Leveraging exploration in off-policy algorithms via normalizing flows

Bogdan Mazoure*[1,2], Thang Doan*[1,2], Audrey Durand[2,3], Joelle Pineau[1,2,4], R Devon Hjelm[2,5,6]

[1]McGill University [2]Mila – Quebec AI Institute [3]Université Laval
[4]Facebook AI Research [5]Microsoft Research Montreal
[6]Université de Montréal

Abstract: The ability to discover approximately optimal policies in domains with sparse rewards is crucial to applying reinforcement learning (RL) in many real-world scenarios. Approaches such as neural density models and continuous exploration (e.g., Go-Explore) have been proposed to maintain the high exploration rate necessary to find high performing and generalizable policies. Soft actor-critic (SAC) is another method for improving exploration that aims to combine efficient learning via off-policy updates, while maximizing the policy entropy. In this work, we extend SAC to a richer class of probability distributions (e.g., multimodal) through normalizing flows (NF) and show that this significantly improves performance by accelerating discovery of good policies while using much smaller policy representations. Our approach, which we call SAC-NF, is a simple, efficient, easy-to-implement modification and improvement to SAC on continuous control baselines such as MuJoCo and PyBullet Roboschool domains. Finally, SAC-NF does this while being significantly parameter efficient, using as few as 5.5% the parameters for an equivalent SAC model.

1 Introduction

Reinforcement learning (RL) provides a principled framework for solving continuous control problems, yet current RL algorithms often do not explore well enough to solve high-dimensional robotics tasks [1]. Environments with a large number of continuous control factors, such as those that involve combinations of leg movement, arm movement, posture, etc, have many local minima [2]. For example, it is possible to achieve forward momentum in humanoid environments [3] with a variety of suboptimal policies over those factors (e.g. arms lean forward or backward, not synchronized with legs), in a way that will fail readily as the environmental variables change (such as environments designed to purposely destabilize the agent, e.g., Coumans and Bai 2016). Success in these environments requires a complex coordination of the control factors, and to learn this, it is necessary to have an exploration strategy that avoids converging too early on suboptimal local minima [5].

Soft Actor-Critic (SAC) [6] is a state-of-the-art exploration-based algorithm that adds a maximum entropy bonus term [7] to a differentiable policy objective specified by a soft critic function. As an off-policy algorithm, SAC enjoys sample efficiency – a desirable property in robotics, where real world experiments might be costly to perform. However, SAC is limited to modeling policies that have closed-form entropy (e.g., unimodal Gaussian policies), which we posit hurts exploration [8]. The main contribution of this work is to extend SAC to a richer class of multimodal exploration policies, by transforming the actions during exploration via a sequence of invertible mapping known as *normalizing flows* (NF) [9]. Our approach, which we call SAC-NF, is a simple and easy-to-implement extension to the original SAC algorithm that gives the agent access to a more expressive multimodal policy and that achieves much better performance on continuous control tasks.

We show empirically that this simple extension significantly improves upon the already high exploration rate of SAC and achieves better convergence properties as well as better performance on both

*These authors contributed equally.

3rd Conference on Robot Learning (CoRL 2019), Osaka, Japan.

sparse and deceptive environments. Next, the class of policies that we propose requires significantly less parameters than its baseline counterpart, while also improving on the baseline results. Finally, we assess the performance of both SAC and SAC-NF across a variety of benchmark continuous control tasks from OpenAI Gym using the MuJoCo simulator [10] and the realistic Bullet Roboschool tasks [4].

2 Related Work

Off-policy RL Off-policy strategies in RL collect samples under some behaviour policy and use those samples to train a target policy. Off-policy algorithms are known to train faster than their on-policy counterparts, but at the cost of higher variance and instability [11]. Among this family, actor critic (AC) strategies have shown great success for solving continuous control tasks. In between value-based and policy-based approaches, an AC algorithm trains an *actor* (policy-based) using guidance from a *critic* (value-based). Two major AC algorithms, SAC [6] and TD3 [12], have shown a large performance improvement over previous off-policy algorithms such as DDPG [11] or A3C [13]. TD3 achieved this by maintaining a second critic network to alleviate the overestimation bias, while SAC enforced more exploration by adding an entropy regularization term to the loss function.

Density estimation for better exploration Using powerful density estimators to model state-action values with the aim to improve exploration generalization has been a long-standing practice in RL. For instance, [14] use dropout approximation [15] within a Bayesian network and show improvement on stability and performance of policy gradient methods. [16] rather rely on an ensemble of neural networks to estimate the uncertainty in the prediction of the value function, allowing to reduce learning times while improving performance. Finally, [17] consider generative adversarial networks [18] to model the distribution of random state-value functions. The current work considers a different approach based on normalizing flows for density estimation.

Normalizing flows Flow-based generative models have proven to be powerful density approximators [9]. The idea is to relate an initial noise density distribution to a posterior distribution using a sequence of invertible transformations, parametrized by a neural network and having desirable properties. For example, invertible autoregressive flows (IAF) are characterized by a simple-to-compute Jacobian [19]. In their original formulation, IAF layers allow learning location-scale invariant (i.e. affine) transformations of a simple initial noise density.

Normalizing flows have been used previously in the on-policy RL setting where IAF extends a base policy found by TRPO [20]. In this work, we tackle the off-policy learning setting, and we focus on planar and radial flows, which are known to provide a good trade-off between function expressivity and time complexity [9]. Our work explores a similar space as hierarchical-SAC (HSAC) [21], which also modifies the policy of SAC to improve expressiveness. However, HSAC has a significantly more complex model, as it uses real NVP [22] along with a hierarchical policy, optimizing a different reward function at each hidden layer. This represents a stronger departure from the original SAC model and algorithm. We show that simply training all NF layers jointly on a single reward function without any additional conditioning produces significant improvement over SAC and HSAC with a model and training procedure that is reasonably close to SAC.

3 Background

In this section, we review the formal setting of RL in a Markov decision process (MDP), Soft Actor-Critic (SAC) [6], and the general framework of normalizing flows (NFs) [9], the latter of which will be used to improve exploration in Section 4.

