Leveraging Constraint Violation Signals For Action-Constrained Reinforcement Learning

Janaka Chathuranga Brahmanage, Jiajing Ling, Akshat Kumar

School of Computing and Information Systems, Singapore Management University.
{janakat.2022, jjling.2018}@phdcs.smu.edu.sg, akshatkumar@smu.edu.sg

Abstract

In many RL applications, ensuring an agent's actions adhere to constraints is crucial for safety. Most previous methods in Action-Constrained Reinforcement Learning (ACRL) employ a projection layer after the policy network to correct the action. However projection-based methods suffer from issues like the zero gradient problem and higher runtime due to the usage of optimization solvers. Recently methods were proposed to train generative models to learn a differentiable mapping between latent variables and feasible actions to address this issue. However, generative models require training using samples from the constrained action space, which itself is challenging. To address such limitations, *first*, we define a target distribution for feasible actions based on constraint violation signals, and train normalizing flows by minimizing the KL divergence between an approximated distribution over feasible actions and the target. This eliminates the need to generate feasible action samples, greatly simplifying the flow model learning. *Second*, we integrate the learned flow model with existing deep RL methods, which restrict it to exploring only the feasible action space. *Third*, we extend our approach beyond ACRL to handle state-wise constraints by learning the constraint violation signal from the environment. Empirically, our approach has significantly fewer constraint violations while achieving similar or better quality in several control tasks than previous best methods.

Code — https://github.com/rlr-smu/cv-flow

1 Introduction

Reinforcement learning has been successfully applied to solve a variety of problems, ranging from mastering Atari games (Mnih et al. 2015; Van Hasselt, Guez, and Silver 2016) to controlling robotics (Pham, De Magistris, and Tachibana 2018; Thananjeyan et al. 2021) and fortifying system security (Adawadkar and Kulkarni 2022; Khoury and Nassar 2020), among others. In many real-world applications, agents have to take actions within a feasible action space defined by some constraints at every RL step. This scenario falls under the domain of action-constrained reinforcement learning (ACRL) (Brahmanage, Ling, and Kumar 2023; Kasaura et al. 2023). In ACRL, action constraints typically take an analytical form based on state and action features (e.g., $\sum_{i=1}^{2} |s_i a_i| <= 1$ with s_i, a_i as features) rather than being expressed using a predefined cost function.

Representative applications of ACRL include robotic control (Kasaura et al. 2023; Lin et al. 2021; Pham, De Magistris, and Tachibana 2018) and resource allocation in supply-demand matching (Bhatia, Varakantham, and Kumar 2019), where kinematic limitations and resource restrictions are effectively modeled using action constraints derived from system specifications. When action constraints are not directly available, they can be approximated using state-based cost functions or offline datasets of valid and invalid trajectories. In such cases, action constraints can be inferred from environmental data (Dalal et al. 2018) or via inverse constraint RL (Malik et al. 2021).

Since action constraints with a closed form can be relatively easy to evaluate for each action, one natural approach is to use a *projection* to generate feasible actions that satisfy the constraints, which involves solving a math program (Amos and Kolter 2017; Bhatia, Varakantham, and Kumar 2019; Dalal et al. 2018; Lin et al. 2021; Pham, De Magistris, and Tachibana 2018). The projection-based approach can either result in a zero gradient problem (Lin et al. 2021) or an expensive overhead due to solving an optimization program (Brahmanage, Ling, and Kumar 2023) or both. When dealing with non-convex action constraints or large action spaces, there is a significant reduction in training speed, which we also validate empirically.

Recently, (Brahmanage, Ling, and Kumar 2023) learn a smooth and invertible mapping between a simple latent space and the feasible action space given states using conditional normalizing flows (Dinh, Sohl-Dickstein, and Bengio 2016) and integrate it with Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2016). This approach effectively avoids the zero gradient problem common in standard projection-based methods as there is no coupling between the policy network and projection layer anymore. However, a key drawback is the necessity to generate feasible actions in advance for training the flow, which is challenging for complex constraints since specialized methods such as Hamiltonian Monte-Carlo (HMC), decision diagrams are required to sample from the feasible action space (Brahmanage, Ling, and Kumar 2023).

In addition to ACRL, generative models are also used for improving exploration and utilizing off-policy data in

RL. In the context of enhancing exploration (Mazoure et al. 2020; Ward, Smofsky, and Bose 2019), generative models are used as a policy for optimizing the maximum entropy objective of Soft Actor Critic (SAC) (Haarnoja et al. 2018). Also, (Fujimoto, Meger, and Precup 2019; Zhang et al. 2023) utilize off-policy data to train a normalizing flow to improve the exploration. Recent work (Changyu et al. 2023) uses Argmax Flow (a type of normalizing flow) with RL to deal with discrete action spaces.

Contributions Our main contributions are as follows.

First, we learn an invertible, differentiable mapping from a simple base distribution (e.g., Gaussian) of the normalized flow model to the feasible action space. Standard methods for flow model training require generating data (feasible environment action samples) using specialized methods such as HMC and decision diagrams (Dinh, Sohl-Dickstein, and Bengio 2016; Brahmanage, Ling, and Kumar 2023). Our method instead directly trains the flow by minimizing the KL divergence between a flow model based distribution and a target density over the feasible action space defined using the constraint violation signal. This avoids the costly step of generating samples from the feasible action space.

Second, we integrate such a trained flow model with SAC (Haarnoja et al. 2018). We also present an analytical method for computing entropy exclusively over the feasible action space, which offers significant advantages over the naive implementation of SAC for ACRL. This method can avoid the zero-gradient problem, as a properly calibrated flow model ensures a one-to-one mapping from the support of a simple distribution (e.g., Gaussian) to the feasible action space without requiring an optimization-based projection layer.

