Escaping from Zero Gradient: Revisiting Action-Constrained Reinforcement Learning via Frank-Wolfe Policy Optimization (Supplementary material)

Jyun-Li Lin[1*] **Wei Hung**[12*] **Shang-Hsuan Yang**[1*] **Ping-Chun Hsieh**[1] **Xi Liu**[3]

[1]Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan
[2]Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
[3]Applied Machine Learning, Facebook AI, Menlo Park, CA, USA
[*]Equal Contribution

Abstract

Action-constrained reinforcement learning (RL) is a widely-used approach in various real-world applications, such as scheduling in networked systems with resource constraints and control of a robot with kinematic constraints. While the existing projection-based approaches ensure zero constraint violation, they could suffer from the zero-gradient problem due to the tight coupling of the policy gradient and the projection, which results in sample-inefficient training and slow convergence. To tackle this issue, we propose a learning algorithm that decouples the action constraints from the policy parameter update by leveraging state-wise Frank-Wolfe and a regression-based policy update scheme. Moreover, we show that the proposed algorithm enjoys convergence and policy improvement properties in the tabular case as well as generalizes the popular DDPG algorithm for action-constrained RL in the general case. Through experiments, we demonstrate that the proposed algorithm significantly outperforms the benchmark methods on a variety of control tasks.

1 INTRODUCTION

Action-constrained reinforcement learning (RL) is a popular approach for sequential decision making in real-world systems. One classic example is maximizing the network-wide utility by optimally allocating the network resource under capacity constraints (Xu et al., 2018; Gu et al., 2019; Zhang et al., 2020a). Another example is robot control under kinematic constraints (Pham et al., 2018; Gu et al., 2017; Jaillet and Porta, 2012; Tsounis et al., 2020), which capture the limitations of the physical components of a robot (e.g., in terms of velocity, torque, or output power). In these examples, the constraints essentially characterize the set of feasible actions at each state. To ensure the safe and normal operation of these real-world systems, it is required that these action constraints are satisfied throughout the evaluation as well as the training processes (Chow et al., 2018; Liu et al., 2020a; Gu et al., 2017). Therefore, in action-constrained RL, an effective training algorithm is required to achieve the following two tasks simultaneously: (i) iteratively improving the policy and (ii) ensuring zero constraint violation at each training step.

To enable RL with action constraints, one popular generic approach is to include an additional differentiable projection layer at the output of the policy network and follow the standard end-to-end policy gradient approach (Pham et al., 2018; Dalal et al., 2018; Bhatia et al., 2019). While being a general-purpose solution, this projection layer could result in the *zero-gradient issue* during training due to the tight coupling of the policy gradient update and the projection layer. Specifically, zero gradient occurs when the original output of the policy network falls outside of the feasible action set and any small perturbation of the policy parameters does not lead to any change in the final output action due to the projection mechanism. To better understand the zero-gradient issue, let us consider a toy example of a policy network with one hidden layer and a linear output layer used to produce a deterministic scalar action. Suppose the actions are required to be non-negative. To satisfy the non-negativity action constraint, an additional L_2-projection layer, which is equivalent to a Rectified Linear Unit (ReLU), is added to the output of the policy. It can be seen that the policy network can easily suffer from zero gradient due to the clipping effect of ReLU (Maas et al., 2013). If the zero-gradient issue occurs in a large portion of the state space, the training process could be sample-inefficient as most of the samples are wasted, and therefore the convergence speed could be slow. Notably, the zero-gradient issue can be particularly severe in the early training phase since the pre-projection actions produced by the policy network are likely to be far away from the feasible sets.

The fundamental cause of the zero-gradient issue is the tight coupling of the policy parameter update and the projection layer under the standard policy gradient framework. Specifically, in the end-to-end policy-gradient-based training process, the update of the policy parameters relies on the gradient of the actual policy output with respect to the policy parameters and thereby involves the gradient of this additional projection layer. To escape from the zero-gradient issue, we take a different approach and propose a learning algorithm that *decouples* the parameter update for policy improvement from constraint satisfaction, without using the policy gradient theorem. The proposed algorithm can be highlighted as follows:

- To accommodate the action constraints, we leverage the Frank-Wolfe method (Frank et al., 1956) to search for feasible action update directions directly *within* the feasible action sets in a *state-wise* manner. Through this procedure, for a collection of states, we obtain the reference actions that are used to guide the update of the policy parameters for improving the current policy.
- To update the parameters of the policy network, we propose to construct a loss function (e.g., mean squared error) that enables the policy network to adjust its outputs toward the reference actions. This update scheme can be viewed as solving a regression problem based on the reference actions by taking one-step gradient descent. In this way, the parameter update is completely decoupled from the action constraints.