3.1 Markov Decision Process

MDPs [23, 24] are useful for modelling sequential decision-making problems. A discrete-time finite-horizon MDP is described by a state space $\mathcal{S}$, an action space $\mathcal{A}$[2], a transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^+$, and a reward function $r : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. On each round t, an agent interacting

[2]The state $\mathcal{S}$ and the action $\mathcal{A}$ spaces can be either discrete or continuous

with this MDP observes the current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, and observes a reward $r_t = r(s_t, a_t) \in \mathbb{R}$ upon transitioning to a new state $s_{t+1} \sim \mathcal{P}(s_t, a_t)$. Let $\gamma \in [0, 1)$ be a discount factor; the goal of an agent evolving in a discounted MDP is to learn a policy $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0, 1]$, such that taking action $a_t \sim \pi(\cdot|s_t)$ would maximize the expected sum of discounted returns,

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right].$$

The corresponding state-action value function can be written as the expected discounted rewards from taking action a in state s, that is,

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\pi}\left[\sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i) | s_t = s, a_t = a\right].$$

If $\mathcal{S}$ or $\mathcal{A}$ are vector spaces, action and space vectors are respectively denoted by $\mathbf{a}$ and $\mathbf{s}$.

3.2 Soft Actor-Critic

SAC [6] is an off-policy algorithm which updates the policy using gradient descent, minimizing the KL divergence between the policy and the Boltzmann distribution using the critic (i.e., Q-function) as a negative energy function,

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} \mathrm{D}_{\mathrm{KL}}\left(\pi'(.|\mathbf{s}_t) \middle\| \frac{\exp\{\frac{1}{\alpha} Q^{\pi_{\text{old}}}(\mathbf{s}_t, .)\}}{Z^{\pi_{\text{old}}}(\mathbf{s}_t)}\right), \tag{1}$$

where $\alpha \in (0, 1)$ controls the temperature, $Q^{\pi_{\text{old}}}$ is the Q-function under the old policy, and the partition function $Z^{\pi_{\text{old}}}(\mathbf{s}_t)$ can be ignored [6]. The KL divergence above is tractable and differentiable as the policies are assumed to be composed of diagonal Gaussians in the classical SAC formulation. It can be seen easily that SAC follows a maximum entropy objective [7], as optimizing w.r.t. Equation 1 is equivalent to maximizing the state-action value function regularized with a maximum entropy term,

$$\begin{aligned}
\mathcal{L}_{\pi} &= \mathop{\mathbb{E}}_{s_t \sim \rho_{\pi}}\left[\mathop{\mathbb{E}}_{a_t \sim \pi}[Q^{\pi}(s_t, a_t)] + \alpha H(\pi(.|s_t))\right] \\
&= \mathop{\mathbb{E}}_{\mathbf{s}_t \sim \rho_{\pi}}[V(\mathbf{s}_t)]; \\
&\text{where} \quad V(\mathbf{s}_t) := \mathop{\mathbb{E}}_{a_t \sim \pi}[Q^{\pi}(s_t, a_t) - \alpha \log \pi(a_t|s_t)]
\end{aligned}$$

is the state-action value function, $H(\pi(.|s_t))$ is the entropy of the policy, α is now the importance given to the entropy regularizer. If $\pi'(\cdot|\mathbf{s}_t) \sim \mathcal{N}(\mu, \mathrm{diag}(\sigma_1^2, .., \sigma_d^2))$, then $\max H(\pi'(\cdot|\mathbf{s}_t)) = \max \log\det(\mathrm{diag}(\sigma_1^2, .., \sigma_d^2)) = \max \sum_{i=1}^{d} \sigma_i^2$, which is unbounded without additional constraints. This prevents collapse to degenerate policies centered at the point with highest rewards and keeps exploration active.

SAC models the value function and the critic using neural networks, V_{ν} and Q_{ω}, and models a Gaussian policy π_{θ} with mean and variance determined by the output of neural networks with parameters θ. The losses for V_{ν}, Q_{ω}, and π_{θ} are computed using a replay buffer $\mathcal{D}$,

$$\mathcal{L}_Q = \mathop{\mathbb{E}}_{(\mathbf{s}_t, \mathbf{a}_t) \sim \mathcal{D}}\left[\frac{1}{2}\left\{Q_{\omega}(\mathbf{s}_t, \mathbf{a}_t) - Q_{\nu}^{\dagger}\right\}^2\right], \tag{2}$$

$$\mathcal{L}_V = \mathop{\mathbb{E}}_{\mathbf{s}_t \sim \mathcal{D}}\left[\frac{1}{2}\left\{V_{\nu}(\mathbf{s}_t) - \mathop{\mathbb{E}}_{\mathbf{a}_t \sim \pi}[Q_{\omega}(\mathbf{s}_t, \mathbf{a}_t) - \alpha \log \pi(\mathbf{a}_t|\mathbf{s}_t)]\right\}^2\right], \tag{3}$$

$$\mathcal{L}_{\pi} = \mathop{\mathbb{E}}_{\mathbf{s}_t \sim \mathcal{D}}\left[\mathop{\mathbb{E}}_{\mathbf{a}_t \sim \pi}[\alpha \log \pi_{\theta,\phi}(\mathbf{a}_t|\mathbf{s}_t) - Q_{\omega}(\mathbf{s}_t, \mathbf{a}_t)]\right], \tag{4}$$

where $Q_{\nu}^{\dagger} = (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma\, \mathbb{E}_{\mathbf{s}_{t+1} \sim \mathcal{D}}[V_{\nu}(\mathbf{s}_{t+1})])$.

In practice, the gradients of the above losses are approximated using Monte-Carlo. As an off-policy algorithm, SAC enjoys the advantages of having lower sample complexity than on-policy algorithms [25], yet it outperforms other off-policy alternatives [11] due to its max entropy term.

