance of momentum ρ_0 in exploration should be larger than that in training, i.e., $\beta_0^{\text{tr}} < \beta_0^{\text{exp}}$, which can improve exploration efficiency. Besides, it is important to make networks T_h and σ_h in leapfrog small, which can stabilize the learning process. Hyper-parameters used in SAC-HPO are shown in Table 1 in Appendix, including number of leapfrog steps (K), number of hidden units in T_h and σ_h (h_n), momentum variances for training and exploration (β_0^{tr} and β_0^{exp}), update rate in leapfrog (ϵ), and temperature parameter (α).

All results in Figure 2 show the evaluation performance. Evaluation happens every 10,000 environmental steps, where each evaluation score (accumulated rewards in one episode) is averaged over 10 runs. The values reported in the plots are smoothed by exponential moving averaging (EMA) with a window size of 5, equivalent to averaging every 50,000 steps to improve comparability. We can see that the SAC-HPO outperforms SAC and SAC-NF in terms of both convergence rate and performance.

5.2 Safe Reinforcement Learning

In this section, we evaluate the performance of Hamiltonian policy in safe RL problems. The environments in this section are Ant-v2 and HalfCheetah-v2 in MuJoCo suite. At each step the robot selects the amount of torque to apply to each joint. In Ant-v2, the safety constraint is on the amount of torque the robot decided to apply at each time step [41]. Since the target of the task is to prolong the motor life of the robot, the robot is constrained from using high torque values. This is accomplished by defining the constraint d_0 as the average torque the agent has applied to each motor. The constraint threshold on torques is set to be 25 in each episode. In HalfCheetah-v2, the safety constraint is that the speed of the robot should be less than 1, i.e., the constraint cost is $\mathbf{1}[|v| > 1]$ at each time step [5]. The constraint threshold d_0 is on the discounted sum of safety costs in each episode, which is set to be 10.



(a) Ant-Reward (b) Ant-Cost (c) HalfCheetah-Reward (d) HalfCheetah-Cost

Figure 3: Learning performance of Hamiltonian policy in safe RL.

In experiments, we learn critic networks for both return and accumulated costs (safety violations), denoted as $Q(s,a)$ and $Q_C(s,a)$ respectively. Both Q and Q_C are realized by two-layer MLP with 256 hidden units and ReLU activation in each layer.

The baseline is the Lagrangian-based SAC [5] which introduces a Lagrangian multiplier λ to balance between return and safety costs, shorted as SAC-Lagrangian. The policy learning objective is $\max_\theta \mathbb{E}_{s\sim\mathcal{B},a\sim\pi_\theta(\cdot|s)}[(Q(s,a)-\lambda Q_C(s,a))-\alpha\log\pi_\theta(a|s)]$. And the multiplier is updated as $\lambda \longleftarrow [\lambda+\eta(J_C^{\pi_\theta}-d_0)]_+$ where $\eta=0.1$. Specifically, in Ant-v2, $J_C^{\pi_\theta}$ is the average sum of safety costs averaged in recent episodes, while in HalfCheetah-v2, $J_C^{\pi_\theta}$ is the discounted sum of safety costs averaged in recent episodes.

In SAC-HPO, the Lyapunov constraint is written as (14), where reference policy π_B is the policy updated in the last iteration, and Q_{π_B}, Q_{C,π_B} are target value networks of return and safety costs which are periodically updated in typical actor-critic RL algorithms. The working process of Hamiltonian policy in safe RL is summarized in Algorithm 1 in Section 4.4. In SAC-HPO, the policy learning objective is in the same form as (12), where Q_{π_θ} is replaced by $Q_{\pi_B}+\lambda Q_{C,\pi_B}$ and λ is updated in the same way as SAC-Lagrangian. The policy network and hyper-parameters of the SAC-HPO are same as Section 5.1, except that the maximum number of leapfrog K is set to be 10 which is larger than that in regular RL problems, and in practice the number of leapfrog steps taken actually is usually much smaller than K.

The performance comparison is presented in Figure 3, showing that our method not only improves the average return, but also reduces the safety violations. In safe RL, the learning objective contains $Q+\lambda Q_C$ instead of Q, where λ is changing in every learning iteration. Hence, since the target posterior of actions in (8) is defined in terms of $Q+\lambda Q_C$, the amortization gap in policy optimization is more significant than that in regular RL due to the rapid changes of λ. So HMC is more necessary here to make the sampled actions better approximate the target posterior. Moreover, iterative HMC sampling can discard potentially unsafe actions until safe actions are sampled. So, Hamiltonian policy can achieve more significant performance improvement in safe RL tasks than that in regular RL tasks.

