EasyChair Preprint

№ 5307

Hamiltonian Policy Optimization in Reinforcement Learning

Derek Hsu

April 8, 2021

Hamiltonian Policy Optimization in Reinforcement Learning

Derek Hsu
Department of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA 30332
dxu301@gatech.edu, dxu301@gmail.com

Abstract

Approximating optimal policies in reinforcement learning (RL) is often necessary in many real-world scenarios, which is termed as policy optimization. By viewing the reinforcement learning 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 policy optimization may lead to 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 policy optimization with HMC. As such we choose evolving actions from the base policy according to HMC. First, HMC can improve the policy distribution to better approximate the posterior and hence reduces the amortization gap. Second, HMC can also guide the exploration more to the regions with higher action values, enhancing the exploration efficiency. Instead of directly applying HMC into RL, we propose a new leapfrog operator to simulate the Hamiltonian dynamics. With comprehensive empirical experiments on continuous control baselines, including MuJoCo, PyBullet Roboschool and DeepMind Control Suite, we show that the proposed approach is a data-efficient, and an easy-to-implement improvement over previous policy optimization methods. Besides, the proposed approach can also outperform previous methods on DeepMind Control Suite, which has image-based high-dimensional observation space.

1 Introduction

Reinforcement learning (RL) algorithms involve policy evaluation and policy optimization [38]. In continuous control the policy optimization can be challenging due to instability and poor asymptotic performance. In deep RL, where policies over continuous actions are often realized by deep neural networks, such issues are typically tackled by the regularization based on past learned policies [35, 36] or by maximizing the policy entropy [27, 11]. These techniques essentially solve RL in the framework of variational inference (VI) [22], using optimization to infer a policy that yields high expected return while satisfying prior policy constraints.

However, from this perspective, when used with entropy or KL regularization, policy networks essentially perform amortized optimization [12, 22]. It means that, many deep RL algorithms, such as soft actor-critic (SAC) [14], optimize a network to directly output the parameters of policy distribution, approximating the posterior given the input state and optimality. While these schemes have improved the efficiency of variational inference as encoder networks [19, 31, 26], the learned policy distribution can be sub-optimal and far away from the posterior, due to the insufficient expressivity of the policy network [8, 17]. This suboptimality is typically defined as the amortization gap [8], resulting into a gap in the RL objective.

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

The HMC has been used in VI in many previous works [3, 48]. Starting Hamiltonian dynamics with initial values of variables sampled from an optimized variational distribution, we can break the expressive limitation of the variational distribution and hence fill in the amortization gap, leveraging the advantages of both VI and HMC [33, 46]. In this work, we propose to use Hamiltonian dynamics (HD) to evolve the actions sampled from the policy network, so as to better approximate the posterior, i.e., the action with highest Q value, and make the exploration to be more directionally informed. We call this new policy integrated with HD as *Hamiltonian policy*. The conventional Gaussian policies in SAC [14] are directionally uninformed, where actions are sampled in arbitrary directions with equal probabilities, wasting a lot of samples. Moreover the expressivity of Gaussian policies is quite limited. However, 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 therein can help sampled actions to jump over the local optima and make the agent to explore more unknown parts in the state space. The proposed leapfrog operator in Hamiltonian policy, which generalizes HMC by neural networks, can also increase the expressivity of the base policy network and adapt to the target distribution defined by Q function which is changing during the learning process.

In this work we only consider stochastic policy, where the policy optimization is regularized by the policy entropy or KL distance [38, 14]. Hence the density of policy distribution should be tractable to compute. However, in most MCMC algorithms such as SGLD [45], the density of output variables is intractable or very expensive to compute, which cannot be used in policy optimization here. Because of the tractable Jacobian determinant, the density of output distribution of HMC is tractable to compute [29]. Hence the policy entropy of the Hamiltonian policy can be easily regularized in the objective, even though the policy distribution evolved by HD is non-Gaussian with complex shape.

Using empirical experiments, we evaluated the proposed method across a variety of benchmark continuous control tasks such as OpenAI Gym using the MuJoCo simulator [42], the realistic Bullet Roboschool tasks [6], and the DeepMind Control Suite [40] with high-dimensional observations. We show that the proposed method improves upon a already high performing method such as SAC and some other related methods, achieving better both convergence rate and performance. Additionally, we also conduct ablation study and sensitivity analysis, which shows the effects of the proposed neural architecture and hyper-parameters. Moreover, we also empirically evaluate the shape of the policy distribution produced by the Hamiltonian policy, verifying its non-Gaussianity and expressivity.

2 Preliminary

In this section, we are going to introduce reinforcement learning (RL) as an Markov Decision Process (MDP), and formulate the problem in the framework of variational inference. Then we briefly review the Soft Actor-Critic (SAC) [14] and Hamiltonian Monte Carlo (HMC) [29] as building blocks of the proposed method.

2.1 Markov Decision Process

We investigate Markov decision processes (MDP), 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 $\tau = (s_1, a_1, \ldots)$ 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 [26, 14], denoted as $Q_\pi(s, a)$ in terms of a policy π.

2.2 Reinforcement Learning via Variational Inference

Recently a surge of works have formulated reinforcement learning and control as probabilistic inference [9, 43, 41, 2, 22]. 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 observed variables $\mathcal{O}$, whose probability conditioned on the trajectory can be expressed as

$$p(\mathcal{O}=1|\tau) \propto \exp(\mathcal{R}(\tau)/\alpha) \tag{2}$$

where α is the temperature hyper-parameter. By referring variables $\mathcal{O}$ as optimality [22], our target is to learn the policy π_θ which can produce actions maximizing the likelihood of optimality. However evaluating this likelihood, i.e., $p(\mathcal{O}=1) = \int p(\mathcal{O}=1|\tau)p(\tau)d\tau$, needs the averaging over all the trajectories, which is computationally intractable especially in high-dimensions. Hence 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, i.e.,

$$q(\tau|\mathcal{O}) = \prod_{t=1}^{T} p_{\text{env}}(s_{t+1}|s_t, a_t)q(a_t|s_t, \mathcal{O}) \tag{3}$$