Since the proposed framework obviates the need for the gradient of a projection layer, it avoids the zero-gradient issue by nature.

Our Contributions. In this paper, we revisit the action-constrained RL problem and propose a novel learning framework that avoids the zero-gradient issue and achieves zero constraint violation simultaneously:

- We formally identify the important zero gradient issue in the existing projection-based approaches for action-constrained RL. We also pinpoint that the fundamental cause of the zero-gradient issue is the tight coupling of the policy parameter update and the projection layer under the standard policy gradient framework. To the best of our knowledge, this is the first time that the zero-gradient issue is discussed in the context of action-constrained RL.
- To better describe the proposed learning framework, we start from the case of finite state spaces and introduce Frank-Wolfe policy optimization (FWPO) with tabular policy parameterization, which can be viewed as an instance of the generalized policy iteration. By directly searching for update directions within the feasible sets via state-wise Frank-Wolfe, FWPO automatically achieves zero constraint violation and does not require any additional projection. Moreover, we establish the convergence of FWPO as well as its policy improvement property.
- Built on FWPO, we propose Neural FWPO (NFWPO) by extending the idea of FWPO to the general neural policies via a regression argument. By constructing a loss function and leveraging state-wise Frank-Wolfe, we decouple the policy parameter update from the action constraints. This design automatically prevents the zero-gradient issue. Moreover, we show that the vanilla DDPG is a special case of NFWPO if there is no action constraints.
- Through experiments on various real applications, we empirically show the zero-gradient problem and demonstrate that the proposed algorithms significantly outperform the popular benchmark methods for action-constrained RL.

2 PRELIMINARIES

We consider an infinite-horizon discounted Markov decision process (MDP) defined by a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is state space, $\mathcal{A}$ denotes the action space, p is the state transition probability, r is the reward function, and $\gamma \in (0,1)$ denotes the discount factor. We assume that the action space $\mathcal{A} \subseteq \mathbb{R}^N$ is continuous and the reward function takes value in $[0,1]$ for all state-action pairs. At each time step $t = 0, 1, \cdots$, the learner observes state s_t, takes an action a_t, and receives an immediate reward r_t. In this paper, we consider the *action-constrained* MDPs where for each state $s \in \mathcal{S}$ there is a feasible action set $\mathcal{C}(s) \subseteq \mathcal{A}$ determined by the underlying collection of constraints. We assume that $\mathcal{C}(s)$ is compact and convex. In this paper, we focus on deterministic policies and use $\pi(\cdot;\theta) : \mathcal{S} \to \mathcal{A}$ to denote a deterministic parametric policy with parameter vector $\theta \in \mathbb{R}^n$. Under a policy π, the value functions are defined as the expected long-term rewards

$$V(s;\pi) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, \pi\Big], \tag{1}$$

$$Q(s,a;\pi) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a, \pi\Big]. \tag{2}$$

To make a comparison between policies, for any two policies π and π', we say that $\pi \geq \pi'$ if $V(s;\pi) \geq V(s;\pi')$, for all $s \in \mathcal{S}$. This essentially constructs a partial ordering among policies. To construct a total ordering of all the policies, consider the performance objective defined as a weighted average of the value function

$$J_\mu(\pi) := \mathbb{E}_{s\sim\mu}[V(s;\pi)], \tag{3}$$

where μ is called the restarting state distribution (Kakade and Langford, 2002). Note that one common choice of μ is the initial state distribution. It is also convenient to define the discounted state visitation distribution d^π_μ as $d^\pi_\mu(s) := (1-\gamma)\,\mathbb{E}_{s_0\sim\mu}[\sum_{t=0}^{\infty} \gamma^t P(s_t = s | s_0, \pi)]$, for each $s \in \mathcal{S}$.