Third, we present a methodology to extend our approach beyond ACRL to address state-wise constraints (non-explicit constraints). We achieve this by learning the constraint violation signal through interaction with the environment.

Finally, we train the flow model on a variety of action constraints and state-wise constraints used in the ACRL literature, leading to an accurate approximation of the feasible action space. Our approach results in fewer constraint violations (>10x for a number of benchmarks) while achieving similar or better solution qualities on a variety of continuous control tasks than the previous best methods (Brahmanage, Ling, and Kumar 2023; Kasaura et al. 2023; Lin et al. 2021).

2 Preliminaries

2.1 Action-Constrained MDP

An action-constrained Markov Decision Process (MDP) is a standard MDP augmented with explicit action constraints. An MDP is defined using a tuple $\langle S, A, p, r, \gamma, b_0 \rangle$, where S is the set of possible states, A is the unconstrained action space–a set of possible actions that an agent can take. The $p(s'|s,a)$ is the probability of leading to state s' after taking action a in state s; $r(s,a)$ is the immediate reward received after taking action a in state s; $\gamma \in [0,1)$ is the discount factor, and b_0 is the initial state distribution. Given a state $s \in S$, we define a feasible action space $\mathcal{C}(s) \subseteq A$ using m inequality and n equality constraints as:

$$\mathcal{C}(s) = \{a | a \in A, g_i(a,s) \leq 0, h_j(a,s) = 0, i = 1:m, j = 1:n\} \quad (1)$$

where g_i and h_j are arbitrary functions of state and action used to define inequality and equality constraints. We assume continuous state and action spaces. Let μ_ϕ denote the policy parameterized by ϕ. For a stochastic policy, we use $\mu_\phi(a|s)$ to denote the probability of taking action a in state s. In RL setting transition and reward functions are not known. The agent learns a policy by interacting with the environment and using collected experiences (s_t, a_t, r_t, s_{t+1}). The goal is to find a policy that maximizes the expected discounted total reward while ensuring that all chosen actions are taken from the feasible action space:

$$\begin{aligned} \max_{\phi} \quad & J(\mu_\phi) = \mathbb{E}_{\mu_\phi, s_0 \sim b_0}[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] \\ \text{s.t.} \quad & a_t \in \mathcal{C}(s_t) \quad \forall t \end{aligned} \quad (2)$$

In ACRL, the environment simulator only accepts feasible actions, as assumed in previous work (Lin et al. 2021; Kasaura et al. 2023). Any infeasible actions will lead to the termination of the simulator. That is why a projection is typically used to ensure that any infeasible actions are mapped into the feasible action space.

2.2 *State-Constrained* MDP

In some real-world problems, constraints only involve state features. For example, in safe RL, when an agent takes an action a in a state s, it must ensure that the next state s' is safe (Dalal et al. 2018; Zhao et al. 2023). These are referred to as state-wise constraints. Mathematically, they can be defined as $c_i(s) \leq 0, i = 1 \ldots k$. Here c_i are state based cost functions. As shown in (Dalal et al. 2018), each state-wise constraint can be transformed into an action constraint by approximating the constraint violations in the next state using a neural network w_i as follows.

$$c_i(s_{t+1}) \approx c_i(s_t) + w_i(s_t)^T a_t \leq 0, \quad i = 1 \ldots k \quad (3)$$

As a pretraining step, we run a random policy to collect experience (s_t, a_t, r_t, s_{t+1}), and the associated costs $c(s_t)$ and $c(s_{t+1})$ for the current and next states. We then train the neural network w_i by minimize the Mean Squared Error (MSE) loss between the predicted cost $c_i(s_t) + w_i(s_t)^T a_t$ and the observed cost $c_i(s_{t+1})$. Once trained, this model defines a feasible action space $\mathcal{C}(s) = \{a | a \in A, c_i(s) + w_i(s)^T a \leq 0; \forall i\}$.

2.3 Existing Approaches to Solving ACRL

As discussed in the introduction, there are two popular approaches to solving ACRL. Figure 1(b) illustrates the projection-based approach. In this method, the policy network μ parameterized by ϕ receives a state s and outputs an action $\hat{a}$. When $\hat{a}$ is infeasible—typically the case at the beginning of the training stage—a quadratic programming (QP) solver is used to project $\hat{a}$ into the feasible action space, resulting in a feasible environment action a. Figure 1(a) shows the mapping-based approach, where the policy network μ_ϕ generates $\hat{a}$ in a latent space rather than the environment

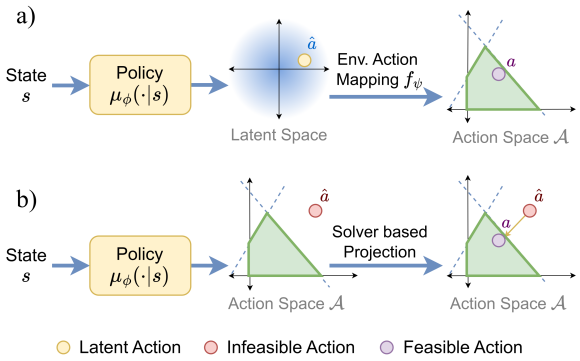

Figure 1: Two approaches to integrate action constraints with RL: (a) Mapping-based approach and (b) Projection-based approach.

action space. This latent action $\hat{a}$ is then converted into an environment action using a mapping function f parameterized by ψ, such that $a = f_\psi(\hat{a})$.

We use a mapping-based approach since it offers several advantages over a projection layer. Unlike a projection layer, which suffers from the zero-gradient problem—where multiple infeasible actions map to a single action, limiting the agent's ability to learn effectively—and distorts probability density by concentrating actions in bordering regions, the mapping function avoids these issues and ensures that the agent can explore the feasible action space evenly. In our work, we use normalizing flows as the mapping function f_ψ.