Since we focus on model-free RL here and the environment dynamics is unknown, what we can learn is the posterior of actions given the input state and optimality, i.e., $q(a_t|s_t, \mathcal{O})$, to maximize the evidence lower bound (ELBO) of the objective $\log p(\mathcal{O}=1)$, shown as below,

$$\begin{aligned}
&\log p(\mathcal{O}=1) \\
&\quad \geq \int q(\tau|\mathcal{O}) \left[\log p(\mathcal{O}=1|\tau) + \log \frac{p(\tau)}{q(\tau|\mathcal{O})} \right] d\tau \\
&\quad = \mathbb{E}_q[\mathcal{R}(\tau)/\alpha] - D_{\text{KL}}(q(\tau|\mathcal{O}) \| p(\tau)))
\end{aligned} \tag{4}$$

Simplifying this ELBO with (1) and (3), we can get the objective of policy optimization as below

$$\mathcal{J}(q, \theta) = \mathbb{E}_{(s_t, r_t)\in\tau, a_t\sim q} \left[\sum_{t=1}^{T} \gamma^t r_t - \alpha \log \frac{q(a_t|s_t, \mathcal{O})}{\pi_\theta(a_t|s_t)} \right] \tag{5}$$

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

$$\mathcal{J}(q, \theta) = \mathbb{E}_q[Q_q(s_t, a_t)] - \alpha D_{\text{KL}}(q(a_t|s_t, \mathcal{O}) \| \pi_\theta(a_t|s_t)) \tag{6}$$

where $D_{\text{KL}}(\cdot\|\cdot)$ denotes the KL divergence. Hence, with π_θ as action prior, policy optimization in the framework of VI [14, 22] is to find optimal q maximizing the objective $\mathcal{J}(q, \theta)$ in (6).

2.3 Soft Actor-Critic

Soft Actor-Critic (SAC) [14] is a state-of-art off-policy RL algorithm widely achieving success in many applications, especially in robotic problems with continuous actions and states. It updates the policy using gradient descent, minimizing the KL divergence between the policy and the Boltzmann distribution using the Q-function as its negative energy function. 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 (6), the objective of SAC can be formulated as the state-action value function regularized with a maximum entropy term, shown as below,

$$\mathcal{L}(q) = \mathbb{E}_{s_t\sim\rho_q} \big[\mathbb{E}_{a_t\sim q} Q_q(s_t, a_t) - \alpha \log \pi(a_t|s_t) \big]. \tag{7}$$

Here ρ_q is the state distribution induced by policy q, and α is the temperature parameter same as (2). In this work, we are going to build the proposed method upon SAC.

2.4 Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a popular MCMC method for generating sequence of samples, which converge to being distributed according to the target distribution [29]. 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, and the state of the physical system 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:

$$\frac{dx}{dt} = \frac{dH}{dx}, \qquad \frac{dv}{dt} = -\frac{dH}{dv} \tag{8}$$

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. Since 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 $p(v) \propto \exp(-\beta v^T v/2)$ [46].

The analytic solutions of HD (8) 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 refers that the transformation based on HD conserve the volume in state space, i.e., applying T_s to some region results in another region with the same volume. Finally, the Hamiltonian $H(x, v)$ keeps constant with time, i.e., $dH/dt = 0$, which is called Hamiltonian conservation.

The HD described in (8) is typically simulated by the *leapfrog operator* [21, 29], of which the single time step can be described as

$$v^{\frac{1}{2}} = v - \frac{\epsilon}{2}\partial_x U(x); \;\; x' = x + \epsilon v^{\frac{1}{2}}; \;\; v' = v^{\frac{1}{2}} - \frac{\epsilon}{2}\partial_{x'} U(x'); \tag{9}$$

which transforms (x, v) to (x', v'). We can see that transformations in (9) 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 (9) 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 [16] uses the cross-entropy method (CEM) [32] to train robots to grasp things. To improve the model-predictive control, CEM and related methods are also used in model-based RL [28, 4, 30, 34].

However, there are less recent works on gradient-based policy optimization [15, 37, 1, 24]. They are specifically designed for model-based RL [15, 37, 1]. Normalizing flow [13, 39, 25] 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 [20, 39, 25], 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 [24]. However, especially when the estimation bias of Q functions is significant [5], 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 the performance in high-dimensional environments is not satisfactory [24]. Finally, one more related work is Optimistic Actor Critic (OAC) [5] using gradient of value function to update actions in exploration. However, their updated policy distribution is still Gaussian without enough expressivity. Further one-step update with gradient is not enough to have significant performance improvement.

4 Methodology

In this work we only consider stochastic policy, where the policy optimization is regularized by the policy entropy or KL distance [38, 14]. Hence the density of policy distribution should be tractable to compute. However, in most MCMC algorithms such as SGLD [45], the proposal distribution is intractable or very expensive to compute, which cannot be used in policy optimization of stochastic policy. Because of the property of the tractable Jacobian determinant, the proposal distribution of HMC is tractable to compute [29], and Hamiltonian dynamics (HD) can be applied into policy optimization. In this work, we propose to use HD to evolve actions produced by the base policy network, so as to make the policy distribution better approximate the maximizer of the ELBO (5), i.e., the posterior of action given the state and optimality. Here we focus on the time-inhomogeneous HD [29, 3]. This method uses reverse kernels which are optimal for reducing variance of the likelihood estimators and allows for simple calculation of the approximate posteriors.

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 which is generalized by additional neural networks to quickly adapt to the changes of the target distribution during the learning. Additional considerations in implementation are also introduced.