Notations. We use the standard notations $\|\cdot\|_p$ and $\|\cdot\|_F$ to denote the L_p-norm of a vector and the Frobenius norm of a matrix, respectively. We use $\langle \cdot, \cdot \rangle$ to denote the inner product of two real vectors. For a set $\mathcal{D}$, we define the diameter of $\mathcal{D}$ as $\text{diam}_{\|\cdot\|_2}(\mathcal{D}) := \sup_{x_1, x_2 \in \mathcal{D}} \|x_1 - x_2\|_2$. We use $\text{dom} f$ to denote the domain of a function f.

2.1 POLICY GRADIENT

To optimize the objective $J_\mu(\pi)$, the typical approach is to apply gradient ascent based on the policy gradient. Under the standard regularity conditions, the deterministic policy gradient (Silver et al., 2014) can be written as

$$\begin{aligned} &\nabla_\theta J_\mu(\pi(\cdot;\theta)) \\ &= \mathbb{E}_{s \sim d_\mu^\pi}\Big[\nabla_\theta \pi(s;\theta) \nabla_a Q(s,a;\pi(\cdot;\theta))|_{a=\pi(s;\theta)}\Big]. \quad (4) \end{aligned}$$

As a practical implementation of the deterministic policy gradient approach, DDPG (Lillicrap et al., 2016) extends deep Q-learning (Mnih et al., 2015) to continuous action space in an actor-critic manner. Specifically, DDPG updates the policy parameter θ by applying stochastic gradient ascent according to (4) and obtains an approximated Q-function $Q(s,a;\phi)$ parameterized by ϕ by using a Q-learning-like critic, which updates ϕ by minimizing the loss $\mathbb{E}_{(s,a,s',r)\sim\rho}[(r + \gamma Q(s', \pi(s';\theta^-);\phi^-) - Q(s,a;\phi))^2]$, where ρ denotes the sampling distribution of the replay buffer, θ^- and ϕ^- are the parameters of the actor and critic target networks, respectively.

2.2 FRANK-WOLFE METHODS

In this section, we provide an overview of the Frank-Wolfe algorithms. Consider an optimization problem in the form

$$\max_{x \in \mathcal{X}} F(x), \quad (5)$$

where $F(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is a differentiable function with a Lipschitz continuous gradient, $\mathcal{X} \subseteq \mathbb{R}^d$ is the feasible set characterized by the underlying constraints on x. One popular approach is to apply the projected gradient ascent method (Bubeck et al., 2015), which combines the standard gradient ascent with a projection step. By contrast, as a projection-free method, the classic Frank-Wolfe algorithms (Frank et al., 1956) and its variants solve the constrained optimization problems in (5) by leveraging a first-order subproblem. We briefly summarize the Frank-Wolfe algorithm for non-convex objective functions in the batch settings as follows (Lacoste-Julien, 2016; Reddi et al., 2016):

- **Initialization.** Let x_k denote the input at the k-th iteration and choose an arbitrary $x_0 \in \mathcal{X}$ to be the initial point.
- **Search for an update direction within the feasible set.** In the k-th iteration, compute $v_k = \text{argmax}_{v \in \mathcal{X}} \langle v, \nabla_x F(x)|_{x=x_k}\rangle$ and update the iterate as $x_{k+1} = x_k + \beta_k(v_k - x_k)$, where $v_k - x_k$ is the update direction and β_k denotes the learning rate.

For unconstrained optimization problems, the convergence properties are typically analyzed in terms of the gradient norm $\|\nabla_x F(x)\|_2$. By contrast, for constrained maximization problems, one widely-used metric of convergence in the Frank-Wolfe literature is the *Frank-Wolfe gap* defined as $\mathcal{G}(x) := \max_{z \in \mathcal{X}} \langle z - x, \nabla_x F(x)\rangle$[1]. It is easy to verify that $\mathcal{G}(x) = 0$ is a necessary and sufficient condition of that x is a stationary point.

3 FRANK-WOLFE POLICY OPTIMIZATION

In this section, we formally present the proposed learning algorithms for action constrained RL. To better describe the proposed learning framework, we start from a stylized setting with tabular policy parameterization for finite state spaces and extend the idea to develop a more practical algorithm for the general parametric policies.

3.1 FRANK-WOLFE POLICY OPTIMIZATION WITH DIRECT POLICY PARAMETERIZATION (FWPO)

For ease of exposition, we first illustrate the proposed algorithm for the case of finite state spaces and tabular policies with direct parameterization, i.e., $\pi(s;\theta) \equiv \theta(s)$, for all $s \in \mathcal{S}$. We consider the performance objective $J_\mu(\pi)$ with some restarting state distribution μ with $\mu(s) > 0$, for all $s \in \mathcal{S}$, and define $\mu_{\min} := \min_{s \in \mathcal{S}} \mu(s)$. For ease of notation, we also define $D_s := \text{diam}_{\|\cdot\|_2}(\mathcal{C}(s))$ for each s and $D_{\max} := \max_{s \in \mathcal{S}} D_s$.