2.4 Normalizing Flows

A normalizing flow model is a type of generative model. It transforms a simple *base distribution* such as a Gaussian into a more complex distribution through an invertible transformation function (Rezende and Mohamed 2015; Dinh, Sohl-Dickstein, and Bengio 2016).

In the context of solving ACRL, normalizing flows are used to map the base distribution into a feasible environment action density distribution. Given a state s, a sample $\hat{a}$ from the base distribution $\hat{q}$, and state-conditioned normalizing flows f parameterized by ψ, we obtain a feasible environment action a as $a = f_\psi(\hat{a}, s)$. Let $q(a|s)$ denote the probability of obtaining a through normalizing flows. Since f is bijective, the log probability can be computed using the change of variables theorem as follows:

$$\log q(a|s) = \log \frac{\hat{q}(\hat{a})}{|\det J_{f_\psi}(\hat{a}; s)|} \tag{4}$$

where $|\det J_{f_\psi}(\hat{a}; s)|$ is the determinant of the Jacobian of f_ψ, which accounts for the change in volume when transforming the base distribution to the feasible action density distribution (Nielsen et al. 2020).

When a training dataset of feasible state-action pairs is available, the function f is learned by maximizing the log-likelihood of the data (Dinh, Sohl-Dickstein, and Bengio 2016). Conversely, when feasible state-action pairs are not provided but the feasible environment action distribution conditioned on state is known, f can be learned by minimizing

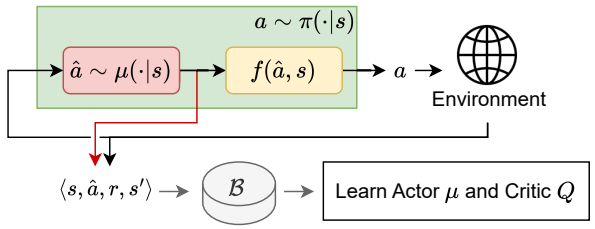

Figure 2: Flow Model integration with the SAC Policy: μ represents the original policy network, $\hat{a}$ is the latent action, and f is the mapping function that maps the latent action into a feasible environment action a. π represents the combined policy. The latent action $\hat{a}$ is stored in the replay buffer to train both the μ and critic networks.

the reverse Kullback-Leibler (KL) divergence between the true target distribution denoted by $p(a|s)$ and $q(a|s)$ (Papamakarios et al. 2021). In ACRL, we want the model to generate all feasible actions with equal likelihood while avoiding any infeasible actions. Therefore, we define $p(a|s)$ as a uniform distribution over feasible environment actions, assigning zero probability to infeasible actions. Given that the reverse-KL loss for a given state s is:

$$\begin{aligned}\mathcal{L}(s) = \mathrm{KL}(q||p) &= \mathop{\mathbb{E}}_{\substack{\hat{a}\sim\hat{q}(\hat{a}) \\ a=f_\psi(\hat{a},s)}} [\log q(a|s) - \log p(a|s)] \\ &= \mathop{\mathbb{E}}_{\hat{a}\sim q(\hat{a})} \left[\log \frac{\hat{q}(\hat{a})}{|\det J_{f_\psi}(\hat{a}; s)|} - \log p(f_\psi(\hat{a}, s)|s) \right] \end{aligned} \tag{5}$$

3 Methodology

3.1 Overview

Our architecture, as shown in Figure 2, consists of two components: the policy μ_ϕ and the mapping function f_ψ. We refer to the combination of these two as the combined policy $\pi_{\phi,\psi}$. The training process involves two primary steps. First, we train the mapping function f_ψ, a normalizing flow. Unlike previous approaches, we do not assume the availability of a dataset of feasible environment actions, so we utilize the reverse KL divergence Eq. (5) to train the flow model. Second, we train the reinforcement learning agent. In contrast to previous work (Brahmanage, Ling, and Kumar 2023), we use the latent action $\hat{a}$ instead of environment action $\hat{a}$ to train both the actor and critic networks. This approach offers a computational advantage, as we do not need to backpropagate through the flow model during RL training. In Sections 3.2 and 3.3, we discuss these steps in detail.

3.2 Feasible Action Mapping Using $\mathbb{CV}$-Flows

As noted in Section 2, a standard approach for training normalizing flows as a mapping function is to maximize the log-likelihood over the training dataset (feasible state-action pairs in our setting). However, a key challenge lies in generating a sufficient number of feasible samples from the constrained action space. This often requires techniques like rejection sampling or advanced methods such as Markov-Chain Monte Carlo (MCMC) (Brubaker, Salzmann, and Urtasun 2012),

which can be computationally intractable and sample inefficient with high rejection rate for complex constraints as shown in (Brahmanage, Ling, and Kumar 2023).

To eliminate the need to generate feasible state-action samples to train normalizing flows, we propose a novel approach, namely Constraint Violation-Flows ($\mathbb{CV}$-Flows) to train the normalizing flows. We first define a true target distribution $p(a|s)$ over feasible state-action samples using constraint violation signals. Then we train the flow by minimizing the reverse KL divergence between an approximated distribution of feasible samples ($q(a|s)$ in Eq. (4)) and the true target distribution, as shown in Eq. (5).

Defining the true target distribution: We consider the constrained action space defined by Eq. (1). We first define the magnitude of constraint violations (CV) as follows:

$$\mathrm{CV}(a,s) = \sum_{i=1}^{m} \max(g_i(a,s),0) + \sum_{j=1}^{n} \max(|\,|h_j(a,s)| - \epsilon|, 0)$$

where ϵ is an error margin for equality constraints. The function $\mathrm{CV}(a,s)$ evaluates to zero if and only if an action a is within the constrained action space, increasing as it moves away from this feasible region. Then we define a non-negative measure over this region as follows:

$$\tilde{p}(a|s) = e^{-\lambda\,\mathrm{CV}(a,s)} \tag{6}$$

This ensures that $\tilde{p}(a|s)$ remains positive in the absence of constraint violations and decreases exponentially with increasing $\mathrm{CV}(a,s)$. The rate parameter λ of the exponential distribution determines the steepness of the probability decrease as constraint violations increase. A larger λ is preferred to boost the loss for even small constraint violations. It was set to 1000 for all the experiments.