4.1 Hamiltonian Policy Optimization

We call the proposed method as Hamiltonian policy optimization (HPO). Similar as HMC, we introduce momentum vector ρ 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. Therefore, in the framework of VI, the target distribution in policy optimization, i.e., unnormalized posterior of action and momentum vector, can be written as

$$\bar{p}_\alpha(s, a, \rho) \propto \exp\big(Q_{\pi_\theta}(s, a)/\alpha\big)\mathcal{N}(\rho|0, \beta_0^{-1} I) \tag{10}$$

Following the formulation in Section 2.4, we can have the potential function of the target $U_\theta(s, a) := -Q_{\pi_\theta}(s, a)/\alpha$ and the kinetic energy for the momentum vector $K(\rho) := \frac{\beta_0}{2}\rho^T\rho$. Essentially the variational distribution is the policy distribution being updated together with the momentum prior, i.e., $\pi_\theta(a|s)\mathcal{N}(\rho|0, \beta_0^{-1} I)$. The core idea of HPO is to improve the variational distribution via HD, where HD is simulated by steps of deterministic transitions (leapfrog), so that we can make the policy distribution better approximate the posterior. Essentially we evolve the action and momentum vector via deterministic transitions $q^k_{\theta,h}(a_k, \rho_k|a_{k-1}, \rho_{k-1}) = \delta_{\Phi^k_{\theta,h}(a_{k-1},\rho_{k-1})}(a_k, \rho_k)$, where θ denotes the parameters of policy π_θ in (10) and h represents trainable parameters in simulating HD, so that we can have

$$(a_K, \rho_K) = f^K_{\text{HMC}}(a_0, \rho_0; \theta, h) := \big(\Phi^K_{\theta,h} \circ \cdots \circ \Phi^1_{\theta,h}\big)(a_0, \rho_0) \tag{11}$$

where $(a_0, \rho_0) \sim \pi_\theta(\cdot|s)\mathcal{N}(0, \beta_0^{-1} I)$, and $\{\Phi^k_{\theta,h}\}_{k=1}^K$ define diffeomorphisms corresponding to a time-discretized and inhomogeneous Hamiltonian dynamics [3]. Different from normalizing flow policies in previous works [39, 44, 25], the gradient information of Q-function is incorporated, and the log density of output actions is easy to compute due to the tractable Jabcobian determinant, facilitating the policy entropy regularization in policy optimization.

Every transition $\Phi^k_{\theta,h}$ here consists of two steps: a leapfrog step, which discretizes the Hamiltonian dynamics, and a tempering step, which adds inhomogeneity to the dynamics and allows us to explore isolated modes of the target distribution [3, 29]. The leapfrog transforming (a, ρ) to (a', ρ') can be described as the following transformations:

$$\begin{aligned} \tilde{\rho} &= \rho - \frac{\epsilon}{2} \odot \nabla U_\theta(s, a) \\ a' &= a + \epsilon \odot \tilde{\rho} \\ \rho' &= \tilde{\rho} - \frac{\epsilon}{2} \odot \nabla U_\theta(s, a') \end{aligned} \tag{12}$$

where ∇ is the differentiation taken with respect to a, and step size $\epsilon \in \mathbb{R}$. We can see that the leapfrog (12) still has unit Jacobian. For the tempering step, the momentum output ρ' of each leapfrog step is multiplied by a scalar $\alpha_k \in (0, 1)$ for $k = 1, \ldots, K$. Since the target distribution defined

by Q-function is varying during the learning process and Q-function may have estimation bias, we adopt fixed tempering scheme proposed in [3], which can keep the learning process numerically stable. Specifically $\alpha_k = \sqrt{\beta_{k-1}/\beta_k}$ and β_k is a deterministic function of β_0 shown in (17), where $0 < \beta_0 \leq \beta_1 \leq \ldots \leq \beta_K = 1$. Then we can easily have that the Jacobian of every transition $\Phi^k_{\theta,h}$ is given by $|\det \nabla \Phi^k_{\theta,h}(a_k, \rho_k)| = \alpha_k^{d_a} = (\beta_{k-1}/\beta_k)^{d_a/2}$. Therefore, the joint distribution of action and momentum variables at K-th step of leapfrog $\Phi^K_{\theta,h}$ can be expressed as

$$\begin{aligned} q^K_{\theta,h}(a_K, \rho_K) &= q^0_{\theta,h}(a_0, \rho_0) \prod_{k=1}^{K} \left| \det \nabla \Phi^k_{\theta,h}(a_k, \rho_k) \right|^{-1} \\ &= q^0_{\theta,h}(a_0, \rho_0) \prod_{k=1}^{K} \left(\frac{\beta_{k-1}}{\beta_k} \right)^{-d_a/2} \\ &= \pi_\theta(a_0|s) \mathcal{N}(\rho_0|0, \beta_0^{-1} I) \beta_0^{-d_a/2} \end{aligned} \tag{13}$$

Hence the density of output action and momentum vector is tractable to compute, facilitating the policy entropy regularization in SAC-style algorithms.

Since HPO is in the formulation of VI, the policy optimization objective is just the ELBO (4). According to foundations of VI, the ELBO can be written as the difference between the log target and log variational distribution, i.e.,

$$\mathcal{L}_{\text{ELBO}}(s; \theta, h) = \mathbb{E}_{(a_0,\rho_0)\sim\pi_\theta(\cdot|s)\mathcal{N}(\cdot|0,I)}[\log \bar{p}(s, a_K, \rho_K) - \log q^K_{\theta,h}(a_K, \rho_K)] \tag{14}$$

where $(a_K, \rho_K) := f^K_{\text{HMC}}(a_0, \rho_0; \theta, h)$ in (11), and parameters to be trained consist of policy network parameters θ and parameters in HMC h.

Finally, combining (13) and (14) 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

$$\mathcal{J}_{\text{HPO}}(\theta, h) = \mathbb{E}_{s\sim\rho_{\pi_\theta}} \left[Q_{\pi_\theta}(s, a_K) - \alpha \log \pi_\theta(a_0|s) - \frac{\alpha\beta_0}{2} \rho_K^T \rho_K \right] \tag{15}$$

where ρ_{π_θ} is the state distribution induced by the policy π_θ, $(a_0, \rho_0) \sim \pi_\theta(\cdot|s)\mathcal{N}(\cdot|0, \beta_0^{-1} I)$ and $(a_K, \rho_K) = f^K_{\text{HMC}}(a_0, \rho_0; \theta, h)$ defined in (11). Specifically α is the temperature parameter tuned in the same way as SAC [14].

