IMPROVING ACTOR-CRITIC REINFORCEMENT LEARNING VIA HAMILTONIAN MONTE CARLO METHOD

Duo Xu and Faramarz Fekri

Department of Electrical and Computer Engineering,
Georgia Institute of Technology,
Atlanta, GA 30332

ABSTRACT

The actor-critic RL is widely used in various robotic control tasks. However, by viewing the actor-critic RL from the perspective of variational inference (VI), 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. 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. 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.

Index Terms— Reinforcement learning, exploration, Hamiltonian Monte Carlo, varitaional inference

1. INTRODUCTION

In continuous control, actor-critic RL algorithms are widely used in solving practical problems. Specifically, most previous methods such as regularization based on past policies [1, 2] and maximum policy entropy [3, 4], essentially solve RL in the framework of variational inference (VI) [5], 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 [6, 5]. It means that most actor-critic RL algorithms, such as soft actor-critic (SAC) [7], 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 as encoder networks [8, 9, 10], the learned policy can be sub-optimal and far away from the target posterior, due to the insufficient expressivity of the policy network [11, 12]. This suboptimality is typically defined as the amortization gap [11], resulting into a gap in the RL objective.

In this work, by leveraging the advantages of both VI and HMC [13], 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, improving the efficiency of exploration. 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. Finally, the proposed leapfrog operator in Hamiltonian policy, which generalizes HMC via neural networks, can also increase the expressivity of the base policy network and adapt to the target distribution defined by the Q function which is changing during the learning process.

Using empirical experiments, we evaluated the proposed method across a variety of benchmark continuous control tasks such as OpenAI Gym using the MuJoCo simulator [14] and the realistic Bullet Roboschool tasks [15]. We show that the proposed method improves upon already high performing methods such as SAC [16] and SAC with normalizing flow policy [17], achieving both a better convergence rate and expected return. The details of hyper-parameter choices and ablation study are included in our full paper[1].

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) [7] and Hamiltonian Monte Carlo (HMC) [13] as building blocks of the

[1]`https://arxiv.org/abs/2103.12020`

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. where the initial state is drawn from the distribution $\rho(s_1)$. The objective of RL is to maximize the expected discounted return $\mathbb{E}_{p(\tau)}[\mathcal{R}(\tau)]$.

2.2. Reinforcement Learning via Variational Inference

Recently a surge of works have formulated reinforcement learning and control as probabilistic inference [18, 5]. 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 in the maximal reward is learned via probabilistic inference. Following the derivation in [5], at any time step t, the policy optimization objective can be written as,

$$\mathcal{J}(q, \theta) = \mathbb{E}_q[Q_q(s_t, a_t)] - \alpha D_{\text{KL}}(q(a_t|s_t, \mathcal{O}) \| \pi_\theta(a_t|s_t)) \tag{1}$$

Hence, with π_θ as action prior, policy optimization in the framework of VI [7, 5] is to find optimal q maximizing the objective $\mathcal{J}(q, \theta)$ in (1).

2.3. Soft Actor-Critic

Soft Actor-Critic (SAC) [7] 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 (1), the objective of SAC can be formulated as the state-action value function regularized with a maximum entropy,

$$\mathcal{L}(q) = \mathbb{E}_{s_t \sim \rho_q}\left[\mathbb{E}_{a_t \sim q} Q_q(s_t, a_t) - \alpha \log q(a_t|s_t)\right]. \tag{2}$$

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 (2) is $\bar{p}_\alpha(a|s) \propto \exp(Q(s, a)/\alpha)$ which is also the target policy distribution.

2.4. Hamiltonian Monte Carlo

Hamiltonian Monte Carlo (HMC) is a popular Markov chain Monte Carlo (MCMC) method for generating sequences of samples, which converge to being distributed according to the target distribution [13]. 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:

$$\frac{dx}{dt} = \frac{dH}{dx}, \qquad \frac{dv}{dt} = -\frac{dH}{dv} \tag{3}$$

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 $p(v) \propto \exp(-\beta v^T v/2)$ [13].

The HD described in (3) is typically simulated by the *leapfrog operator* [13], of which the single time step can be described as

$$v^{\frac{1}{2}} = v - \frac{\epsilon}{2}\partial_x U(x); \;\; x' = x + \epsilon v^{\frac{1}{2}}; \;\; v' = v^{\frac{1}{2}} - \frac{\epsilon}{2}\partial_{x'} U(x'); \tag{4}$$

which transforms (x, v) to (x', v'). We can see that transformations in (4) 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 (4) 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 improving policy optimization in recent years [7, 5]. Some papers used normalizing flow [16, 19, 17] to improve the expressivity of action distribution produced by the policy network. But none of them includes gradient information, and hence the exploration is still insufficient in some environments and performance degrades.