6 Conclusion

In this work, we propose to integrate policy optimization with HMC, evolving actions from the base policy network by Hamiltonian dynamics simulated by leapfrog steps. In order to adapt to the changes of Q functions which define the target posterior, we propose a new leapfrog operator which generalizes HMC via gated neural networks. The proposed method can improve the efficiency of policy optimization and make the exploration more directionally informed. In empirical experiments, the proposed method can outperform baselines in terms of both convergence rate and performance. In safe RL problems, the Hamiltonian policy cannot only improve the achieved return but also reduce the number of safety constraint violations.

References

[1] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. In *Learning for Dynamics and Control*, pages 277–286. PMLR, 2020.

[2] Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in cognitive sciences*, 16(10):485–488, 2012.

[3] Anthony L Caterini, Arnaud Doucet, and Dino Sejdinovic. Hamiltonian variational auto-encoder. In *Advances in Neural Information Processing Systems*, pages 8167–8177, 2018.

[4] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *arXiv preprint arXiv:1805.07708*, 2018.

[5] Yinlam Chow, Ofir Nachum, Aleksandra Faust, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*, 2019.

[6] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4759–4770, 2018.

[7] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann. Better exploration with optimistic actor critic. In *Advances in Neural Information Processing Systems*, volume 32, pages 1787–1798, 2019.

[8] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[9] Chris Cremer, Xuechen Li, and David Duvenaud. Inference suboptimality in variational autoencoders. In *International Conference on Machine Learning*, pages 1078–1086, 2018.

[10] Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

[11] Benjamin Ellenberger. Pybullet gymperium. `https://github.com/benelot/pybullet-gym`, 2019.

[12] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 202–211, 2016.

[13] Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

[14] Tuomas Haarnoja, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 1851–1860. PMLR, 2018.

[15] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, pages 1352–1361. PMLR, 2017.

[16] 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, 2018.

[17] Mikael Henaff, William F Whitney, and Yann LeCun. Model-based planning with discrete and continuous actions. *arXiv preprint arXiv:1705.07177*, 2017.

[18] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[19] Yoon Kim, Sam Wiseman, Andrew Miller, David Sontag, and Alexander Rush. Semi-amortized variational autoencoders. In *International Conference on Machine Learning*, pages 2678–2687, 2018.

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

[21] Diederik P Kingma and Max Welling. Stochastic gradient vb and the variational auto-encoder. In *Second International Conference on Learning Representations, ICLR*, volume 19, 2014.

[22] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 29:4743–4751, 2016.

[23] Benedict Leimkuhler and Sebastian Reich. *Simulating hamiltonian dynamics*. Number 14. Cambridge university press, 2004.

[24] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[25] Lingge Li, Andrew Holbrook, Babak Shahbaba, and Pierre Baldi. Neural network gradient hamiltonian monte carlo. *Computational statistics*, 34(1):281–299, 2019.

[26] Joseph Marino, Alexandre Piché, Alessandro Davide Ialongo, and Yisong Yue. Iterative amortized policy optimization. *arXiv preprint arXiv:2010.10670*, 2020.

[27] Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pages 430–444. PMLR, 2020.

[28] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799, 2014.

[29] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[30] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[31] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[32] Alexandre Piché, Valentin Thomas, Cyril Ibrahim, Yoshua Bengio, and Chris Pal. Probabilistic planning with sequential monte carlo methods. In *International Conference on Learning Representations*, 2018.

[33] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[34] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[35] Tim Salimans, Diederik Kingma, and Max Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.

[36] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

[37] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

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

[39] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR, 2018.

[40] Yunhao Tang and Shipra Agrawal. Boosting trust region policy optimization by normalizing flows policy. *arXiv preprint arXiv:1809.10326*, 2018.

[41] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2018.

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

[43] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.

[44] Marc Toussaint and Amos Storkey. Probabilistic inference for solving discrete and continuous state markov decision processes. In *Proceedings of the 23rd international conference on Machine learning*, pages 945–952, 2006.

[45] Christopher Wolf, Maximilian Karl, and Patrick van der Smagt. Variational inference with hamiltonian monte carlo. *arXiv preprint arXiv:1609.08203*, 2016.