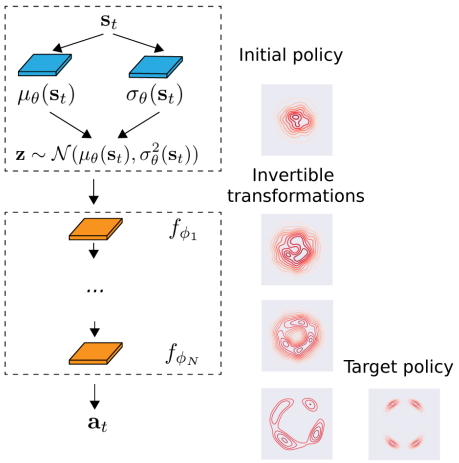

Figure 1: Schematic representation of a SAC-based normalizing flow policy, which takes as input noise and state information. The initial noise density is then fed through a sequence of invertible transformations to match the target Boltzmann Q-function.

3.3 Normalizing Flows

NFs [9] are a class of iterative methods for transforming probability distributions introduced as a way to improve the approximate posterior in amortized inference algorithms [26, 27]. More generally, they provide a framework for extending the change of variable theorem for density functions to a sequence of d-dimensional real random variables $\{\mathbf{z}_i\}_{i=0}^N$. The initial random variable $\mathbf{z}_0$ has density function q_0 and is linked to the final output of the flow $\mathbf{z}_N$ through a sequence of invertible, smooth mappings $\{f_i\}_{i=1}^N$ called *normalizing flows* of length N. A number of different invertible function families can be specified through the choice of neural network parameterization and regularization [9, 19, 28]. One good choice [19] is the family of radial contractions around a point $\mathbf{z}_0 \in \mathbb{R}^d$ defined as [9],

$$f(\mathbf{z}) = \mathbf{z} + \frac{\beta}{\alpha + ||\mathbf{z} - \mathbf{z}_0||_2}(\mathbf{z} - \mathbf{z}_0), \tag{5}$$

which are highly expressive (i.e. represent a wide set of distributions) and yet very light (parameter-wise), in addition to enjoying a closed-form determinant. This family allows approximating the target posterior through a sequence of concentric expansions of arbitrary width and centered around a learnable point $\mathbf{z}_0$. In order to guarantee that the flow is invertible, it is sufficient to pick $\beta \geq -\alpha$.

4 Augmenting SAC with Normalizing Flows

We now propose a flow-based formulation of the off-policy maximum entropy RL objective (Eq. 2) and argue that SAC is a special case of the resulting approach, called SAC-NF.

4.1 Exploration through normalizing flows

Figure 1 outlines the architecture of a normalizing flow policy based on SAC. Let ε be an initial noise sample, $h_\theta(\varepsilon, \mathbf{s}_t)$ a state-noise embedding, and $\{f_\phi\}_{i=1}^N$ a normalizing flow of length N parameterized by $\phi = \{\phi_i\}_{i=1}^N$. Sampling from the policy $\pi_{\phi,\theta}(\mathbf{a}_t|\mathbf{s}_t)$ can be described by the following set of equations:

$$\begin{aligned} \mathbf{a}_t &= f_{\phi_N} \circ f_{\phi_{N-1}} \circ ... \circ f_{\phi_1}(\mathbf{z}), \\ \mathbf{z} &= h^j_\theta(\varepsilon, \mathbf{s}_t), \qquad\qquad j = 1, 2 \\ \varepsilon &\sim \mathcal{N}(0, \mathbf{I}), \end{aligned} \tag{6}$$

where the state-noise embedding $h^j_\theta(\varepsilon, \mathbf{s}_t)$ models samples from a base Gaussian distribution, with state-dependent means, $\mu_\theta(\mathbf{s_t})$. The index j denotes a hyperparameter, choosing either state-

dependent ($\mathbf{L}_\theta(\mathbf{s_t})$ for $j=1$) or state-independent ($\mathbf{L}_\theta$ for $j=2$) diagonal scale matrices,

$$h_\theta^j(\varepsilon_0, \mathbf{s}_t) = \begin{cases} \varepsilon \mathbf{L}_\theta(\mathbf{s}_t) + \mu_\theta(\mathbf{s}_t), & j=1 \text{ (conditional)} \\ \varepsilon \mathbf{L}_\theta + \mu_\theta(\mathbf{s}_t), & j=2 \text{ (average);} \end{cases}$$
$$\text{where} \quad \mathbf{L}(\mathbf{s_t}) = \text{diag}\{(\sigma_1(\mathbf{s}_t), .., \sigma_{|\mathcal{A}|}(\mathbf{s}_t))\}$$
$$\mathbf{L} = \text{diag}\{(\sigma_1, .., \sigma_{|\mathcal{A}|})\} \tag{7}$$

We chose j according to experiments in the supplementary. Both functions allow to sample either from a heteroscedastic or homoscedastic Gaussian distribution, following the reparametrization trick in variational Bayes [29], and we explore these choices in more detail in the supplementary material. Precisely, $\mu(\mathbf{s}_t) : \mathcal{S} \rightarrow \mathbb{R}^d$ is a state embedding function and $\mathbf{L}(\mathbf{s}_t), \mathbf{L}$ is a scale parameter. For flows of the form $f_\phi(\mathbf{z}) = \mathbf{z} + g_\phi(\mathbf{z})$, we can asymptotically recover the original base policy through heavy regularization,

$$\lim_{||\phi_1||,..,||\phi_N|| \rightarrow 0} \pi(\cdot|\mathbf{s}_t) \overset{d}{=} \mathcal{N}\big(\mu(\mathbf{s}_t), \mathbf{L}(\mathbf{s}_t)\mathbf{L}(\mathbf{s}_t)^\top\big), \tag{8}$$

for all states $\mathbf{s}_t \in \mathcal{S}$. By analogy with the SAC updates, SAC-NF minimizes the KL divergence between the Boltzmann Q and the feasible set of normalizing flow-based policies. The KL term is once again tractable and the policy density now depends on the sum of log Jacobians of the flows:

$$\log \pi(\mathbf{a}_t, \mathbf{s}_t) = \log q_0(\mathbf{a}_0) - \log |\det \mathbf{L}| - \sum_{i=1}^{N} \log \Big| \det \frac{\partial f_i(\mathbf{a}_{i-1})}{\partial \mathbf{a}_{i-1}} \Big|. \tag{9}$$

Algorithm 1 outlines the proposed method: the major distinction from the original SAC is the additional gradient step on the normalizing flows layers while fixing the SAC weights θ.