Now we present the proposed FWPO algorithm. We use θ_k to denote the policy parameters in the k-th iteration and choose feasible initial policy parameters θ_0 which satisfy $\theta_0(s) \in \mathcal{C}(s)$, for all $s \in \mathcal{S}$. FWPO adopts the generalized policy iteration framework (Sutton and Barto, 2018) by alternating between two subroutines in each iteration:

- **Policy update via state-wise Frank-Wolfe.** FWPO updates the policy by finding a feasible update direction of each state $s \in \mathcal{S}$ via Frank-Wolfe as

$$c_k(s) = \underset{c \in \mathcal{C}(s)}{\text{argmax}} \, \langle c, \nabla_a Q(s,a;\pi(\cdot;\theta_k))|_{a=\theta_k(s)}\rangle, \quad (6)$$

$$\theta_{k+1}(s) = \theta_k(s) + \alpha_k(s)(c_k(s) - \theta_k(s)), \quad (7)$$

where $c_k(s) - \theta_k(s)$ is the update direction and $\alpha_k(s)$ denotes the (state-dependent) learning rate. Moreover, it

[1]In the literature, the Frank-Wolfe gap is typically defined as $\max_{z \in \mathcal{X}} \langle z - x, -\nabla_x F(x)\rangle$ since the goal is to minimize an objective function. By contrast, as the goal of RL is to optimize the policy in terms of rewards, we consider the maximization problem in the form of (5) and make the required changes accordingly.

is natural to define the *state-wise Frank-Wolfe gap* of the Q-function at θ_k as

$$g_k(s) := \langle c_k(s) - \theta_k(s), \nabla_a Q(s, a; \pi(\cdot;\theta_k))|_{a=\theta_k(s)}\rangle. \tag{8}$$

It is easy to verify that $g_k(s) \geq 0$, for all $k \in \mathbb{N}$ and for all $s \in \mathcal{S}$. As will be shown momentarily, to ensure convergence, the learning rate is configured to be $\alpha_k(s) = \frac{(1-\gamma)\mu_{\min}}{LD_s^2} g_k(s)$.

- **Evaluation of the current policy.** FWPO then evaluates the updated policy and obtain the Q-function (or an approximated version) for the next iteration. This can be done by a standard policy evaluation approach.

The above scheme of FWPO is detailed in Algorithm 1. As suggested by Algorithm 1, FWPO always searches for an update direction within the feasible action sets. Therefore, FWPO automatically achieves zero constraint violation and does not require any additional projection by nature.

Algorithm 1 Frank-Wolfe Policy Optimization (FWPO)

```
1: Input: Initialize the policy parameters as $\theta_0$ that satisfies $\theta_0(s) \in \mathcal{C}(s)$ for all $s \in \mathcal{S}$
2: for each iteration $k = 0, 1, \cdots$ do
3:     Evaluate $\pi(\cdot;\theta_k)$ and obtain $Q(s, a; \pi(\cdot;\theta_k))$
4:     for each state $s \in \mathcal{S}$ do
5:         Compute the Frank-Wolfe update direction by $c_k(s) = \text{argmax}_{c \in \mathcal{C}(s)} \langle c, \nabla_a Q(s, a; \pi(\cdot;\theta_k))\rangle$
6:         $g_k(s) = \langle c_k(s) - \theta_k(s), \nabla_a Q(s, a; \pi(\cdot;\theta_k))\rangle$
7:         $\alpha_k(s) = \frac{(1-\gamma)\mu_{\min}}{LD_s^2} g_k(s)$
8:         $\theta_{k+1}(s) = \theta_k(s) + \alpha_k(s)(c_k(s) - \theta_k(s))$
9:     end for
10: end for
```

Remark 1 One salient feature of FWPO is that the policy update in (6)-(7) is done by searching for feasible update directions based on $\nabla_a Q(s, a; \pi)$ on a *per-state* basis with state-dependent learning rates, instead of using the standard policy gradient of the performance objective $J_\mu(\pi)$. As will be seen in Section 3.2, this design plays a critical role in decoupling the policy parameter update from constraint satisfaction. Another advantage of FWPO is that it is agnostic to the discounted state visitation distribution d_μ^π (*cf.* the deterministic policy gradient in (4)) due to the state-wise nature. This feature allows FWPO to be directly applicable in the off-policy settings in its original form[2].