Lastly, to obtain a valid probability distribution given s, we normalize the non-negative measure by dividing a constant $M(s) = \int_{-\infty}^{\infty} \tilde{p}(a|s)da$. Thus, we have,

$$p(a|s) = \frac{\tilde{p}(a|s)}{M(s)} \tag{7}$$

We note that this target distribution is a uniform distribution over feasible environment actions given s since $\mathrm{CV}(a,s)$ is the same for every single feasible environment action. This is also desirable as it implies that after flow training, $q(a|s)$ will be able to generate most feasible environment actions. The probability for infeasible actions will be very close to zero.

Training the flow model: To train the normalizing flow, we first need to sample a sufficient number of $\hat{a} \sim \hat{q}(\hat{a})$ from a base distribution $\hat{q}$ (e.g., standard Gaussian). Sampling $\hat{a}$ from the standard Gaussian is much easier than sampling feasible environment actions from $p(a|s)$. Substituting the true target distribution Eq. (7) in Eq. (5), the loss function (for a state s) is:

$$\begin{aligned}\mathcal{L}(s) &= \mathop{\mathbb{E}}_{\hat{a}\sim\hat{q}(\hat{a})}\left[\log\frac{\hat{q}(\hat{a})}{|\det J_{f_\psi}(\hat{a};s)|} - \log\frac{\tilde{p}(f_\psi(\hat{a},s)|s)}{M(s)}\right] \\ &= \mathop{\mathbb{E}}_{\hat{a}\sim\hat{q}(\hat{a})}\left[\log\frac{\hat{q}(\hat{a})}{|\det J_{f_\psi}(\hat{a};s)|} - \log\frac{e^{-\lambda\,\mathrm{CV}(f_\psi(\hat{a},s),s)}}{M(s)}\right]\end{aligned} \tag{8}$$

Algorithm 1: $\mathbb{CV}$-Flows Pretraining Algorithm

```
Initialize normalizing flow f_ψ, with random weights ψ
for epoch = 1...N do
    â ~ q̂(â) {Batch sample â from the base dist.}
    s ~ p_S(s) {Batch sample s from state space}
    ψ ← ψ − λ_f ∇̂_ψ J^f(ψ) {Update flow model}
end for
```

Since both $\hat{q}(\hat{a})$ and $M(s)$ are independent of ψ, they can be excluded. Then, given a probability distribution over the state space $p_S(s)$, the final loss function can be written as:

$$J^f(\psi) = \mathop{\mathbb{E}}_{s\sim p_S,\hat{a}\sim\hat{q}}[\lambda\,\mathrm{CV}(f_\psi(\hat{a},s),s) - \log|\det J_{f_\psi}(\hat{a};s)|] \tag{9}$$

We note that the distribution $p_S(s)$ can be uniform if the state space is bounded; otherwise, it can be Gaussian. In the case of a Gaussian distribution, the mean and standard deviation can be determined by running an unconstrained agent in the environment. Alternatively, state samples can be collected directly from the environment by operating the agent under a different policy. The pseudo-code of our proposed approach to training the $\mathbb{CV}$-Flows is provided in Algorithm 1.

Training the flow model for state-wise constraints: For state-wise constraints we do not have access to the analytical form of the action constraint. Instead, we only have state constraints $c_i(s)$ in the form of Eq. (3). We follow the linear approximation model discussed in Section 2.2, which results in CV function in Eq. (10):

$$\mathrm{CV}(a,s) = \sum_{i=1}^{k} \max(c_i(s) + w_i(s)^T a, 0) \tag{10}$$

In Eq. (10), w_i is learned through a pretraining process using the transitions collected by running a random agent (Dalal et al. 2018) in the environment, as discussed in Section 2.2. Subsequently, we use this CV signal to train the $\mathbb{CV}$-Flows, as discussed in the previous section.

Benefits of using $\mathbb{CV}$-Flows: The benefits of $\mathbb{CV}$-Flows are twofold. First, it integrates seamlessly with maximum entropy deep RL methods like SAC (details in the next section). Assuming a Gaussian base, the flow model maps policy samples to the feasible region differentiably, avoiding optimization solvers and the zero-gradient issue in RL policy training. Second, normalizing flows outperform GANs and VAEs in maximum entropy RL by enabling efficient log-probability computation due to bijectivity. Prior work also shows flow models offer better accuracy and recall for mapping to feasible action spaces (Brahmanage, Ling, and Kumar 2023).

3.3 Integrating RL Agent With the $\mathbb{CV}$-Flows

As shown in the previous section, the normalizing flow model maps the base distribution to the feasible action space. Here, we demonstrate its integration with the existing RL algorithm SAC (Haarnoja et al. 2018). We do not change the training loop of the base algorithm. It collects rollout from the environment and save $\langle s, \hat{a}, r, s'\rangle$ are stored in a replay buffer $\mathcal{B}$, which will be used to update the policy network μ_ϕ and the critic network Q_θ where θ represents the parameters. But

we store the latent action $\hat{a}$ in the replay buffer instead of the final action a, because we want to train both critic and policy networks using the latent action to avoid backpropagation through the flow model. Additionally, during this step, we keep the flow model parameters ψ fixed. The mapping function may not be fully accurate, so action a can violate constraints. To ensure feasibility, we add a projection step to map a into the feasible region if needed.