Algorithm 1 SAC-NF

Input: Mini-batch size m; replay buffer $\mathcal{D}$; number of epoch T; learning rates $\alpha_\theta, \alpha_\phi, \alpha_\nu, \alpha_\omega$
Initialize value function network $V_\nu(\mathbf{s})$
Initialize critic network $Q_\omega(\mathbf{s}, \mathbf{a})$
Initialize policy network with weights $\pi_{\phi,\theta}(\mathbf{s})$
for epoch $= 1, ..., T$ **do**
  $\mathbf{s} \leftarrow \mathbf{s}_0$
  for t=0... **do**
    $\mathbf{a}_t \sim \pi(.|\mathbf{s}_t)$
    Observe $\mathbf{s}_{t+1} \sim P(\cdot|\mathbf{s}_t, \mathbf{a}_t)$ and get reward r_t
    Store transition ($\mathbf{s}_t$,$\mathbf{a}_t$,r_t,$\mathbf{s}_{t+1}$) in $\mathcal{D}$
    for each learning step **do**
      {Update networks with m MC samples each}
      $\nu \leftarrow \nu - \alpha_\nu \nabla_\nu \hat{\mathcal{L}}_V$ {Update value function}
      $\omega \leftarrow \omega - \alpha_\omega \nabla_\omega \hat{\mathcal{L}}_Q$ {Update critic}
      $\theta \leftarrow \theta - \alpha_\theta \nabla_\theta \hat{\mathcal{L}}_\pi$ {Update base policy}
      $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi \hat{\mathcal{L}}_\pi$ {Update NF layers}
    end for
  end for
end for

5 Experiments

This section addresses three major points: (1) it highlights the beneficial impact of NF deceptive rewards domains through a navigation task, (2) compares the proposed SAC-NF approach against SAC on a set of continuous benchmark control tasks from MuJoCo, Rllab and the more realistic Roboschool PyBullet suite [4, 10, 30] and finally (3) investigates the shape and number of modes that can be learned by radial NF policies. For all experiments[3], the entropy rate α is constant and

[3] We trained all policies on Intel Gold 6148 Skylake @ 2.4 GHz processors.

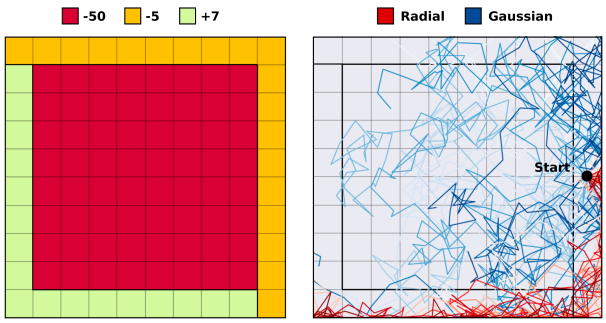

Figure 2: A deceptive reward environment where the high reward region is hidden behind a negative reward region (left subfigure). The right subfigure shows trajectories of Gaussian and radial agents. While the SAC policy is exploring vaguely the yellow region and falls into the pit, the SAC-NF policy manages to find the optimal region by going from the yellow to green zone.

can be found in Supp. Table 3. For the SAC baseline, we used hyperparameters reported in [6]. A thorough study of multi-modality and non-Gaussianity of SAC-NF policies on MuJoCo is shown in the Appendix.

5.1 Robustness to confounding rewards

We first demonstrate that normalizing flow policies are able to find better solutions than a Gaussian policy for SAC in an environment with deceptive rewards. We consider an environment composed of three reward areas: a locally optimal strip around the initial state, a global optimum on the opposing end of the room, separated by a pit of highly negative reward. The agent starts at the position $\mathbf{s}_0 = (4.5, 0)$ and must navigate into the high rewards area without falling into the pit. On each time t, the agent receives the reward r_t associated to its current location $\mathbf{s}_t$. The experimental setup can be found in Supplementary Material.

Figure 2 displays the trajectories visited by both agents. This highlights the biggest weakness of vanilla SAC policies: the agent is unable to simultaneously reach the region of high rewards while avoiding the center of the room. In this case, lowering the entropy threshold will lead to the conservative behaviour of staying in the yellow zone; increasing the entropy leads the agent to die without reaching the goal. Breaking the symmetry of the policy by adding (in this case three) radial flows allows the agent to successfully reach the target area by walking along the safe path surrounding the room.

In the case of steep reward functions, where low rewards border on high rewards, symmetric distributions force the agent to explore into all possible directions. This leads the agent to sometimes attain the high reward region, but, more dangerously, falling into low reward areas with non-zero probability at training time.

5.2 Continuous control tasks

MuJoCo locomotion benchmarks

Next, we compare our SAC-NF method against the SAC baseline on six continuous control tasks from the MuJoCo suite (see Figure 3) and one sparse reward MuJoCo task[4]. All results curves show evaluation time performance which, in the case of SAC and SAC-NF, is equivalent to setting the noise to 0. Evaluation happens every 10,000 steps, and values reported in the tables are not smoothed. The values reported in the plots are smoothed with a window size of 7, equivalent to smoothing every 70,000 steps to improve readability.

The SAC-NF agent consists of one feed-forward hidden layer of 256 units acting as state embedding, which is then followed by a normalizing flow of length $N \in \{3, 4, 5\}$. Details of the model can be found in Supp. Table 3. For the SAC baseline, two hidden layers of 256 units are used. The critic and value function architectures are the same as in [6]. All networks are trained with Adam optimizer [31]

[4]The sparse Humanoid task can be found here: https://github.com/bmazoure/sparseMuJoCo