Remark 2 As the policy update under FWPO is done on a state-by-state basis instead of directly on $J_\mu(\pi)$, the convergence guarantees of the standard Frank-Wolfe methods do not directly apply to the objective $J_\mu(\pi)$ under FWPO. From this perspective, FWPO is *not* a trivial combination of the Frank-Wolfe methods and policy iteration.

As suggested by Remark 2, we proceed to establish the convergence result of FWPO. For the convergence analysis, based on the state-wise Frank-Wolfe gaps defined in (8), we define the *effective Frank-Wolfe gap* of $J_\mu(\pi(\cdot;\theta))$ at θ_k as

$$\mathcal{G}_k := \Big(\sum_{s\in\mathcal{S}} g_k(s)^2\Big)^{1/2}. \tag{9}$$

Note that $\mathcal{G}_k = 0$ if and only if the update direction is zero for all the states, i.e., $c_k(s) - \theta_k(s) = 0$. Hence, $\mathcal{G}_k$ indicates whether the $J_\mu(\pi(\cdot;\theta))$ converges to a stationary point. We also define $\bar{\mathcal{G}}_T := \min_{0\leq k\leq T} \mathcal{G}_k$. To establish the convergence results, we also assume mild regularity conditions on r and p as follows.

Definition 1 A differentiable function $f : \text{dom} f \to \mathbb{R}$ is said to be L_0-*smooth* if there exists $L_0 \geq 0$ such that for any $x, y \in \text{dom} f$, $\|\nabla f(x) - \nabla f(y)\|_2 \leq L_0 \|x - y\|_2$.

Regularity Assumptions:

(A1) The reward function $r(s, a)$ is differentiable and is L_r-smooth in a, for all s, a.

(A2) The transition probability $p(s'|s, a)$ is twice differentiable and L_p-smooth in a, for all s, s', a. Moreover, $p(s'|s, a)$ satisfies $\sup_{s,a,s'} \|\nabla_a p(s'|s, a)\|_2 < C_p$.

As the first step, we introduce the following proposition on the smoothness of the performance objective $J_\mu(\pi(\cdot;\theta))$. Notably, given the regularity assumptions of r and p in action, it remains non-trivial to establish the smoothness of $J_\mu(\pi(\cdot;\theta))$ in θ due to the multi-step compound effect of the changes in policy parameters on the value functions.

Proposition 1 *Under the regularity assumptions (A1)-(A2), there exists some constant $L > 0$ such that for any restarting state distribution μ, $J_\mu(\pi(\cdot;\theta))$ is L-smooth in θ.*

The proof of Proposition 1 is provided in Appendix A.1. Now we are ready to present the convergence result.

Proposition 2 *Under the FWPO algorithm with $\alpha_k(s) = \frac{(1-\gamma)\mu_{\min}}{LD_s^2} g_k(s)$, $\{\pi(\cdot;\theta_k)\}$ form a non-decreasing sequence of policies in the sense that $\pi(\cdot;\theta_{k+1}) \geq \pi(\cdot;\theta_k)$, for all k. Moreover, the effective Frank-Wolfe gap of FWPO converges to zero as $k \to \infty$, and the convergence rate can be quantified as*

$$\sum_{k=0}^{\infty} \mathcal{G}_k^2 \leq \frac{2LD_{\max}^2}{(1-\gamma)^3 \mu_{\min}^2}, \tag{10}$$

which implies that $\bar{\mathcal{G}}_T = O(T^{-1/2})$.

[2]In the off-policy settings, the deep policy gradient approaches typically require dropping a term in the policy gradient expression to accommodate the behavior policy (Silver et al., 2014).

Proof Due to space limitation, we provide a sketch of proof: (i) To show the non-decreasing property, we leverage the policy difference lemma (Kakade and Langford, 2002) and verify a sufficient condition of strict policy improvement; (ii) To show the convergence result, we leverage the smoothness of the value functions as well as the objective and use the technique for convergence of non-convex optimization similar to that in (Reddi et al., 2016; Lacoste-Julien, 2016); (iii) A proper learning rate can be selected by taking the smoothness conditions as well as the restarting state distribution into account. For completeness, the detailed proof is provided in Appendix A.2.