The base policy μ_ϕ produces the latent action $\hat{a}$, which is mapped to the feasible environment action a by f_ψ. Let us call the composition of base policy μ_ϕ and the normalizing flow f_ψ that generates a feasible environment action as $\pi_{\phi,\psi}$. When writing the objectives for SAC (Haarnoja et al. 2018) based on this combined policy, we get the following objectives for critic and policy update.

$$J^\pi(\phi) = - \mathop{\mathbb{E}}_{s\sim\mathcal{B}, a\sim\pi(\cdot|s)} \left[Q^{\pi(s,a)}(s,a) - \alpha \log \pi(a|s)\right] \quad (11)$$

$$J^Q(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{B}, a'\sim\pi(\cdot|s')} [(Q_\theta(s,a) - (r + \gamma(Q_{\bar{\theta}}(s',a') - \alpha \log \pi(a'|s'))))^2] \quad (12)$$

where $\alpha > 0$ is the trade-off coefficient of the entropy regularization term and γ is the discount factor. $Q_{\bar{\theta}}$ is the target network (Haarnoja et al. 2018). However, this objective is written in terms of combined policy π and the environment action a. We want to change this objective to use the latent action $\hat{a}$ instead of a, so we can avoid the need to backpropagate through the mapping function during RL training.

Proposition 1. *The log-probability of the combined policy,* $\log \pi(a|s)$*, can be approximated using* $\hat{a}$ *as:*

$$\log \pi(a|s) \approx \log \mu_\phi(\hat{a}|s) + \frac{\|\hat{a}\|_2^2}{2} + K(s)$$

where $K(s)$ is a constant independent of the action a.

Proof. We first apply the change of variables theorem to the combined policy, which results in:

$$\log \pi(a|s) = \log \mu_\phi(\hat{a}|s) - \log |\det J_{f_\psi}(\hat{a}; s)| \quad (13)$$

Then we eliminate the Jacobian term by considering another property of the trained mapping function: the trained model $q(a|s)$ should approximate the target distribution $p(a|s)$ which is a uniform distribution over feasible environment actions as noted in Eq. (7) (i.e. $\log p(a|s) \approx \log q(a|s)$). By substituting $\log q(a|s)$ from Eq. (4) we get,

$$\log p(a|s) \approx \log \hat{q}(\hat{a}) - \log |\det J_{f_\psi}(\hat{a}; s)| \quad (14)$$

Then we subtract Eq. (14) from Eq. (13) to eliminate the common term $\log |\det J_{f_\psi}(\hat{a}; s)|$, which results in:

$$\log \pi(a|s) \approx \log \mu_\phi(\hat{a}|s) + \log p(a|s) - \log \hat{q}(\hat{a}) \quad (15)$$

Here, the base distribution $\hat{q}(\hat{a})$ is a standard Gaussian (d-dimensional) and $p(a|s)$ is constant for all feasible actions given s. Thus, we have,

$$\begin{aligned}\log \pi(a|s) &\approx \log \mu_\phi(\hat{a}|s) + \log p(a|s) - \log \frac{1}{\sqrt{(2\pi)^d}} e^{-\frac{\hat{a}^T\hat{a}}{2}} \\ &\approx \log \mu_\phi(\hat{a}|s) + \log p(a|s) + \log \sqrt{(2\pi)^d} + \frac{\|\hat{a}\|_2^2}{2} \\ &\approx \log \mu_\phi(\hat{a}|s) + \frac{\|\hat{a}\|_2^2}{2} + K(s) \quad (16)\end{aligned}$$

□

Loss Functions: We substitute $\log \pi(a|s)$ using Eq. (16) in the objective Eq. (11) and Eq. (12) to get our final objectives for policy and critic update. $K(s)$ can be ignored as it is constant for all feasible actions.

$$J^\mu(\phi) = - \mathop{\mathbb{E}}_{s\sim\mathcal{B}, \hat{a}\sim\mu_\phi(\cdot|s)} [Q^{\mu_\phi}(s,\hat{a}) - \alpha(\log \mu_\phi(\hat{a}|s) + \frac{\|\hat{a}\|_2^2}{2})] \quad (17)$$

$$J^Q(\theta) = \mathop{\mathbb{E}}_{(s,\hat{a},r,s')\sim\mathcal{B}, \hat{a}'\sim\mu_\phi(\cdot|s')} [(Q_\theta(s,\hat{a}) - (r + \gamma(Q_{\bar{\theta}}(s',\hat{a}') - \alpha \log(\mu_\phi(\hat{a}'|s') + \frac{\|\hat{a}'\|_2^2}{2}))))^2] \quad (18)$$

For notation simplicity, we ignore the reparameterization (Kingma and Welling 2022) for sampling actions in the objective. However, we do use this when optimizing the parameters ϕ.

4 Experimental Results

The goal of our experiments is to: (1) evaluate whether our approach results in fewer constraint violations during training compared to other approaches, without sacrificing returns; (2) whether $\mathbb{CV}$-Flows can be adapted successfully for state-wise constraints where the analytical form of the action constraints is not available; (3) assess whether the $\|a\|_2^2$ term in the entropy regularization segment of SAC is truly beneficial; and (4) determine whether $\mathbb{CV}$-Flows yields higher accuracy while covering most of the feasible region compared to the standard method of training the flow using a sampled dataset of feasible environment actions;

Action-constrained environments: We evaluate our approach on four MuJoCo (Todorov, Erez, and Tassa 2012) continuous control environments: Reacher (R), Hopper (H), Walker2D (W), and HalfCheetah (HC). Using action constraints from previous work (Kasaura et al. 2023), we establish eight constrained control tasks: *R+L2*, *H+M*, *H+O+S*, *W+M*, *W+O+S*, *HC+O*, *R+D*, and *H+D*. These constraints restrict joint movement in each task, with details in Table 1 of the supplementary. *R+D* and *H+D* are non-convex constraints derived from modified convex constraints in (Kasaura et al. 2023), aimed at evaluating the efficiency of projection-based methods for challenging non-convex problems.