4.2 Theoretical Justification

By regarding the joint distribution of action and momentum $\pi_\theta(a|s)\mathcal{N}(0, \beta^{-1} I)$ as the variational distribution, we essentially improve it by running HD initialized by $(a_0, \rho_0) \sim \pi_\theta(a|s)\mathcal{N}(0, \beta^{-1} I)$, and the stationary distribution of action is the posterior of the target (10) with ρ marginalized which is denoted as $\bar{p}_{a,\alpha}(\cdot|s)$. Assume we run the leapfrog for k steps, by summing out ρ, the distribution of action is denoted as $q^k_{a,\theta}(\cdot)$. It is provable that given state s, the distribution $q^k_{a,\theta}(\cdot)$ is an improvement over the variational distribution $\pi_\theta(\cdot|s)$ and it is closer to the posterior of the target (10), i.e., $\bar{p}_{a,\alpha}(\cdot|s)$, in terms of KL divergence [7],

$$\text{KL}(\pi_\theta(\cdot|s)|\bar{p}_{a,\alpha}(\cdot|s)) \geq \text{KL}(q^k_{a,\theta}(\cdot)|\bar{p}_{a,\alpha}(\cdot|s))$$

Hence it reduces the *amortization gap*. According to [29], as the step $k \to \infty$, the distribution of action $q^k_{a,\theta}(\cdot)$ will tend to the posterior $\bar{p}_{a,\alpha}(\cdot|s)$ in terms of KL divergence. We empirically find that using $K = 2, 3, 4$ leapfrog steps is enough to yield satisfactory improvement.

4.3 Proposed Leapfrog Operator

In the practical policy optimization, the action value given by the Q network may have estimation bias, and directly incorporating gradient information like (12) may mislead the learning to wrong regions in the policy space, leading to suboptimal policy estimate. Since the Q value is changing in the learning process, how to make the policy distribution quickly adapt to updated Q networks in a numerically stable way is the desiderata here. The key is to improve policy expressivity without

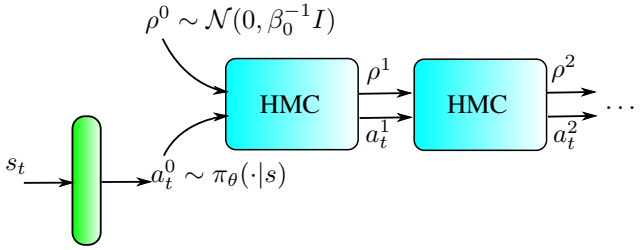


Figure 1: Diagram of Hamiltonian policy. The HMC box represents one leapfrog step.

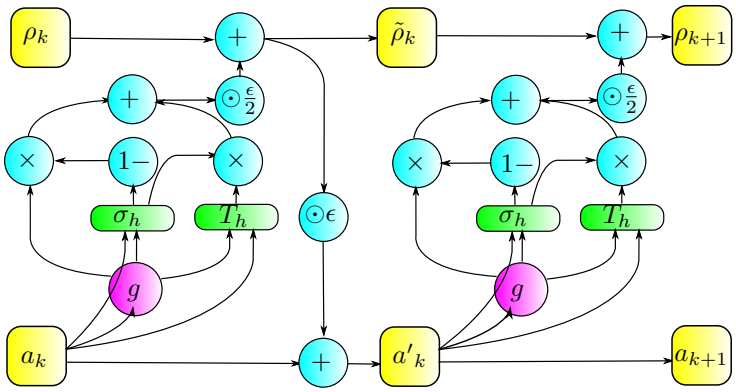


Figure 2: Diagram of the proposed leapfrog operator.

scarifying numerical stability. The previous work using neural network to generalize HMC [23] cannot be applied here, because based on our empirical study the direction variable, binary mask and exp operation therein can make the policy optimization unstable, degrading the RL performance.

Therefore, we use gating-based operation to generalize the conventional leapfrog operator(12), and propose 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 (12), 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 via 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.

According to empirical experiments we find that the gradient normalized by its l_2 norm can make the model perform best. 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, ρ) to (a', ρ'), can be described as

$$\begin{aligned} \tilde{\rho} &= \rho - \frac{\epsilon}{2} \odot (\sigma_h(s, a, g) \odot g + (1 - \sigma_h(s, a, g)) \odot T_h(s, a, g)) \\ \rho' &= \tilde{\rho} - \frac{\epsilon}{2} \odot (\sigma_h(s, a, g') \odot g' + (1 - \sigma_h(s, a, g')) \odot T_h(s, a, g')) \end{aligned} \tag{16}$$

where $g := \frac{\nabla U_\theta(s,a)}{\|\nabla U_\theta(s,a)\|}, g' := \frac{\nabla U_\theta(s,a')}{\nabla U_\theta(s,a')}$ and $a' = a + \epsilon \odot \tilde{\rho}$. Specifically, T_h and σ_h have different parameters contained in h, but they should have the same architecture based on empirical study. We find that the architecture of T_h and σ_h should be simple, having one hidden layer with 32 or 64 units. The output activations of T_h and σ_h are sigmoid and tanh respectively. The transformations in (16) are visualized in Figure 2. Obviously the proposed leapfrog operator (16) still keeps the properties of reversibility and tractable Jacobian determinant.

4.4 Implementation Considerations

In implementation, we build the Hamiltonian dynamics (HD) simulated by leapfrog steps on top of the policy network in both exploration and policy optimization. This new policy is termed as *Hamiltonian policy*, whose working mechanism is shown in Figure 1. The randomness of momentum vector ρ can boost the exploration and improve the generalization of the policy trained in the policy optimization. The neural networks T_h and σ_h in the proposed leapfrog operator can increase the expressivity of policy distribution, verified in the experiment section. In exploration, the policy distribution is expanded towards regions with higher Q values, which makes the exploration more directed and hence improves the sample efficiency.