Remark 3 The style of the convergence guarantee in Proposition 2 is common in the analysis of gradient descent methods for non-convex smooth functions (Bottou et al., 2018). Moreover, the result (i.e., convergence to a stationary point) in Proposition 2 resembles those of the policy gradient algorithms (Sutton, 2000; Silver et al., 2014), but for the action-constrained RL settings. On the other hand, in (6), the search of the update direction requires the gradient of the Q-function. In practice, it may not be feasible to obtain the whole true Q-function, and a value function approximator can be included. In practice, it can be expected that a sufficiently accurate critic shall provide a sufficiently good update direction.

3.2 NEURAL FRANK-WOLFE POLICY OPTIMIZATION (NFWPO)

In this section, we formally present the proposed NFWPO algorithm for general parametric policies for action-constrained RL. As highlighted in Section 1, we propose to decouple constraint satisfaction from the policy parameter update. Specifically, to accommodate the action constraints, we extend the state-wise Frank-Wolfe subroutine to the general parametric policies. One inherent challenge of such extension is that the Frank-Wolfe method searches for an update direction within the feasible set by nature. However, under neural parameterization, an action produced by the neural network is not guaranteed to stay in the feasible action set. To address this, we propose to incorporate a projection step into the state-wise Frank-Wolfe subroutine. Define a projection operator as

$$\Pi_{\mathcal{C}(s)}(z) = \underset{y\in\mathcal{C}(s)}{\operatorname{argmin}} \|y-z\|_2. \tag{11}$$

For ease of exposition, in the sequel we call the input z a *pre-projection action* and $\Pi_{\mathcal{C}(s)}(z)$ a *post-projection action*.

NFWPO adopts the actor-critic architecture. Let $\bar{\theta}$ and $\bar{\phi}$ be the current parameters of the actor and the critic, respectively. The main features of NFWPO are captured by the actor part as below.

- **Derive reference actions via state-wise Frank-Wolfe.** For each s in the mini-batch $\mathcal{B}$, NFWPO uses Frank-Wolfe to compute the reference action at each state s as

$$\tilde{a}_s = \Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta})) + \alpha\big(\bar{c}(s) - \Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta}))\big), \tag{12}$$

where α is the learning rate of Frank-Wolfe and

$$\bar{c}(s) = \underset{c\in\mathcal{C}(s)}{\operatorname{argmax}} \langle c, \nabla_a Q(s,a;\bar{\phi})|_{a=\Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta}))}\rangle. \tag{13}$$

(Note that the projection $\Pi_{\mathcal{C}(s)}(\cdot)$ is only for generating feasible actions and does not require backpropagation.)

- **Construct an MSE loss function.** NFWPO constructs a loss function $L_{\text{NFWPO}}(\theta;\bar{\theta})$ as the MSE between the actions of the current policy and the reference actions, i.e.,

$$\mathcal{L}_{\text{NFWPO}}(\theta;\bar{\theta}) = \sum_{s\in\mathcal{B}} \big(\pi(s;\theta) - \tilde{a}_s\big)^2. \tag{14}$$

- **Update policy by gradient descent.** NFWPO updates the policy parameter by minimizing the MSE loss in (14) by using gradient descent for one step, i.e.,

$$\theta \leftarrow \theta - \beta\nabla_\theta \mathcal{L}_{\text{NFWPO}}(\theta;\bar{\theta}). \tag{15}$$

On the other hand, the critic of NFWPO can be based on any standard policy evaluation technique. For ease of exposition, for NFWPO, we use the same critic as the vanilla DDPG (as described in Section 2.1). The detailed pseudo code of NFWPO is provided in the supplementary material.