State-constrained environments: We evaluate our approach on four continuous control tasks with state-wise constraints: *Ball1D*, *Ball3D*, *Space-Corridor*, and *Space-Arena*, as proposed in previous work (Dalal et al. 2018). In *Ball1D* and *Ball3D*, the goal is to move a ball as close as possible to a target by adjusting its velocity within safe regions of $[0, 1]$ and $[0, 1]^3$, respectively. In *Space-Corridor* and *Space-Arena*, the task is to navigate a spaceship to a target location by controlling thrust engines within a two-dimensional universe, avoiding walls. Unlike in action constrained environments, episodes terminate upon violation of state-wise constraints.

Baselines: We integrated $\mathbb{CV}$-Flows with a standard Gaussian base distribution with SAC as described in the Section 3.3, referring to it as **SAC+CVFlow**. Our approach is compared with four baseline algorithms, **DPre+**, **SPre+**, **NFW** and **FlowPG** which have shown best results in previous

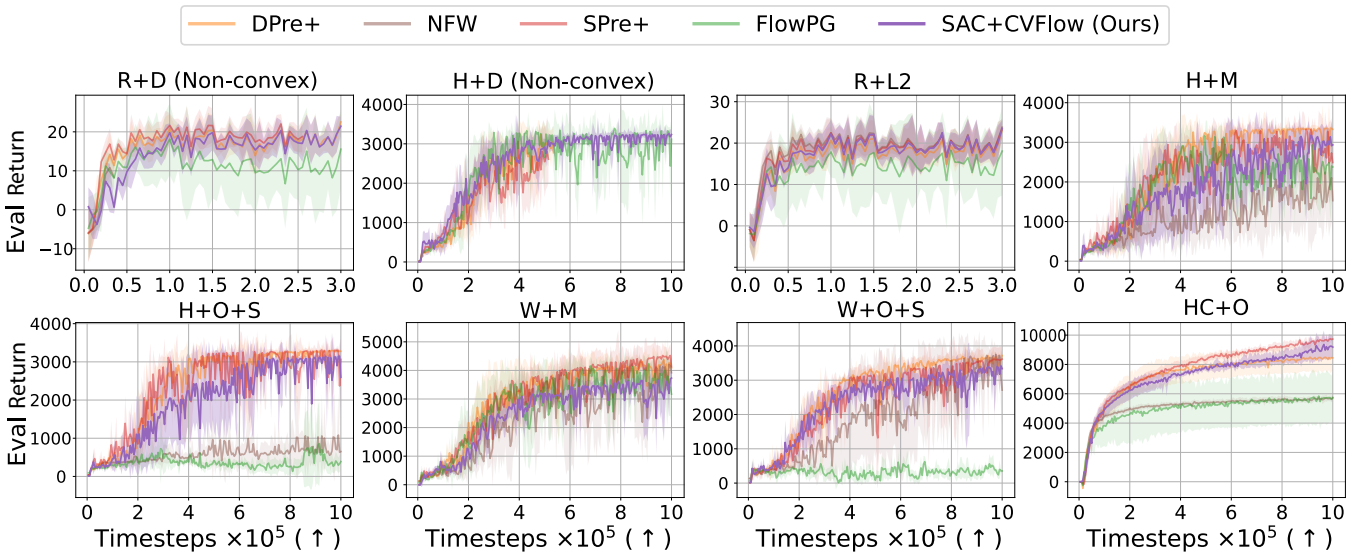

Figure 3: Evaluation returns for eight MuJoCo continuous control tasks during training. A higher return is better.

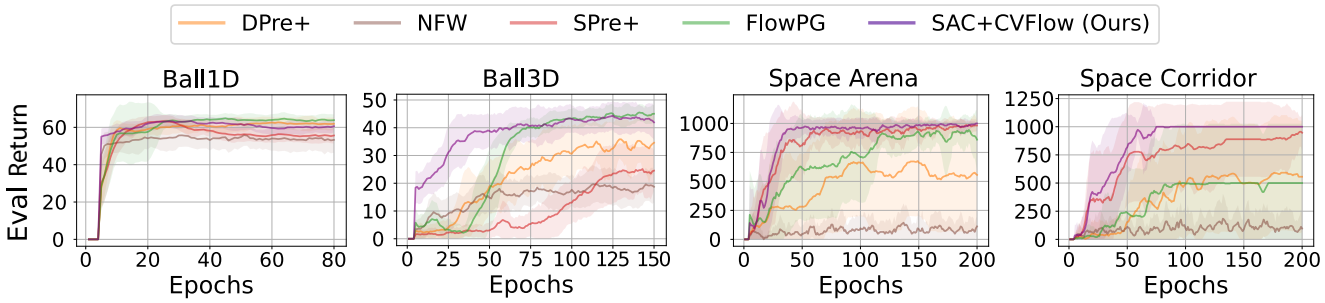

Figure 4: Evaluation returns for four state-constrained tasks during training. A higher return (↑) is better.