Specifically, we find that the HD has different effects on performance in the exploration and policy optimization, which is analyzed in the experiment section. Using different momentum variances β_0 for exploration and policy optimization, denoted as β_0^{exp} and β_0^{tr} respectively, can yield the best empirical performance. And in general we set $\beta_0^{\text{exp}} > \beta_0^{\text{tr}}$, since the exploration needs more randomness than policy optimization.

In order to achieve the best empirical performance, we make some specific changes in the implementation of RL algorithms. First, it is better to omit the last term of the objective (15), i.e., $\frac{\alpha}{2}\rho_K^T\rho_K$, in the practical implementation of policy optimization. Second, different from previous works on HMC [33, 46, 23], the Metropolis-Hastings accept/reject step is omitted here. That is because the Q values are dramatically varying during the training of RL. If accept/reject step was applied here, the acceptance rate could be very low, which could limit the exploration and hence degrade the learning performance significantly. The chart of the proposed algorithm is shown in Algorithm 1, and HD simulated by leapfrog steps is described in Algorithm 2.

Algorithm 1: Hamiltonian Policy Optimization

Data: Denote a_t, s_t as the action and state at time$t$; Denote the replay buffer as $\mathcal{B}$;
Initialize θ, h;
for $t = 1, 2, \ldots$ **do**
  Sample $a_t \sim \pi_{\theta_t}(\cdot|s_t)$;
  Obtain $a_t^K, \rho_t^K = f_{\text{HMC}}^K(s, a; \theta_t, h_t)$;
  Apply a_t^K into the environment, and obtain next state s_{t+1};
  Store the experience tuple (s_t, a_t^K, s_{t+1}) into $\mathcal{B}$;
  Sample a minibatch of experience tuples $\mathcal{D}_t$ from $\mathcal{B}$;
  Update the Q network by $\mathcal{D}_t$;
  Obtain the evolved action and momentum $a^K, \rho^K = f_{\text{HMC}}^K(s, a; \theta_t, h_t), \forall (s, a) \in \mathcal{D}_t$;
  Update parameters θ and h by optimizing $\mathcal{J}(\theta, h)$ in (15);
end

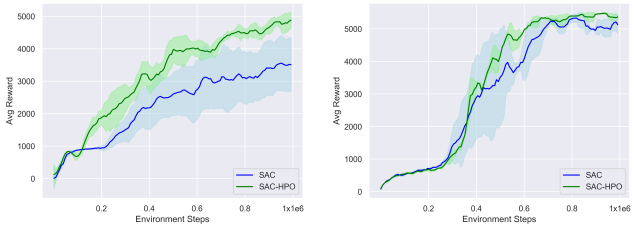
Algorithm 2: $f_{\text{HMC}}^K(s, a; \theta, h), \beta_0, \epsilon$

Sample $\rho^0 \sim \mathcal{N}(0, \beta_0^{-1} I)$;
for $k = 1, \ldots, K$ **do**
  Obtain $\tilde{\rho}^k$ by the first equation in (16);
  Update $a^k = a^{k-1} + \epsilon \odot \tilde{\rho}^k$;
  Obtain ρ^k by the second equation in (16);
  Compute β_k via

$$\sqrt{\beta_k} = \left(\left(1 - \frac{1}{\sqrt{\beta_0}}\right) \cdot \frac{k^2}{K^2} + \frac{1}{\sqrt{\beta_0}} \right)^{-1} \tag{17}$$

  Set $\rho_k \longleftarrow \rho_k \cdot \sqrt{\beta_{k-1}/\beta_k}$
end
Output: a^K, ρ^K

(a) HalfCheetah-v2 (b) Hopper-v2 (c) Walker2d-v2



(d) Ant-v2 (e) Humanoid-v2

Figure 3: The learning performance comparison of SAC-HPO and SAC over 5 MuJoCo tasks. All the curves are averaged over 4 random seeds, where shadowed regions are standard deviations. And every curve is smoothed by exponential moving averaging (EMA) with window size 5.

5 Experiment

In experiments, the proposed method is built on top of soft actor critic (SAC), having the same setup as [14], denoted as "SAC-HPO". In this section, we present the empirical evaluation of the proposed method from many perspectives. First, we compare SAC-HPO against the primitive SAC on a set of various continuous control tasks, including OpenAI Gym MuJoCo [42], the realistic Roboschool PyBullet suit [6] and DeepMind Control Suite [40]. Then we conduct ablation study on the proposed leapfrog operator and the specific effects of Hamiltonian dynamics (HD) simulated by the proposed leapfrog operator on the exploration and policy optimization analysis. Further the sensitivity analysis of hyper-parameters is also presented, which includes the number of leapfrog steps and variances of momentum vectors in exploration and policy optimization. Finally, we analyze the shape of policy distribution evolved by HD, verifying its non-Gaussianity and multi-modality.

5.1 Continuous Control Tasks

5.1.1 OpenAI Gym MuJoCo Benchmarks

First, the proposed method, SAC-HPO, is compared against the SAC on five continuous control tasks from the MuJoCo suite, shown in Figure 3. All result curves show the performance in evaluation, which, in the case of both methods, is equivalent to setting the noise (variance) to 0 at the policy network output. Evaluation happens every 10,000 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. The best obtained accumulated rewards reported in the tables are primitive values without being smoothed.

We use uniform action prior $p(a|s) \in \mathcal{U}(-1, 1)$ for both SAC-HPO and SAC. Both methods use the same architecture for Q networks, hyper-parameters such as batch size and learning rate, and tuning scheme for the temperature α. The policy network in SAC consists of two feed-forward hidden layers with 256 units. In SAC-HPO, we tried two kinds of policy networks, having one or two hidden layers, with details shown in Table 1. The output of policy network in SAC-HPO is further improved by leapfrog operations (16) for $K \in \{2, 3, 4\}$ steps, where the neural networks σ_h, T_h are chosen to be small, consisting of one feed-forward hidden layer with 32 or 64 units. The detailed architectures of σ_h, T_h used throughout the paper are shown in Table 2. And the output activation functions for T_h

and σ_h are tanh and sigmoid, respectively. Since the number of parameters of σ_h, T_h networks is only about 1% of that of the base policy network, the performance improvement of SAC-HPO does not come from the increase of trainable parameters, i.e., overparameterization. The critic (Q) networks follows the same architecture as [14], i.e., two-layer fully connected neural networks with 256 units in each layer, and two Q networks are implemented and trained by bootstrapping. All networks are updated by Adam optimizer [18] with the learning rate of 3e-4.