Notably, similar to (6), NFWPO only uses $\nabla_a Q(s,a;\bar{\phi})$ for deriving reference actions, *without* using the deterministic policy gradient in (4). This design allows NFWPO to decouple constraint satisfaction in (12)-(13) from the parameter update in (14)-(15). As highlighed in Section 1, this decoupling obviates the need for the gradient of a projection layer and hence automatically avoids the zero-gradient issue. Moreover, below we show that DDPG is actually a special case of NFWPO when there is no action constraints. The proof is provided in Appendix B

Proposition 3 *If there is no action constraints, then the policy update scheme of NFWPO in (12)-(15) is equivalent to the vanilla DDPG by (Lillicrap et al., 2016).*

Remark 4 While NFWPO leverages a projection step in (12), this projection step is only for deriving reference actions and does not take part in the policy parameter update. As a result, NFWPO does not require backpropagation of the projection step (as shown in (12)-(15)) and therefore automatically avoids the zero-gradient issue. Hence, NFWPO is essentially different from the existing solutions that combine DDPG with a projection layer for end-to-end training (Pham et al., 2018; Dalal et al., 2018).

4 EXPERIMENTAL RESULTS

In this section, we empirically evaluate FWPO and NFWPO in various real-world applications, including bike sharing systems, communication networks, and continuous control in MuJoCo. We compare the proposed algorithms against the following popular benchmark methods:

- **DDPG+Projection**: The training procedure is identical to the vanilla DDPG (Lillicrap et al., 2016) except that the action is post-processed by the L_2-projection operator $\Pi_{\mathcal{C}(s)}(\cdot)$ before being applied to the environment.
- **DDPG+RewardShaping**: Built on DDPG+Projection, this algorithm adds the L_2-norm between the pre-projection and post-projection actions as a penalty to the intrinsic reward.
- **DDPG+OptLayer**: This design uses a differentiable projection layer, namely the OptLayer, that supports end-to-end training via gradient descent (Pham et al., 2018).

Moreover, for the projection step (without the need of backpropagation) required by DDPG+Projection, DDPG+RewardShaping, and NFWPO, we implement this functionality on the Gurobi optimization solver (Gurobi Optimization, 2021). Therefore, the post-projection actions are guaranteed to satisfy the action constraints for all the algorithms. For each task, each algorithm is trained under the common set of 5 random seeds. Each evaluation consists of 10 episodes, and we report the average performance along with the standard deviation in Figures 1-5. We also summarize the average return over the final 10 evaluations in Table 1 and Table 2. The detailed training setup can be found in Appendix D. The code of our experiments is available [3].

4.1 BIKE SHARING SYSTEMS

We use the open-source BSS simulator[4], which was originally proposed by (Ghosh and Varakantham, 2017) and later used for evaluating action-constrained RL by (Bhatia et al., 2019). In a bike-sharing problem, there are m bikes and n stations, each of which has a pre-determined bike storage capacity C. An action is to allocate m bikes to n stations under random demands. The reward signal consists of three parts: (i) Moving cost: the cost of moving one or multiple bikes from one station to another; (ii) Lost-demand cost: the cost of unserved demand due to bike outage. (iii) Overflow cost: the cost incurred when the number of bikes in one station exceeds its capacity.

Evaluating FWPO. Since the bike sharing environment has a finite state space, we first use it to evaluate FWPO against the baseline methods, all with tabular policy parameterization. For the action value function, we use the same Q-learning-like critic as the vanilla DDPG for all the algorithms. A medium-sized system with $n = 3$, $m = 90$, and $C = 35$ is chosen that allows to analytically find the optimal policy. There are two types of constraints: (i) Global constraint: all the action entries shall sum to 90; (ii) Local constraints: each entry of the action shall be between 0 and 35. Figure 1(a) shows the average return of the three algorithms. We observe that FWPO performs the best, while DDPG+Projection and DDPG+RewardShaping both suffer from slow learning. This is also reflected by Figure 1(b), which shows that FWPO converges to a near-optimal policy much faster than the baselines. The above phenomenon is mainly due to the inaccurate policy gradient of DDPG under action constraints. Specifically, the critics of the two baselines are trained with samples with feasible actions while the gradients $\nabla_a Q(s, a; \phi)$ are mostly evaluated at those actions outside the feasible sets. By contrast, FWPO always stays in the feasible action sets and hence naturally avoids the issue of inaccurate gradients.

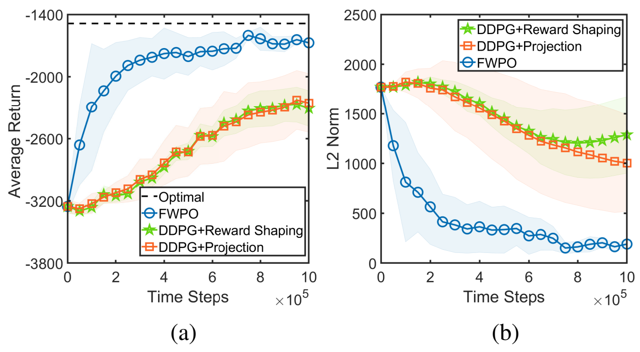


Figure 1: Bike sharing problem with $n = 3$ (BSS-3) under tabular policies: (a) Average return over 5 random seeds; (b) L_2-norm between the learned policies and the optimal policy at each training step.