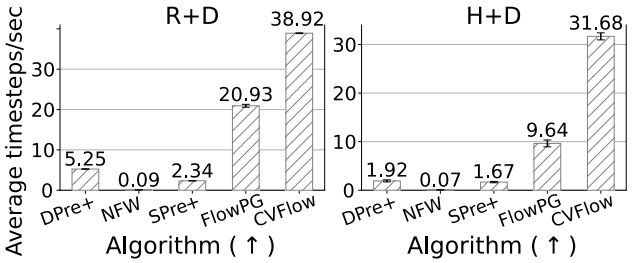

Figure 5: Average timesteps per second of the RL agent for non-convex constraints tasks (higher is better ↑), CVFlow based approach has a significantly higher frame rate.

studies (Kasaura et al. 2023; Lin et al. 2021; Brahmanage, Ling, and Kumar 2023). DPre+ is a DDPG-based algorithm with an additional projection step, ensuring that the projected action is feasible. SPre+ is same to DPre+ except for the underlying RL algorithm, which is SAC. Both DPre+ and SPre+ employ a penalty term based on the constraint violation signal. Neural Frank Wolfe (NFW) Policy Optimization (Lin et al. 2021) is also a DDPG-based algorithm. It leverages the Frank-Wolfe algorithm to update the policy network instead of the standard policy gradient. FlowPG (Brahmanage, Ling, and Kumar 2023) is built upon DDPG and uses normalizing flows to map the latent action into the feasible region. However, unlike our method, FlowPG requires feasible state-action pairs for training the normalizing flows and does not use the maximum entropy objective with RL. All algorithms include a projection step to correct actions if they are infeasible, as ACRL requires only feasible actions. In state-constrained environments, where action constraints are linearly approximated (as discussed in Section 2.2), the projection step might still produce infeasible actions, which can result in the termination of the episode.

Each algorithm is trained with 10 random seeds, capped at 48 hours per run, using hyperparameters and architectures from (Kasaura et al. 2023) (details in supplementary material).

4.1 Performance on MuJoCo Tasks

Reward comparisons: Evaluation returns are computed by running five episodes per random seed every 5k training steps. Figure 3 shows that our approach SAC+CVFlow achieves

Problem	DPre+	NFW	SPre+	FlowPG	Ours
R+D	98.15	95.50	97.03	24.79	**0.01**
H+D	74.10	74.89	77.72	32.29	**2.18**
R+L2	82.51	22.71	16.47	**0.03**	1.70
H+M	3.67	4.45	3.25	4.93	**0.25**
H+O+S	42.44	**2.14**	7.83	53.91	2.42
W+M	30.55	4.50	11.70	16.40	**2.41**
W+O+S	84.89	3.00	20.44	47.80	**1.55**
HC+O	73.57	9.73	46.66	61.10	**5.04**
Ball1D	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Ball3D	16.01	**0.00**	23.16	4.07	0.37
SpaceC	54.32	89.54	23.71	59.07	**12.06**
SpaceA	51.65	91.74	13.60	29.32	**10.78**

Table 1: The percentage of constraint violations during RL training. A lower value is preferable. The standard deviation of the constraint violations is reported in the supplementary. The abbreviations SpaceC and SpaceA refer to the Space-Corridor and Space-Arena benchmarks, respectively.

comparable results to DPre+ and SPre+ in terms of evaluation return across all the tasks. It finds high-quality policies that yield good returns with fewer constraint violations as discussed next.

Constraint violations: We measured the percentage of timesteps during the training period in which the agent produced infeasible actions before the projection step. This is equal to the number of QP-solver calls because each infeasible action must be projected into the feasible region before it is executed in the environment. As shown in Table 1 our approach results in fewer constraint violations across all tasks compared to all the baselines, except for (H+O+S, R+L2). In some cases (i.e., H+M, R+D, H+D), our approach even achieves a reduction in constraint violations by an order-of-magnitude compared to all the baselines. In the case of R+L2, we also observe an order-of-magnitude reduction in constraint violations compared to DPre+, SPre+, and NFW. Even in cases where our approach ranks second (H+O+S, R+L2), it yields comparable results to that of the best baseline. Furthermore, we observe that our method not only has fewer constraint violations but the magnitude of these violations is also lower as shown in the Figure 9 of the supplementary. This indicates that even when the constraints are violated the produced infeasible action is still closer to the feasible region.

Runtime: In terms of timesteps per second, our algorithm generally achieves comparable results to baseline methods, except for the non-convex tasks R+D and H+D, where it demonstrates significantly faster runtime, as shown in Figure 5. For non-convex tasks, our approach achieves an order of magnitude faster runtime compared to DPre+, SPre+ and NFW, while achieving at least twice as fast as FlowPG. The overhead cost in DPre+, SPre+, and NFW arises from using solvers to find feasible actions that satisfy non-convex constraints. FlowPG, on the other hand, incurs significant runtime due to backpropagation through the normalizing flow during training. Results for timesteps per second on other tasks can be found in Figure 8 of the supplementary material, along with computing infrastructure details.

4.2 Performance on State-Wise Constraints

Reward comparisons: The advantage of our approach is further demonstrated in state-constrained environments. As shown in Figure 4, our approach outperforms baseline methods in terms of return across all environments, except for Ball1D, where it still achieves comparable results.

Constraint violations: Since the analytical form of action constraints is not available for these tasks, it is not accurate to measure pre-projection constraint violations. Therefore, we report the percentage of post-projection constraint violations, which corresponds to the percentage of episode terminations due to state-wise constraint violations. Table 1 presents the constraint violation results. In Ball1D, all algorithms show no constraint violations, as the constraints are relatively easy to handle compared to other tasks. Our approach results in fewer constraint violations across all tasks, except for Ball3D. Although NFW has fewer violations in Ball3D, it only converged to a significantly lower return.

4.3 The Effect of the Entropy Term

In Figure 6 of the supplementary, we evaluate whether the proposed entropy term in Eq. (18) and Eq. (17) has a meaningful effect on SAC+CVFlow algorithm with a Gaussian base distribution. Therefore, we show the return and constraint violation percentage of the algorithm, with and without the entropy term ($\|\hat{a}\|_2^2$). We can see that without the entropy term, the agent produces higher constraint violations (except for H+M). Additionally, without the entropy term, the agent struggles to learn in all environments.