[46] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

A Algorithms

Algorithm 2: Hamiltonian Policy Optimization

```
1  Denote $a_t, s_t$ as the action and state at timet; Denote the replay buffer as $\mathcal{B}$;
2  Initialize $\theta, h$;
3  for $t = 1, 2, \ldots$ do
4      Sample $a_t \sim \pi_{\theta_t}(\cdot|s_t)$;
5      Obtain $a_t^K, \rho_t^K = f_{\text{HMC}}^K(s_t, a_t; \theta_t, h_t)$;
6      Apply $a_t^K$, and obtain next state $s_{t+1}$;
7      Store the experience tuple $(s_t, a_t^K, s_{t+1})$ into $\mathcal{B}$;
8      Sample a minibatch of transitions $\mathcal{D}_t$ from $\mathcal{B}$;
9      Update the Q network by $\mathcal{D}_t$;
10     Update $\theta$ and $h$ by optimizing $\mathcal{J}(\theta, h)$ in (12) with minibatch $\mathcal{D}_t$;
11 end
```

Algorithm 3: $f_{\text{HMC}}^K(s, a; \theta, h), \beta_0, \epsilon$

```
1  Sample $\rho^0 \sim \mathcal{N}(0, I)$;
2  Set $\rho_0 \longleftarrow \rho_0 \cdot \frac{1}{\sqrt{\beta_0}}$;
3  for $k = 1, \ldots, K$ do
4      Obtain $\rho_{k+1/2}$ by the first equation in (13);
5      Update $a_k = a_{k-1} + \epsilon \odot \rho_{k+1/2}$;
6      Obtain $\rho_{k+1}$ by the second equation in (13);
7  end
8  Return $a_K, \rho_K$;
```

B Hyper-parameter Table

	α	$1/\sqrt{\beta_0^{\text{tr}}}$	$1/\sqrt{\beta_0^{\text{exp}}}$	ϵ	K	l	h_n	m	$\mathcal{B}$ size
HalfCheetah-v2	0.2	1	0.5	0.2	3	1	32	256	10^6
Hopper-v2	0.2	0.1	1.5	0.15	2	1	32		
Walker2d-v2	0.2	0.2	1.5	0.15	3	1	32		
Ant-v2	0.2	0.1	1.0	0.1	3	1	32		
Humanoid-v2	0.05	1	1	0.1	3	1	64		
Humanoid PyB.	0.05	0.4	1.5	0.2	3	1	64		
Flagrun	0.05	0.2	1	0.15	3	1	32		
Flagrun Harder	0.05	0.2	1.5	0.15	3	1	64		

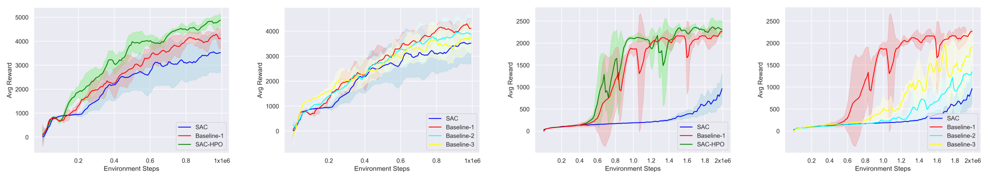
Table 1: Hyperparameters in SAC-HPO.

C Analysis

In this section, we conduct ablation study, sensitivity analysis and investigate the shape of the policy distributions evolved by HD.

C.1 Ablation Study

In ablation study, we first verify the effect of the proposed leapfrog operator (13) in comparison with the conventional leapfrog in (9). Then we study the differences of the effects of HMC in exploration and policy training, where the policy training (policy optimization) refers to the training step for policy network and neural networks T_h, σ_h in leapfrog (13).



(a) Ant-v2 (b) Ant-v2 (c) Humanoid PyBullet (d) Humanoid PyBullet

Figure 4: Performance Comparison in Ablation Study.

Specifically, we introduce three baselines adopting different leapfrog operators in exploration and policy training, which are summarized in Table 2. Here "Gaussian Policy" means that the same policy network in SAC is directly used without evolving actions by HMC. "Conv. Leapfrog" refers to that actions are evolved by the conventional leapfrog in (9), and "Prop. Leapfrog" denotes that actions are evolved by the proposed leapfrog in (13). We cannot use the proposed leapfrog only in exploration, since the neural networks T_h, σ_h in (13) need to be trained.