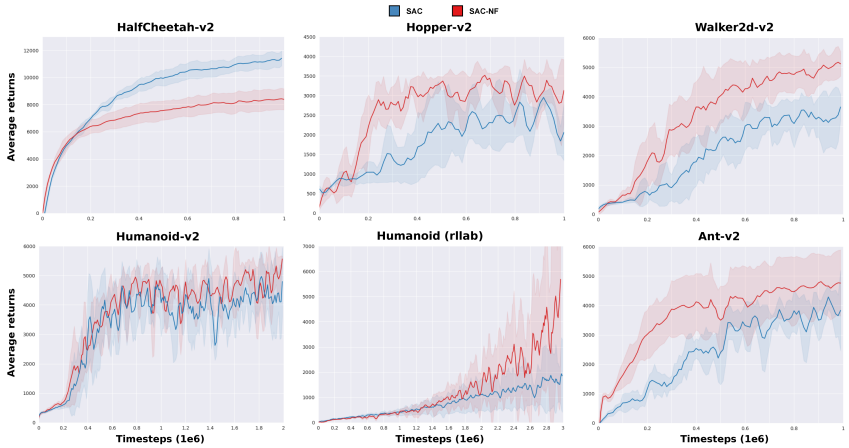

Figure 3: Performance of SAC-NF against SAC across 6 MuJoCo tasks (higher is better). Curves are averaged over 5 random seeds and then smoothed using Savitzky-Golay filtering with window size 7.

with a learning rate of 3E^{-4}. Every $10,000$ environment steps, we evaluate our policy 10 times and report the average. The best observed reward for each method can be found in Table 1.

Figure 3 displays the performance of both SAC and SAC-NF. We observe that SAC-NF shows faster convergence, which translates into better sample efficiency, compared to the baseline. SAC-NF takes advantage of the expressivity of normalizing flows to allow for better exploration and thus discover new policies. In particular, we notice that SAC-NF performs well on three challenging tasks: `Humanoid-v2`, `Humanoid (rllab)` and `Ant-v2`. High rewards collected by SAC-NF agents suggest that Gaussian policies that are widely used for continuous control [25, 32] might not be best suited for certain domains (see Supplementary Material for a shape analysis of SAC-NF policies on `Ant-v2`).

Table 1 not only shows better performance from SAC-NF in most of the environments, but shows the ratio in the number of parameters in the policy architecture between SAC-NF and vanilla SAC. For instance, on `Hopper-v2`, we could reduce by up to 95% the number of parameters ($70,406$ parameters for SAC baseline versus $3,861$ for SAC-NF) and by 41% the number of parameters in `Humanoid-v2`, while performing at least as well as the baseline. For space constraints, we also reported results from TD [12] in the Supplementary material.

	SAC	SAC-NF	$\frac{\#\{\text{SAC-NF}\}}{\#\{\text{SAC}\}}$
Ant-v2	$4,372 \pm 900$	$\mathbf{4912 \pm 954}$	≈ 0.31
HalfCheetah-v2	$\mathbf{11410 \pm 537}$	8429 ± 818	≈ 0.09
Hopper-v2	3095 ± 730	$\mathbf{3538 \pm 108}$	≈ 0.06
Humanoid-v2	$\mathbf{5505 \pm 116}$	$\mathbf{5506 \pm 147}$	≈ 0.6
Humanoid (rllab)	2079 ± 1432	$\mathbf{5531 \pm 4435}$	≈ 0.4
Walker2d-v2	3813 ± 374	$\mathbf{5196 \pm 527}$	≈ 0.09
SparseHumanoid-v2	88 ± 159	$\mathbf{547 \pm 268}$	≈ 0.6

Table 1: Maximal average return $\pm$ one standard deviation across 5 random seeds for SAC and SAC-NF. Last column shows the ratio in number of policy parameters between the two methods. Learning curve for SparseHumanoid-v2 can be found in the Appendix.

SAC could be run with fewer parameters for better comparison with the small architecture of SAC-NF. We also run SAC with a reduced number of hidden units (64 and 128 for the policy network only). In general, running SAC with fewer parameters achieves worse results: best results with either 64 or 128 units are as follows (5 seeds, after 1M steps, 3M for Rllab): 7300 versus 11,410 for the 256 units architecture (HalfCheetah), 4400 versus 5505 (Humanoid), 2900 versus 2079 (Humanoid-rllab), 3800 versus 3813 (Ant).

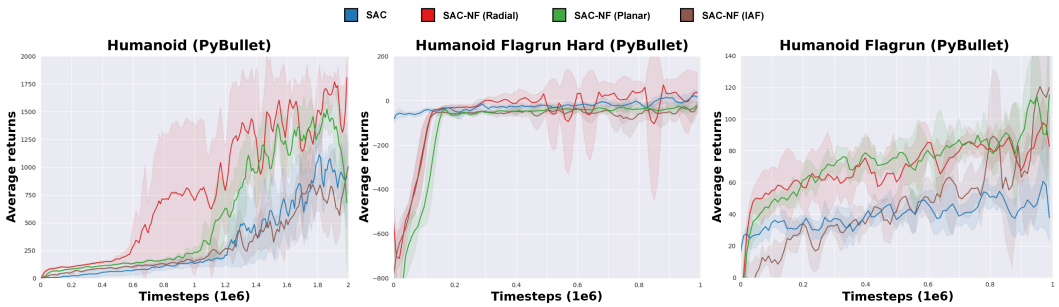

Figure 4: Performance of SAC-NF with IAF, planar, and radial flows compared against SAC (Gaussian policy) accross 3 Roboschool PyBullet domains (higher is better). Curves are averaged over 5 random seeds and then smoothed using Savitzky-Golay filtering with window size 7. Radial flows perform consistently well across the 3 robotics environments.

	SAC	Radial	Planar	IAF
Humanoid (PyBullet)	1263 ± 290	$\mathbf{1755 \pm 131}$	1561 ± 238	1130 ± 206
Humanoid Flagrun Hard (PyBullet)	31 ± 33	$\mathbf{46 \pm 61}$	-10 ± 16	-2 ± 33
Humanoid Flagrun (PyBullet)	67 ± 41	106 ± 23	128 ± 42	$\mathbf{152 \pm 173}$

Table 2: Maximal average return obtained on three Roboschool PyBullet environments by Gaussian, IAF, planar, and radial policies $\pm$ one standard deviation across 5 random seeds.

Realistic continuous control with Bullet Roboschool

To assess the behaviour of SAC-NF in realistic environments, we tested our algorithm on the PyBullet Gym implementation[5] of Roboschool tasks [4]. The Bullet library is among the most realistic collision detection and multi-physics simulation engines available up to now, and is widely used for sim-to-real transfer tasks.