Another approach is to apply iterative amortization in policy optimization, which uses gradients of the Q function to iteratively update the parameters of the policy distribution [20]. However, especially when the estimation bias of Q functions is significant, if we directly use gradients to improve policy distributions without additional randomness, the policy search may get stuck at local optima and hence limit the exploration. Finally, another related work is Optimistic Actor Critic (OAC) [21] that employs the gradient of the value function to update actions in exploration. However, their updated policy distribution is still Gaussian without sufficient expressivity.

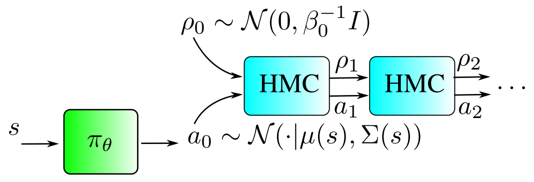

$\rho_0 \sim \mathcal{N}(0, \beta_0^{-1} I)$
HMC
ρ_1
a_1
HMC
ρ_2
a_2
$\cdots$
s
π_θ
$a_0 \sim \mathcal{N}(\cdot|\mu(s), \Sigma(s))$

Fig. 1. Diagram of Hamiltonian policy. The HMC box represents one leapfrog step described in (9).

4. METHODOLOGY

In this section, we first formulate the policy optimization method for Hamiltonian policy (HPO) 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. Additional considerations in implementation are also introduced.

4.1. Hamiltonian Policy Optimization

We call the proposed method as Hamiltonian policy optimization (HPO). The 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.

In our method, 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 (4) 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,

$$\begin{aligned} \rho_{k+1/2} &= \rho_k - \frac{\epsilon}{2} \odot \nabla U_\theta(s, a_k) \\ a_{k+1} &= a_k + \epsilon \odot \rho_{k+1/2} \\ \rho_{k+1} &= \rho_{k+1/2} - \frac{\epsilon}{2} \odot \nabla U_\theta(s, a_{k+1}) \end{aligned} \tag{5}$$

where ∇ is the differentiation taken with respect to a, $\epsilon \in \mathbb{R}$ is the learning step size, and $k = 0, \dots, K-1$. 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

Denote the k-th leapfrog step described above as $(a_k, \rho_k) := \Phi_{\theta,h}^k(a_{k-1}, \rho_{k-1})$. 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

$$\begin{aligned} q_{\theta,h}^K(a_K, \rho_K) &= q_{\theta,h}^0(a_0, \rho_0) \prod_{k=1}^{K} \left| \det \nabla \Phi_{\theta,h}^k(a_k, \rho_k) \right|^{-1} \\ &= \pi_\theta(a_0|s)\mathcal{N}(\rho_0|0, \beta_0^{-1} I) \end{aligned} \tag{6}$$

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 [5]. Since ELBO can be written as the difference between the log of target distribution and log of variational distribution [8], we can write the ELBO for HPO as below,

$$\mathcal{L}_{\text{ELBO}}(\theta, h; s) = \mathbb{E}_{(a_0,\rho_0)}[\log \bar{p}(a_K, \rho_K|s) - \log q_{\theta,h}^K(a_K, \rho_K)] \tag{7}$$

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

Finally, combining (6) and (7) 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

$$\mathcal{J}(\theta, h) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[Q_{\pi_\theta}(s, a_K) - \alpha \log \pi_\theta(a_0|s) - \frac{\alpha\beta_0}{2}\rho_K^T \rho_K\right] \tag{8}$$

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 [7].

4.2. Proposed Leapfrog Operator

Since HMC with conventional leapfrog (5) converges and mixes slowly, some past works proposed to use neural networks to generalize HMC [5, 22]. However, since Q networks are changing in RL, the techniques proposed in [5, 22] cannot be used here. Based on our empirical study, the direction variable, binary mask and exp operation therein [5, 22] 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 (5) and design a new leapfrog operator.