	Exploration	Policy Training
SAC-HPO	Prop. Leapfrog	Prop. Leapfrog
Baseline-1	Conv. Leapfrog	Conv. Leapfrog
Baseline-2	Conv. Leapfrog	Gaussian Policy
Baseline-3	Gaus. Policy	Prop. Leapfrog
SAC	Gaussian Policy	Gaussian Policy

Table 2: Exploration and Training Strategies in Baselines

The SAC-HPO and baselines are evaluated over Ant-v2 and HumanoidPyBulletEnv-v0, and learning curves are shown in Figure 4, where we use the same hyper-parameters as Section 5.1. It can be seen that in Figure 4(a) and 4(c), SAC-HPO outperforms the baseline-1 in terms of both performance and learning stability, showing the advantage of the proposed leapfrog operator over the conventional counterpart. And this advantage can also be observed in other environments.

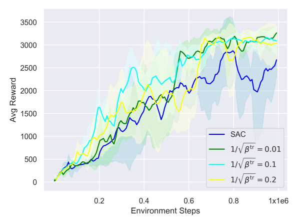
In Figure 4(b) and 4(d), the baseline-1 outperforms both baseline-2 and baseline-3, showing the effects of leapfrog in both exploration and policy optimization. Further, we can see that the improvement of baseline-2 over SAC is higher than that of baseline-3 over SAC, meaning that using HMC in exploration can yield more performance improvement than that in policy training.

C.2 Sensitivity Analysis

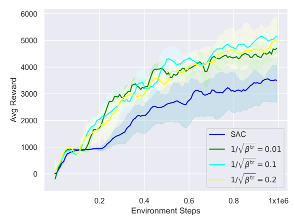
Here we conducted sensitivity analysis on three important hyper-parameters. The first is the number of leapfrog steps in simulating HD, i.e., $K = 1, 2, 3$, shown in Figures 5(g), 5(h) and 5(i). We can see that in Hopper-v2, the cases of $K = 2$ and $K = 3$ perform similar, but much better than that of $K = 1$, showing that it is not meaningful to have larger K. The second parameter analyzed here is the variance of momentum vector ρ in exploration, i.e., $1/\sqrt{\beta_0^{\text{exp}}} = 0.5, 1.0, 1.5$, shown in Figure 5(d), 5(e) and 5(f). We can see that the performance is sensitive to the choice of β^{exp}, showing the variance of ρ in exploration is important to the learning performance. Although more randomness in exploration can encourage the agent to explore more unseen states, not the highest choice of β^{exp} leads to best performance, such as Figure 5(e). That is because too much variance of ρ may make the sampled state-action pairs deviate too much from the optimal trajectory during the exploration. In Figure 5(a), 5(b) and 5(c), we analyze three different values for the variances for momentum vector ρ in policy training, i.e., $1/\sqrt{\beta_0^{\text{tr}}} = 0.01, 0.1, 0.2$. However, we can see that the performance is not sensitive to β_0^{tr}. According to more evaluations, larger β_0^{tr} cannot lead to better performance. For example when $\beta_0^{\text{tr}} = 1$, the performance degrades significantly.

C.3 Effect of Random Momentum Vector

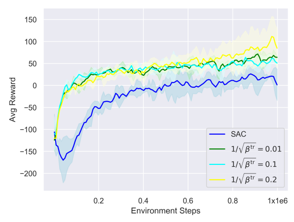
In Figure 6, we study the effect of the randomness of momentum vector ρ on the performance improvement. It is similar as the comparison with iterative amortized policy optimization (IAPO)