4.4 Comparison of Flow Models

We also compare the quality of the trained normalizing flows using our proposed $\mathbb{CV}$-Flow with the method from (Brahmanage, Ling, and Kumar 2023), which relies on feasible state-action pairs. As shown in supplementary Figure 7, we evaluated accuracy, recall, and F1-score during training. The key observation is that $\mathbb{CV}$-Flow achieves higher accuracy and a better F1-score compared to normalizing flows trained on a feasible dataset. This higher accuracy allows our flow model to output feasible actions with high probability, reducing the need for QP-based action projections, which are time-consuming when integrated with ACRL methods.

5 Conclusion

We have introduced a normalizing flows-based mapping function that transforms samples from a base distribution into feasible actions in ACRL. A key advantage of our flow model is that it eliminates the need to generate feasible state-action samples from the constrained action space for training, which is often challenging. Instead, our model is trained using constraint violation signals. When integrated with SAC to address both action and state-wise constraints, our approach results in significantly fewer constraint violations without sacrificing returns, compared to previous methods. Additionally, it incurs less overhead when handling non-convex constraints.

Acknowledgments

This research/project is supported by the National Research Foundation, Singapore and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2- RP-2020-017).

References

Adawadkar, A. M. K.; and Kulkarni, N. 2022. Cyber-security and reinforcement learning — A brief survey. *Engineering Applications of Artificial Intelligence*, 114: 105116.

Amos, B.; and Kolter, J. Z. 2017. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, 136–145.

Bhatia, A.; Varakantham, P.; and Kumar, A. 2019. Resource Constrained Deep Reinforcement Learning. In *Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling*, 610–620.

Brahmanage, J. C.; Ling, J.; and Kumar, A. 2023. FlowPG: Action-constrained Policy Gradient with Normalizing Flows. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Brubaker, M.; Salzmann, M.; and Urtasun, R. 2012. A Family of MCMC Methods on Implicitly Defined Manifolds. In *International Conference on Artificial Intelligence and Statistics*, 161–172.

Changyu, C.; Karunasena, R.; Nguyen, T. H.; Sinha, A.; and Varakantham, P. 2023. Generative Modelling of Stochastic Actions with Arbitrary Constraints in Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Dalal, G.; Dvijotham, K.; Vecerik, M.; Hester, T.; Paduraru, C.; and Tassa, Y. 2018. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*.

Dinh, L.; Sohl-Dickstein, J.; and Bengio, S. 2016. Density estimation using real nvp. In *International Conference on Learning Representations*.

Fujimoto, S.; Meger, D.; and Precup, D. 2019. Off-Policy Deep Reinforcement Learning without Exploration.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. PMLR.

Kasaura, K.; Miura, S.; Kozuno, T.; Yonetani, R.; Hoshino, K.; and Hosoe, Y. 2023. Benchmarking Actor-Critic Deep Reinforcement Learning Algorithms for Robotics Control With Action Constraints. *IEEE Robotics and Automation Letters*, 8(8): 4449–4456.

Khoury, J.; and Nassar, M. 2020. A Hybrid Game Theory and Reinforcement Learning Approach for Cyber-Physical Systems Security. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 1–9. ISSN: 2374-9709.

Kingma, D. P.; and Welling, M. 2022. Auto-Encoding Variational Bayes. ArXiv:1312.6114 [cs, stat].

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

Lin, J.-L.; Hung, W.; Yang, S.-H.; Hsieh, P.-C.; and Liu, X. 2021. Escaping from zero gradient: Revisiting action-constrained reinforcement learning via Frank-Wolfe policy optimization. In *Uncertainty in Artificial Intelligence*, 397–407.

Malik, S.; Anwar, U.; Aghasi, A.; and Ahmed, A. 2021. Inverse Constrained Reinforcement Learning. In *Proceedings of the 38th International Conference on Machine Learning*, 7390–7399. PMLR. ISSN: 2640-3498.

Mazoure, B.; Doan, T.; Durand, A.; Pineau, J.; and Hjelm, R. D. 2020. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, 430–444. PMLR.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Nielsen, D.; Jaini, P.; Hoogeboom, E.; Winther, O.; and Welling, M. 2020. Survae flows: Surjections to bridge the gap between vaes and flows. *Advances in Neural Information Processing Systems*, 33: 12685–12696.

Papamakarios, G.; Nalisnick, E.; Rezende, D. J.; Mohamed, S.; and Lakshminarayanan, B. 2021. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57): 1–64.

Pham, T.-H.; De Magistris, G.; and Tachibana, R. 2018. Optlayer-practical constrained optimization for deep reinforcement learning in the real world. In *International Conference on Robotics and Automation*, 6236–6243.

Rezende, D.; and Mohamed, S. 2015. Variational inference with normalizing flows. In *International conference on machine learning*, 1530–1538. PMLR.

Thananjeyan, B.; Balakrishna, A.; Nair, S.; Luo, M.; Srinivasan, K.; Hwang, M.; Gonzalez, J. E.; Ibarz, J.; Finn, C.; and Goldberg, K. 2021. Recovery rl: Safe reinforcement learning with learned recovery zones. *IEEE Robotics and Automation Letters*, 6(3): 4915–4922.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. ISSN: 2153-0866.

Van Hasselt, H.; Guez, A.; and Silver, D. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Ward, P. N.; Smofsky, A.; and Bose, A. J. 2019. Improving exploration in soft-actor-critic with normalizing flows policies. *arXiv preprint arXiv:1906.02771*.

Zhang, J.; Zhang, C.; Wang, W.; and Jing, B.-Y. 2023. APAC: Authorized Probability-controlled Actor-Critic For Offline Reinforcement Learning.

Zhao, W.; He, T.; Chen, R.; Wei, T.; and Liu, C. 2023. State-wise Safe Reinforcement Learning: A Survey. ArXiv:2302.03122 [cs].