Both SAC-HPO and SAC have same choices for α, the temperature for entropy regularization. And the momentum random variables have different variance for training β^{tr} and exploration β^{exp}. The update rate ϵ in the leapfrog operator (16) is also carefully tuned for every task. All the hyperparameters for simulating HD are shown in Table 3. All the other important hyperparameters for running these experiments can be found in Table 4, where l represents the number of hidden layers in the policy network, h is the number of units in the hidden layer in T_h and σ_h networks, m denotes the batch size in training, and $\mathcal{B}$ represents the replay buffer.

Hyperparameter	Value
Number of Layers (l)	1 or 2
Number of Units/Layer	256
Activation	ReLU

Table 1: Policy Network

Hyperparameter	Value
Number of Layers	1
Number of Units/Layer (h)	32 or 64
Activation	ELU

Table 2: Neural Networks (T_h, σ_h) in (16)

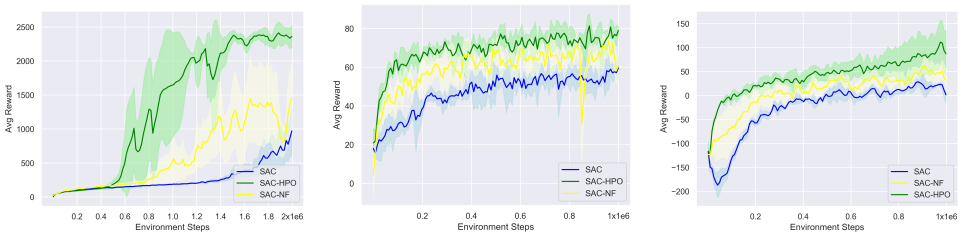
	α	$1/\sqrt{\beta_0^{\text{tr}}}$	$1/\sqrt{\beta_0^{\text{exp}}}$	ϵ
HalfCheetah-v2	0.2	1	0.5	0.2
Hopper-v2	0.2	0.1	1.5	0.15
Walker2d-v2	0.2	0.2	1.5	0.15
Ant-v2	0.2	0.1	1.0	0.1
Humanoid-v2	0.05	1	1	0.1
Humanoid PyBullet	0.05	0.4	1.5	0.2
Humanoid Flagrun PyBullet	0.05	0.2	1	0.15
Humanoid Flagrun Harder PyBullet	0.05	0.2	1.5	0.15

Table 3: Hyperparameters of HD in SAC-HPO.

	K	l	h	m	$\mathcal{B}$ size
HalfCheetah-v2	3	2	32	256	10^6
Hopper-v2	2	2	32		
Walker2d-v2	3	2	32		
Ant-v2	3	1	32		
Humanoid-v2	3	1	64		
Humanoid PyBullet	3	1	64		
Humanoid Flagrun PyBullet	3	2	32		
Humanoid Flagrun Harder PyBullet	3	2	64		

Table 4: Important Hyperparameters in SAC-HPO.

The learning performance of SAC-HPO and SAC are displayed in Figure 3. The Table **??** presents the best observed accumulated reward in evaluation. We can see that the SAC-HPO outperforms SAC in terms of both convergence rate and performance, achieving better sample complexity and higher accumulated rewards. There are primarily two reasons for that. First, the HD incorporating gradient information of Q networks reduces the amortization gap in policy optimization and hence breaks the expressivity limitation of the base policy network. Second, the HD simulated by the proposed leapfrog steps boost the exploration by the randomness of momentum vectors ρ and also inform the agent directions to regions with higher Q values. We can see that SAC-HPO has smaller regions of standard deviation for every task in Figure 3, and it shows that SAC-HPO achieves better *learning stability* than SAC. That is because the randomness of momentum vectors ρ in HD can make the



(a) Humanoid PyBullet (b) Humanoid Flagrun PyBullet (c) Humanoid Flagrun Harder PyBullet

Figure 4: Performance Comparison in 3 PyBullet Environments.

PyBullet Env Name	SAC	SAC-NF	SAC-HPO
Humanoid	1357 ± 269	1695 ± 258	$\mathbf{2518 \pm 219}$
Humanoid Flagrun	61 ± 25	148 ± 59	$\mathbf{164 \pm 32}$
Humanoid Flagrun Harder	45 ± 21	77 ± 34	$\mathbf{113 \pm 28}$

Table 5: The best observed average return with one standard deviation across 4 random seeds.

policy optimization (training of the policy network) cover more state-action pairs, improving the generalization capability of the learned policy network.

In particular, SAC-HPO can perform better than SAC on difficult tasks with high-dimensional states, including Ant-v2 and Humanoid-v2, even though previous methods such as iterative amortization policy optimization [24] fail in these tasks. We can also see that HalfCheetah-v2 is exceptional, where the momentum variance for policy optimization (training) ρ^{tr} needs more randomness than that for exploration ρ^{exp}. That is because in HalfCheetah-v2 exploration is less important than policy optimization for performance improvement.