To assess the impact of flow family on performance, we compared three types of normalizing flows: radial, planar, and IAF[6]. Figure 4 displays the performance of both SAC and SAC-NF for all three flows families obtained using the same setup as for MuJoCo.

The best observed reward for each method can be found in Supp. Table 2. SAC-NF with radial flows consistently ranks better (performance and parameter-wise, see Supp. Table 4) than the Gaussian policy and, in some domains, better than planar and IAF flows.

6 Conclusion

We proposed an algorithm which combines soft actor-critic updates together with a sequence of normalizing flows of arbitrary length. The high expressivity of the later allows to (1) quickly discover richer policies (2) compress the cumbersome Gaussian policy into a lighter network and (3) better avoid local optima. Our proposed algorithm leverages connections between maximum entropy reinforcement learning and the evidence lower bound used to optimize variational approximations. Finally, we validated the model on six MuJoCo tasks, three Bullet Roboschool tasks and one sparse domains, on which SAC-NF showed significant improvement against the SAC baseline in terms of convergence rate as well as performance. Interesting challenges for future work include studying the generalization and theoretical properties of normalizing flow SAC policies to better transfer from rich simulators to real robots.

Acknowledgements

We want to thank Compute Canada/Calcul Québec and Mila – Quebec AI Institute for providing computational resources. We also thank Chin-Wei Huang for insightful discussions.

[5]The environment can be found at: https://github.com/benelot/pybullet-gym

[6]We adapted the more general implementation from: https://github.com/CW-Huang/naf

References

[1] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*, 2019.

[2] J. Lehman. *Evolution through the search for novelty*. PhD thesis, University of Central Florida, Orlando, Florida, 2012.

[3] J. Lehman and K. O. Stanley. Novelty search and the problem with objectives. In *Genetic programming theory and practice IX*, pages 37–56. Springer, 2011.

[4] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. *GitHub repository*, 2016.

[5] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *Advances in Neural Information Processing Systems*, 2017.

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International Conference on Machine Learning*, 2018.

[7] R. J. Williams and J. Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

[8] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

[9] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *International Conference on Machine Learning*, 2015.

[10] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[11] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations*, 2016.

[12] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *International Conference on Machine Learning*, 2018.

[13] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

[14] P. Henderson, T. Doan, R. Islam, and D. Meger. Bayesian policy gradients via alpha divergence dropout inference. *NIPS Bayesian Deep Learning Workshop*, 2017.

[15] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[16] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy. Deep exploration via bootstrapped DQN. *Advances in Neural Information Processing Systems*, 2016.

[17] T. Doan, B. Mazoure, and C. Lyle. Gan q-learning. *arXiv preprint arXiv:1805.04874*, 2018.

[18] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[19] D. P. Kingma, T. Salimans, and M. Welling. Improving variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems*, 2016.

[20] Y. Tang and S. Agrawal. Boosting trust region policy optimization by normalizing flows policy. *CoRR*, abs/1809.10326, 2018. URL http://arxiv.org/abs/1809.10326.

[21] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine. Latent space policies for hierarchical reinforcement learning. *arXiv preprint arXiv:1804.02808*, 2018.

[22] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[23] R. Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

[24] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[25] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

[26] T. Salimans, D. Kingma, and M. Welling. Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, pages 1218–1226, 2015.

[27] D. Hjelm, R. R. Salakhutdinov, K. Cho, N. Jojic, V. Calhoun, and J. Chung. Iterative refinement of the approximate posterior for directed belief networks. In *Advances in Neural Information Processing Systems*, pages 4691–4699, 2016.

[28] C. Huang, D. Krueger, A. Lacoste, and A. C. Courville. Neural autoregressive flows. *International Conference on Machine Learning*, 2018.

[29] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

[30] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[33] J. Pineau. *The machine learning reproducibility checklist, v.1.2*, 2019. URL https://www.cs.mcgill.ca/~jpineau/ReproducibilityChecklist.pdf.

[34] R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2): 411–423, 2001.

Supplementary Material

Reproducibility Checklist

We follow the reproducibility checklist [33] and point to relevant sections explaining them here. For all algorithms presented, check if you include:

- **A clear description of the algorithm, see main paper and included codebase.** The proposed approach is completely described by Alg. 1.
- **An analysis of the complexity (time, space, sample size) of the algorithm.** Experimentally, we demonstrate improvement in sample complexity as discussed in our main paper. In term of computation time, the proposed method retains the same running time as SAC, since the overhead for training the NF layers is minimal in comparison. The biggest advantage of SAC-NF with radial flows over SAC is its significantly reduced number of parameters. For instance, in the MuJoCo Hopper environment, SAC-NF uses only 5.5% of neural network parameters used by SAC while achieving much better performance.
- **A link to a downloadable source code, including all dependencies.** The code is included with Supplemental Material as a zip file; all dependencies can be installed using Python's package manager. Upon publication, the code would be available on Github.

For all figures and tables that present empirical results, check if you include:

- **A complete description of the data collection process, including sample size.** We use standard benchmarks provided in OpenAI Gym (Brockman et al., 2016) and PyBullet.
- **A link to downloadable version of the dataset or simulation environment.** See: http://www.mujoco.org/ and https://pybullet.org/wordpress/.
- **An explanation of how samples were allocated for training / validation / testing.** We do not use a training-validation-test split, but instead report the mean performance (and one standard deviation) of the policy at evaluation time across 5 random seeds.
- **An explanation of any data that were excluded.** We did not compare on easy environments (e.g. `Reacher-v2`) because all existing methods perform well on them. In that case, the improvement of our method upon baselines is incremental and not worth mentioning.
- **The exact number of evaluation runs.** 5 seeds for all experiments, 1M, 2M or 3M environment steps depending on the domain.
- **A description of how experiments were run.** See Section 5 in the main paper and didactic example details in Appendix.
- **A clear definition of the specific measure or statistics used to report results.** Undiscounted returns across the whole episode are reported, and in turn averaged across 5 seeds.
- **Clearly defined error bars.** Confidence intervals and table values are always mean$\pm$ 1 standard deviation over 5 seeds.
- **A description of results with central tendency (e.g. mean) and variation (e.g. stddev)**. All results use the mean and standard deviation.
- **A description of the computing infrastructure used.** All runs used 1 CPU for all experiments (toy, MuJoCo and PyBullet) with 8Gb of memory.