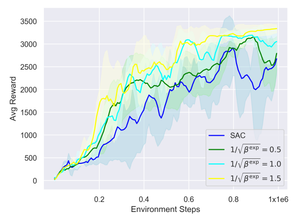
(a) Hopper-v2



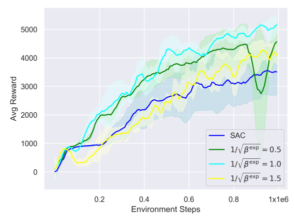
(b) Ant-v2



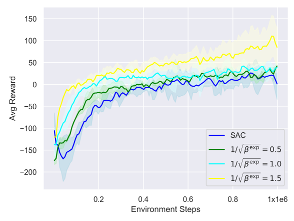
(c) PyBullet Flagrun Harder



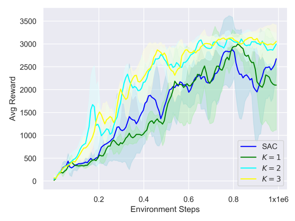
(d) Hopper-v2



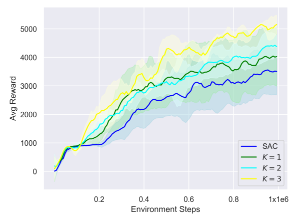
(e) Ant-v2



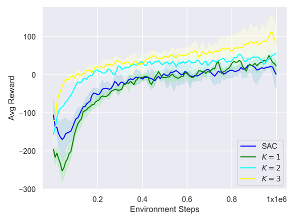
(f) PyBullet Flagrun Harder



(g) Hopper-v2

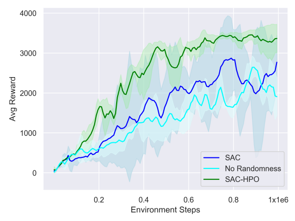


(h) Ant-v2

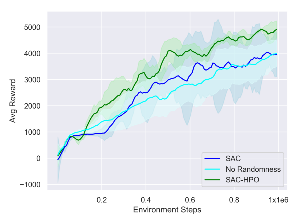


(i) PyBullet Flagrun Harder

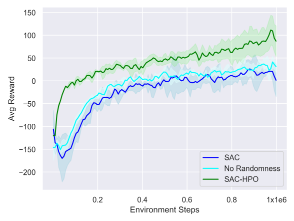
Figure 5: Sensitivity analysis on K, β^{exp}, and β^{tr}.



(a) Hopper-v2



(b) Ant-v2



(c) PyBullet Flagrun Harder

Figure 6: Performance comparison on the randomness of momentum vector. "No Randomness" refers to SAC-HPO without using random momentum vectors which is to set $\rho = 0$.

[26]. The baseline here is the SAC-HPO without random momentum vector, where we set $\rho = 0$ in both exploration and policy training. We can see that if no randomness in momentum vectors, the performance of SAC-HPO degrades significantly. Previous work such as IAPO [26] directly uses gradients to update the actions sampled from the base policy, without using any extra random variables, which is same as the baseline here. However, their performance is not good in high-dimensional environments [26]. The comparison in Figure 6 can explain the reason and show the advantage of our method. And similar advantage of our method can be observed in other environments as well.

C.4 Visualization of Policy Distribution

In Figure 7, we visualize the action distributions of Hamiltonian policy (actions evolved by HMC) at different environmental steps, where x and y axes represent two different action dimensions. Specifically, in Figure 7, the red dots represent 1000 actions sampled from the policy distribution evolved by leapfrog steps (13). For comparison, the blue dots represent 1000 actions sampled from

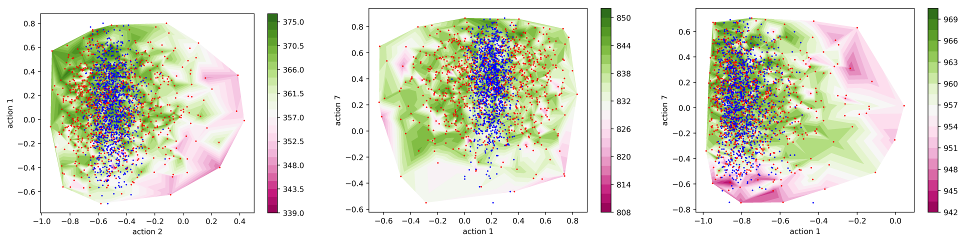

(a) 270K Step (b) 580K Step (c) 880K Step

Figure 7: The shape of policy distribution in Ant-v2. The dimensions of action are shown as x and y labels. The color bar is for Q values. $\beta_0^{\text{exp}} = 1$ for every step.

the base policy network π_θ directly, which are Gaussian and are not evolved by HMC. The contour of Q values is shown as background for reference, which is drawn by triangular interpolation method. In Figure 7, comparing red and blue dots, we can see that HMC can evolve actions sampled from the base policy more towards regions with higher Q values, making sampled actions more directionally informed and hence improving exploration efficiency. We can also observe that policy distribution evolved by leapfrog operators can be highly non-Gaussian and have larger variance with much broader effective support. Besides, there are still some actions evolved to regions with similar or lower Q values, so that a reasonable trade-off of exploration and exploitation can be reached. That is why the exploration of RL agent can be boosted by Hamiltonian policy and the learning performance can be improved.