5.1.2 Roboschool PyBullet Benchmarks

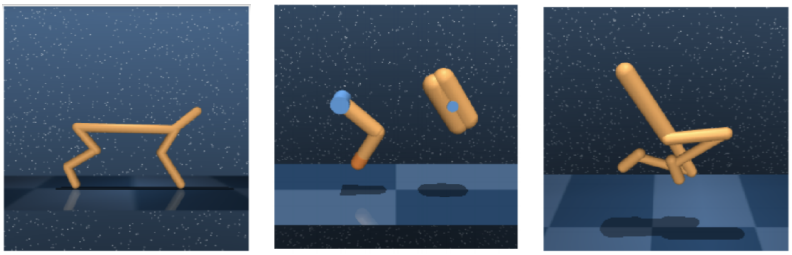
In addition to the classical tasks in MuJoCo suit, we also evaluate the proposed method, SAC-HPO, in the PyBullet implementation of Roboschool tasks [6]. The Bullet library is one of the most realistic collision detection and multi-physics simulation engines available up to now, which is widely used in many robotic tasks [10].

In this section, we introduce another baseline, SAC-NF, which builds a normalizing flow on top of the base policy network [39, 25]. According to the previous evaluations [25], we adopt the radial normalizing flow. Although authors did not release their codes, we try our best to implement to match their reported results. We compare SAC-HPO with SAC and SAC-NF in three representative PyBullet environments, including HumanoidPyBulletEnv-v0, HumanoidFlagrunPyBulletEnv-v0 and HumanoidFlagrunHarderPyBulletEnv-v0.

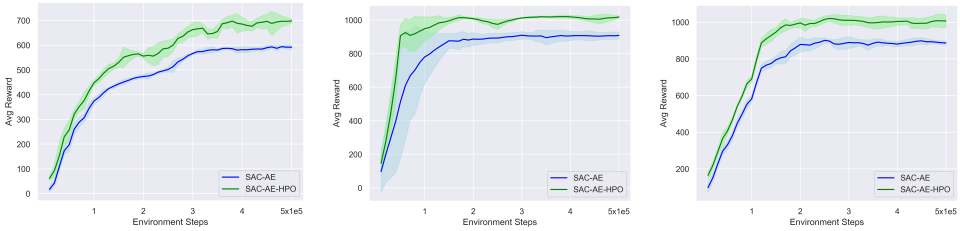
The learning performance is shown in Figure 4, where the evaluation scheme and smoothing method are same as the previous section. And the best observed accumulated rewards in evaluations are reported in Table 5.1.2. We can see that SAC-HPO performs best in terms of both convergence rate and obtained rewards. All these environments are more realistic and have high-dimensional state and action spaces, where the exploration capability is important. However, although the normalizing flows in method SAC-NF improves the expressivity of the policy distribution, in SAC-HPO the gradient information can make the exploration more directionally informed and the expressivity can be also improved by momentum vector ρ and neural networks T_h, σ_h in the proposed leapfrog operator.

5.1.3 DeepMind Control Suite

We also conduct experiments on DeepMind Control Suite [40], which have high-dimensional observations. The baseline is SAC-AE model [47], where the input to the policy and critic is the latent representation learned by an autoencoder. We apply HD on top of the policy in SAC-AE, termed as SAC-AE-HPO. And the gradients of critics are conditioned on the latent representation,



(a) cheetah_run (b) finger_spin (c) walker_walk



(d) cheetah_run (e) finger_spin (f) walker_walk

Figure 5: The learning performance comparison of SAC-AE and SAC-AE-HPO over 3 DeepMind Control environments. All the curves are averaged over 4 random seeds, where shadowed regions are standard deviations.

rather than raw observations. For all three environments, we select the hyper-parameters of HPO as $\beta^{\text{tr}} \in \{0.1, 0.15, 0.2\}, \beta^{\text{exp}} \in \{1.0, 1.5, 2.0\}, \epsilon = 0.1$. And the hyper-parameters of the baseline is set same as [47], except that the replay buffer capacity is 100000.

The screenshots and performance comparison are shown in Figure 5. We can see that adding HD into SAC-AE can consistently improve its learning speed and performance. Since the gradient of critic is only conditioned on the low-dimensional latent representation, the leapfrog steps do not have significant computation overhead.

5.2 Analysis

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

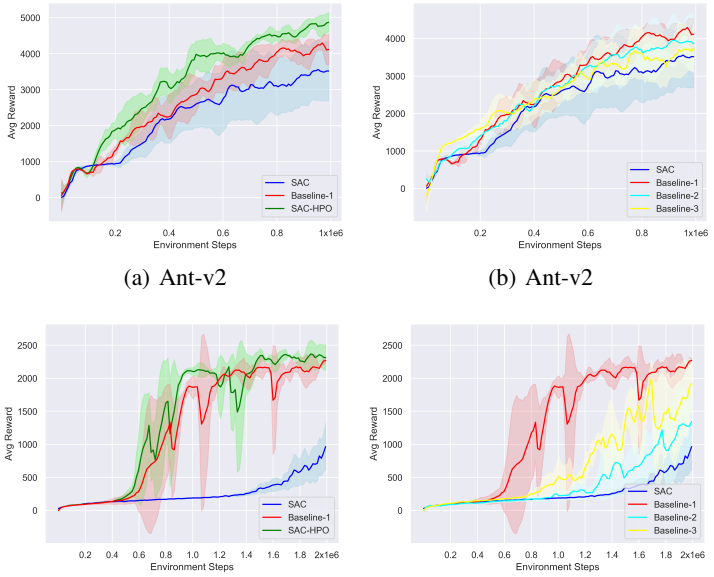
5.2.1 Ablation Study

In ablation study, we first verify the effect of the proposed leapfrog operator (16), compared with the conventional leapfrog in (12). Then we study the difference between the effects of HD in exploration and policy optimization, where the policy optimization refers to the training step of the policy network and neural networks T_h, σ_h in HD. Specifically, we introduce three baselines adopting different leapfrog operators in exploration and policy optimization, which are summarized in Table 6. Here "Gaussian Policy" means that the same policy network as SAC is directly used without evolving actions by simulating HD. "Conventional Leapfrog" refers to that actions are evolved by HD simulated by conventional leapfrog steps (12), and "Proposed Leapfrog" means that HD evolving actions is simulated by the proposed leapfrog steps (16). We cannot use HD simulated by the proposed leapfrog only in exploration, since the neural networks T_h, σ_h in (16) need to be trained.