Experimental setup for ablation study

We compare the SAC-NF agent (Algorithm 1 with mini-batch size $m = 256$, 4 flows and one hidden layer of 8 neurons), which can represent radial policies, with a classical SAC agent(two hidden layers of 16 units) that models Gaussian policies. Both agents are trained over $T = 500$ epochs, each epoch consisting of 20 time steps.

Model parameters

We provide a table of hyperparameters used to obtain results in the MuJoCo and PyBullet domains. Note that h^1 corresponds to the average and h^2 to the conditional models.

NF parameters				
	# flows	Type	Alpha	Model
Ant-v2	4	radial	0.05	average
HalfCheetah-v2	3	radial	0.05	conditional
Hopper-v2	5	radial	0.05	average
Humanoid-v2	4	radial	0.05	average
Walker-v2	5	radial	0.05	conditional
Humanoid (Rllab)	2	radial	0.05	conditional
HumanoidPyBulletEnv-v0	3	radial	0.05	average
HumanoidFlagrunPyBulletEnv-v0	5	radial	0.05	conditional
HumanoidFlagrunHarderPyBulletEnv-v0	3	radial	0.05	conditional
HumanoidPyBulletEnv-v0	3	IAF	0.01	conditional
HumanoidFlagrunPyBulletEnv-v0	4	IAF	0.05	conditional
HumanoidFlagrunHarderPyBulletEnv-v0	3	IAF	0.01	average
HumanoidPyBulletEnv-v0	4	planar	0.01	conditional
HumanoidFlagrunPyBulletEnv-v0	3	planar	0.05	average
HumanoidFlagrunHarderPyBulletEnv-v0	3	planar	0.05	average
Adam Optimizer parameters				
α_γ	3.10^{-4}			
α_ω	3.10^{-4}			
α_θ	3.10^{-4}			
α_ϕ	3.10^{-4}			
Algorithm parameters				
m	256			
$\mathcal{B}$ size	10^6			

Table 3: SAC-NF parameters.

Environment	Gaussian	Radial	IAF	Planar
HumanoidPyBulletEnv-v0	82,463 (1)	15,963 (0.19)	17,436 (0.21)	**13,594** (0.16)
HumanoidFlagrunPyBulletEnv-v0	82,463 (1)	**12,397** (0.15)	18,864 (0.23)	16,875 (0.20)
HumanoidFlagrunHarderPyBulletEnv-v0	82,463 (1)	**12,359** (0.15)	21,040 (0.26)	16,875(0.20)

Table 4: Number of model parameters for SAC (Gaussian), SAC-NF (Radial, Planar and IAF) used to achieve results on the PyBullet environments. In parentheses, the ratio of parameters with respect to SAC (Gaussian) is shown. A value lower than 1.0 means a lower number of parameters than SAC baseline. While having the lowest number of parameters, radial flows achieve consistently best performances.

Performances against other baselines

	SAC	SAC-NF	TD3
Ant-v2	$4,372 \pm 900$	**4912 ± 954**	$4,372 \pm 900$
HalfCheetah-v2	**11410 ± 537**	8429 ± 818	$9,543 \pm 978$
Hopper-v2	3095 ± 730	**3538 ± 108**	**3,564 ± 114**
Humanoid-v2	**5505 ± 116**	**5506 ± 147**	71 ± 10
Humanoid (rllab)	2079 ± 1432	**5531 ± 4435**	286 ± 151
Walker2d-v2	3813 ± 374	**5196 ± 527**	$4,682 \pm 539$
SparseHalfCheetah-v2	767 ± 247	**939 ± 4**	809 ± 92
SparseHumanoid-v2	88 ± 159	**547 ± 268**	0 ± 0

Table 5: Maximal average return $\pm$ one standard deviation across 5 random seeds for SAC, TD3 and SAC-NF.

Toy navigation task

We conduct a synthetic experiment to illustrate how the augmentation of a base policy with normalizing flows allows to represent multi-modal policies. We consider a navigation task environment with continuous state and action spaces consisting of four goal states symmetrically placed around the origin. The agent starts at the origin and, on each time t, receives reward r_t corresponding to the Euclidean distance to the closest goal. We consider a SAC-NF agent (Algorithm 1 with mini-batch size $m = 256$, 4 flows and one hidden layer of 8 neurons) which can represent radial policies. The agent is trained over $T = 500$ epochs, each epoch consisting of 20 time steps.

Figure 5 displays some trajectories sampled by the SAC-NF agent along with the kernel density estimation (KDE) of terminal state visitations by the agent. Trajectories are obtained by sampling from respective policy distributions instead of taking the average action. We observe that the SAC-NF agent, following a flow-based policy, is able to successfully visit all four modes.

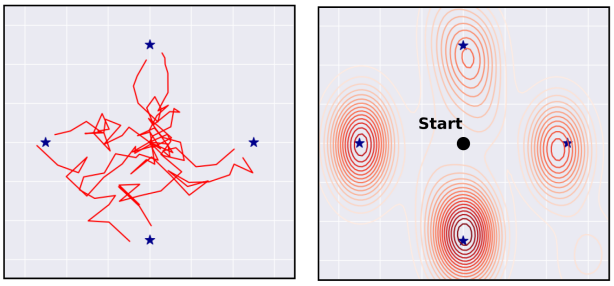

Figure 5: 4-goals navigation task outlining the ability of normalizing flows to learn multi-modal policies. The left subfigure shows some trajectories sampled from the SAC-NF agent. The right subfigure shows a KDE plot of terminal state visitations by the agent.