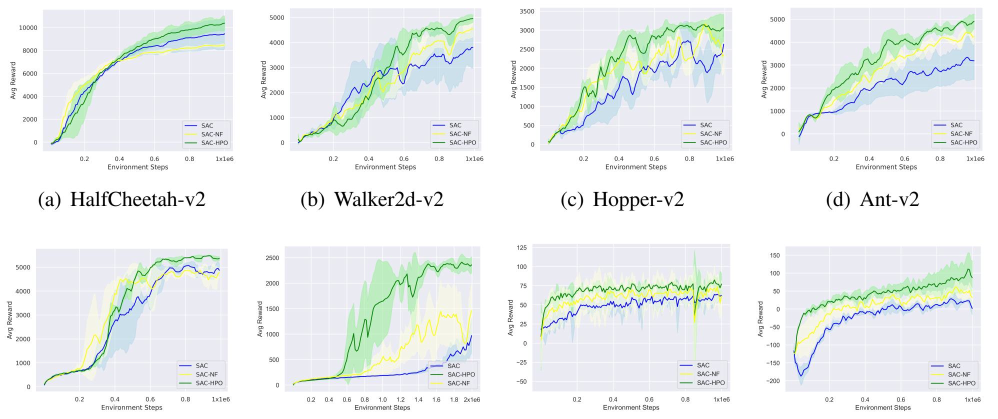
The inputs of T_h and σ_h include normalized gradients, action and state, where the state is optional and can be ignored in some environments. Therefore, the proposed leapfrog operation, transforming from (a_k, ρ_k) to (a_{k+1}, ρ_{k+1}), can be described as

$$\begin{aligned} \rho_{k+1/2} &= \rho_k - \frac{\epsilon}{2} \odot (\sigma_h(s, a_k, g) \odot g \\ &\quad + (1 - \sigma_h(s, a_k, g)) \odot T_h(s, a_k, g)) \\ \rho_{k+1} &= \rho_{k+1/2} - \frac{\epsilon}{2} \odot (\sigma_h(s, a_{k+1}, g') \odot g' \\ &\quad + (1 - \sigma_h(s, a_{k+1}, g')) \odot T_h(s, a_{k+1}, g')) \end{aligned} \tag{9}$$

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 $a_{k+1} = a_k + \epsilon \odot \rho_{k+1/2}$. Obviously the proposed leapfrog operator (9) still keeps the properties of reversibility and unit Jacobian, so that the distribution of (a_K, ρ_K) in (6) is still tractable.

4.3. Implementation and Algorithms

In implementation, we only use one hidden layer for the base policy network π_θ, so the number of parameters of our model



(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

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

is much smaller than the model in previous papers [16, 20] 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 (8) is used. It is termed as SAC-HPO. The process of producing actions from the Hamiltonian policy is summarized as below and shown in Figure 1.

5. EXPERIMENT

The environments in our experiments are diverse, ranging from OpenAI Gym MuJoCo [14] to the realistic Roboschool PyBullet suit [15, 23]. First, SAC-HPO is compared with the primitive SAC [7] and SAC with normalizing flow policy (SAC-NF) [17]. The details of hyper-parameter choices and ablation study is included in our full paper on arXiv[1].

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. And SAC-NF is a widely used method to improve the expressivity of policy distribution. We use their official implementations [7] for SAC. And we try our best to implement SAC-NF according to [17]. 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.

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 [7], 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 [24] 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-HPO, actions are evolved by HD simulated by leapfrog operations (9) 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 32 or 64 hidden units and ELU activation. The variances of the momentum vector (β_0) should be different for training and exploration, denoted as $\beta_0^{\text{tr}} \in [0.01, 0.2]$ and $\beta_0^{\text{exp}} \in [0.5, 1.5]$ 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.

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.

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 the Q function, we propose a new leapfrog operator which generalizes HMC via 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.

7. ACKNOWLEDGEMENT

This work was supported by NSF Grant CNS-1837369.

8. REFERENCES

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

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

[3] 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*, 2016, pp. 1928–1937.

[4] 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*, 2016, pp. 202–211.

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

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

[7] 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*, 2018, pp. 1861–1870.

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

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

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

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

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

[13] Radford M Neal et al., "Mcmc using hamiltonian dynamics," *Handbook of markov chain monte carlo*, vol. 2, no. 11, pp. 2, 2011.

[14] 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*. IEEE, 2012, pp. 5026–5033.

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

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

[17] 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*. PMLR, 2020, pp. 430–444.

[18] Matthew Botvinick and Marc Toussaint, "Planning as inference," *Trends in cognitive sciences*, vol. 16, no. 10, pp. 485–488, 2012.

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

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

[21] Kamil Ciosek, Quan Vuong, Robert Loftin, and Katja Hofmann, "Better exploration with optimistic actor critic," in *Advances in Neural Information Processing Systems*, 2019, vol. 32, pp. 1787–1798.

[22] Lingge Li, Andrew Holbrook, Babak Shahbaba, and Pierre Baldi, "Neural network gradient hamiltonian monte carlo," *Computational statistics*, vol. 34, no. 1, pp. 281–299, 2019.

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

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