The SAC-HPO and baselines are evaluated over Ant-v2 and HumanoidPyBulletEnv-v0, and learning curves are shown in Figure 6, where we use the same hyper-parameters as Section 5.1.1 and 5.1.2 for every task here. It can be seen that in Figure 6(a) and 6(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 one. In Figure 6(b) and 6(d), the baseline-1 outperforms both baseline-2 and baseline-3, showing the effects of HD in both exploration and policy optimization. Further, we

	Exploration	Policy Optimization
SAC-HPO	Proposed Leapfrog	Proposed Leapfrog
Baseline-1	Conventional Leapfrog	Conventional Leapfrog
Baseline-2	Conventional Leapfrog	Gaussian Policy
Baseline-3	Gaussian Policy	Proposed Leapfrog
SAC	Gaussian Policy	Gaussian Policy

Table 6: Exploration and Policy Optimization Strategies of Baselines



(a) Ant-v2 (b) Ant-v2

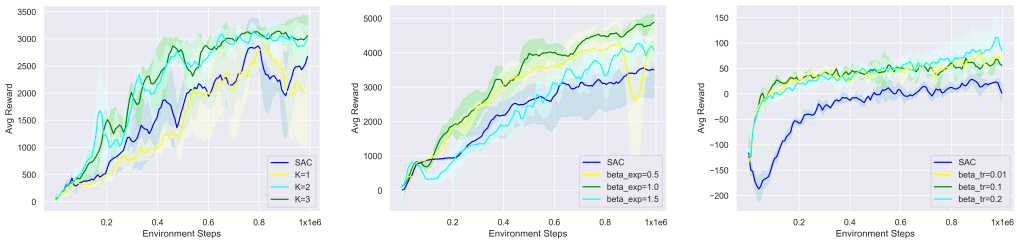
(c) HumanoidPyBulletEnv-v0 (d) HumanoidPyBulletEnv-v0

Figure 6: Performance Comparison in Ablation Study.

can see that the improvement of baseline-2 over SAC is higher that of baseline-3, meaning that using HD in exploration can have more performance improvement more than that in policy optimization.

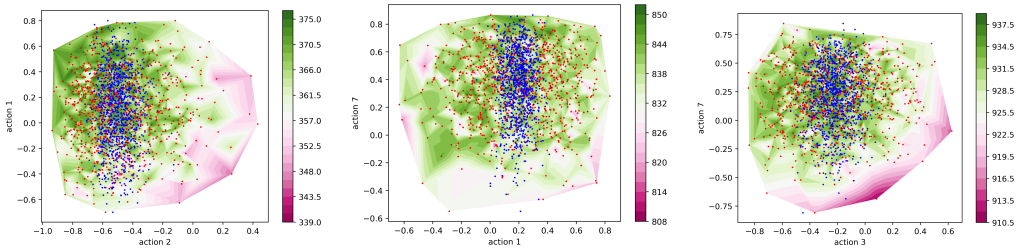
5.2.2 Sensitivity Analysis

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 Figure 7(a). 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 large K. The second parameter analyzed here is the variance of momentum vector ρ in exploration, i.e., $\beta_0^{\text{exp}-1} = 0.5, 1.0, 1.5$, shown in Figure 7(b). We



(a) Hopper-v2 (b) Ant-v2 (c) HumanoidPyBullet Flagrun Harder

Figure 7: Sensitivity analysis on K, β^{exp}, and β^{tr}.



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

Figure 8: The shape of policy distribution. 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.

can see that the performance is sensitive to the choice of β^{exp}, and not the highest choice leads to best performance. That is because although ρ with larger variance can boost the exploration, too much variance of ρ may make the sampled state-action pairs deviate too much from the optimal path. In Figure 7(c), we analyze three different the variance for momentum vector ρ in training, i.e., $\beta_0^{\text{tr}-1} = 0.01, 0.1, 0.2$. However, we can see that the performance is not sensitive to β_0^{tr}. According to more evaluations, larger β_0^{tr} can not lead to better performance, since when $\beta_0^{\text{tr}} = 1$, the performance degrades significantly.

5.2.3 Shape of Policy Distribution

We visualize the policy distributions at different environmental steps of Ant-v2 in Figure 8, where x and y axes represent two different dimensions of sampled actions in exploration. Specifically, the red dots represent 1000 actions sampled from the policy distribution produced by HD simulated by leapfrog steps. For comparison the blue dots represent 1000 actions sampled from the Gaussian distribution at the output of base policy network. The contour of Q value is shown as background for reference. From Figure 8 we can see that HD can evolve actions from the base policy to regions with higher Q values. Compared with actions sampled from the base policy, those evolved by HD can be non-Gaussian and have larger variance with much larger effective support. Besides, there are still some actions evolved to regions with similar or lower Q values, reaching a reasonable trade-off of exploration and exploitation. That is why the exploration can be boosted by HPO and the performance can be improved.

6 Conclusion

In this work, we propose to integrate policy optimization with HMC, evolving actions from the base policy network by Hamiltonian dynamics simulated leapfrog steps. In order to adapt to the changes of the target distribution defined by Q function, we propose a new leapfrog operator which generalizes HMC by neural networks. The proposed method can reduce the amortization gap in 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 the future, we are going to apply the proposed method in the real-world robotic tasks.

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] 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.

[5] 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.

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

[7] T Cover and J Thomas. Elements of information theory, 2ndwiley. *Hobo-ken, NJ*, 2006.

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

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

[10] Tom Erez, Yuval Tassa, and Emanuel Todorov. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4397–4404. IEEE, 2015.

[11] 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.

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

[19] 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.

[20] 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.

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

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

[23] Daniel Levy, Matt D Hoffman, and Jascha Sohl-Dickstein. Generalizing hamiltonian monte carlo with neural networks. In *International Conference on Learning Representations*, 2018.

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

[25] 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.

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

[27] 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.

[28] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

[40] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

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

[42] 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.

[43] 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.

[44] Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*, 2019.

[45] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

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

[47] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

[48] 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.