6.1 Assessing the shape of SAC-NF and its multimodality

While in simple domains the shape of a policy might not matter much, using a Gaussian policy in more complicated environments can yield suboptimal behaviour due to rigidity of its shape. To test whether SAC-NF with radial flows implicitly learns a Gaussian policy (i.e. most learning happens at the noise and not at the flow layers), we examine the KL divergence between a Gaussian distribution and SAC-NF policies trained on MuJoCo's `Ant-v2` environment.
As argued previously, heavy regularization of radial and planar flows approximately recovers the identity map $f(\mathbf{z}) = \mathbf{z}$, in which case the normalizing flow policy has a Gaussian shape centered at $\mu(\mathbf{s})$. However, when the flows are unconstrained, the policy is allowed to evolve as to maximize the evidence lower bound.

Figure 6 shows the evolution over time of a radial policy on the MuJoCo `Ant-v2` environment. The average KL divergence conditional on an observed state is computed between zero mean and unit variance radial flow and Gaussian policies, respectively. This standardization is done to eliminate the dependence of KL on the location and scale of the policy. For two multivariate Gaussian policies $\pi_1 \sim \mathcal{N}(\mu_1, \mathbf{I}), \pi_2 \sim \mathcal{N}(\mu_2, \mathbf{I})$ in a single state environment, the KL divergence follows this proportionality: $D_{KL}(\pi_1||\pi_2) \propto (\mu_1 - \mu_2)^\top(\mu_1 - \mu_2)$.
To ensure that the KL reports the difference in shape and not in location-scale, it is necessary to re-center and re-scale both policies (equivalent to superposing both policies on top of each other):

$$\begin{aligned} &\mathbb{E}_{\mathbf{s}}[\mathrm{D}_{KL}\{\pi_{NF}(\mathbf{a}|\mathbf{s})||\pi_{Gaussian}(\mathbf{a}|\mathbf{s})\}] \\ &= \mathbb{E}_{\mathbf{s}}[D_{KL}\{\pi_{NF}(\mathbf{a}|\mathbf{s})||\mathcal{N}(0, \mathbf{I})\}], \end{aligned} \tag{10}$$

and $\mathbb{E}_\pi[\pi_{NF}(\mathbf{a}|\mathbf{s})] = 0, \mathbb{V}_\pi[\pi_{NF}(\mathbf{a}|\mathbf{s})] = 1$ for all states $\mathbf{s}$ observed during rollouts. The differences in log probabilities for every given action are summed over all actions in the sample and averaged across states. We see that as training progresses, the state-averaged KL between the normalizing flow policy and the reference unit Gaussian increases.

Figure 6: Average KL divergence between 500 MC action samples taken from a radial flow policy scored against the standard Gaussian distribution and averaged over 1,000 states every 100k iterations. The KL divergence increases over the time, suggesting that the radial flow policy's shape gets further from that of a Gaussian distribution.

Now, we check the key property of SAC-NF, as a proposed improvement to unimodal Gaussian policies in SAC, is its ability to produce rich, multimodal policies. We can measure the degree of multimodality using the gap statistic method [34] for k-means hyperparameter selection. Once computed, this coefficient measures goodness-of-fit of k clusters (i.e. modes) to a given distribution. For that purpose, we collect 500 states from the given policy rollout, under which we sample 250 actions from the SAC-NF policy and evaluate the number of modes with the aforementioned test for each state separately. Differences between GS values for k=1 and k=2, and between k=1 and k=10 are given respectively and are averaged across all states: 6.8, 11.8 (Ant), 16.8, 31 (Walker), 3.2, 7.3 (Humanoid-rllab). Here, k is the number of clusters, higher difference means more likely to have

more than one mode. In comparison, all gap statistics for SAC have differences less than 1. These results suggest that the SAC-NF policies are multimodal (have at least 2 modes, note this is for each state, not marginalized over states). Since modes are not always clearly identifiable, we computed skewness (symmetry measure) and excess kurtosis (non-Gaussianity measure), respectively, for the same policy samples: -0.017, -0.26 (Ant), -0.34, -0.9 (Humanoid-rllab), -0.53, -0.57 (Walker). All three policies have large negative excess kurtosis (suggesting that they do not have a Gaussian shape), and have negative skew (policies learned on Humanoid-rllab and Walker are hence not symmetric). This evidence indicates that the shape of the policies learned by SAC-NF is, on average, not likely to be Gaussian.

6.2 Sparse rewards environments

Even if SAC-NF is meant to better track suboptimal solutions, we tested whether adding normalizing flow layers improves performance within sparse reward environments. To do so, we evaluated on Sparse Humanoid (SparseHU). For SparseHU, a reward of +1 is granted when the agent reaches a distance threshold above 0.6. As shown in Figure 7, SAC-NF has better performance than its SAC counterpart and TD3 which struggle to take off.

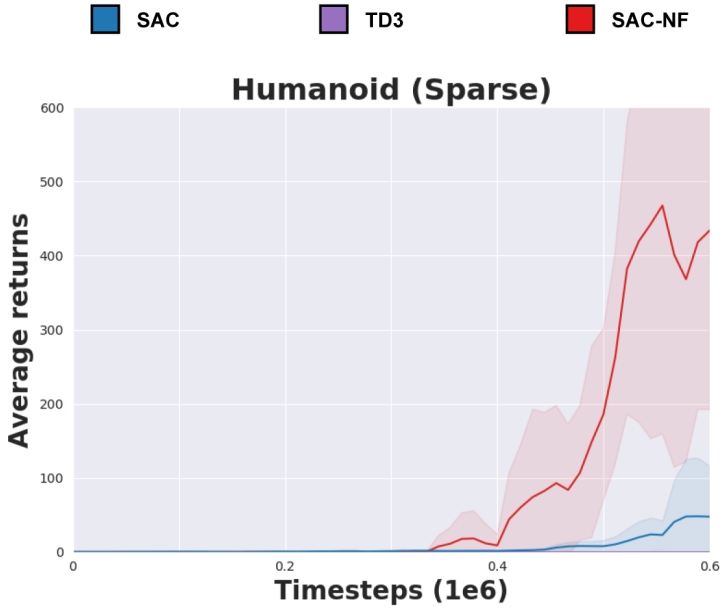

Figure 7: Performance of SAC-NF compared against SAC (Gaussian policy) for a sparse environment in which reward is observed after the agent reaches a certain threshold distance.