Evaluating NFWPO. We proceed to compare NFWPO with the other three baselines in solving a larger-scale bike-sharing problem with $m = 150$, $n = 5$, and $C = 35$. As shown by Figure 2(a), NFWPO converges faster and achieves a larger return than the other baselines. To better understand its behavior, Figure 2(b) shows the cumulative constraint violations of the *pre-projection* actions. Interestingly, the pre-projection actions of NFWPO can largely avoid constraint violation, and thus requires less help from the projection during training. By contrast, all the baselines rely heavily on the projection step to stay feasible, because most of their pre-projection actions fail to satisfy the constraints. We also observe that DDPG+Projection and DDPG+RewardShaping attain similar average return and frequency of violation. This is because they both produce pre-projection actions far from the feasible sets and thereby obtain similar post-projection actions. Meanwhile, DDPG+OptLayer suffers from nearly zero learning progress

[3] https://github.com/upupsheep/NFWPO_Final_Code
[4] BSS: https://github.com/bhatiaabhinav/gym-BSS

due to the zero-gradient issue. Figure 2(c) compares the sample-based gradient with respect to the pre and post-OptLayer actions. Since the gradients of the pre-OptLayer actions (green line in Figure 2(c)) are mostly close to zero, the sample-based policy gradients $\nabla_\theta \hat{J}_\mu(\pi(\cdot;\theta))$ are therefore close to zero for most of the training steps. As the gradients with respect to the post-OptLayer actions are always *non-zero* (blue dotted line in Figure 2(c)), we know that the zero-gradient issue of $\nabla_\theta \hat{J}_\mu(\pi(\cdot;\theta))$ indeed results from the projection layer. This confirms that the additional OptLayer could easily lead to the zero-gradient issue and sample-inefficient training.

4.2 UTILITY MAXIMIZATION OF COMMUNICATION NETWORKS

In this section, we evaluate the proposed methods over the task of utility maximization in communication networks. We simulate the network with the open-source network simulator from PCC-RL[5] (Jay et al., 2019). For the network topology, we consider the classic T3 NSFNET Backbone and set the bandwidth of each link to be 50 packets per second throughout the experiments. We generate three network flows, each of which has three candidate paths from its source to the destination. The action is to determine the rate allocation of each flow along each candidate path. The reward consists of three parts: (i) Throughput: the number of received packets per second; (ii) Drop rate: the number of dropped packets per second; (iii) Latency: the average latency of the packets in the last second. For each flow i, its immediate reward is $\log\left(\frac{\text{throughput}_i}{\text{drop rate}_i^{0.5} \times \text{latency}_i^{1.5}}\right)$, which corresponds to the widely-used proportional fairness criteria (Kelly, 1997). One salient feature of a communication network is that when the total packet arrival rate of a link approaches its bandwidth, the latency will grow rapidly, and accordingly most of the packets would be dropped. Therefore, in this environment, the action constraints correspond to the link bandwidth constraints, i.e., the total assigned packet arrival rate of each link should be bounded by 50.

Figure 3(a) shows the training curves and indicates that NFWPO still converges fast (in about 10^5 steps) and achieves much larger return than the baselines. Moreover, similar to the bike-sharing problems, we see from Figure 3(b) that most of the pre-projection actions of NFWPO already satisfy the constraints. In this task, we find that reward shaping does help in guiding the pre-projection actions towards the feasible action sets, but only under some random seeds (and therefore the large variance in Figures 3(a)-(b)). Regarding DDPG+OptLayer, in the initial training phase, we observe that it mostly produces pre-OptLayer actions with small flow rates, which lead to a smaller number of constraint violations and moderate returns. To achieve a higher return, DDPG+OptLayer then gradually increases the flow rates but accidentally causes more constraint violations of pre-OptLayer actions and suffers from the inaccurate gradient issue described in Section 4.1. Ultimately, DDPG+OptLayer can only achieve a fairly low return.

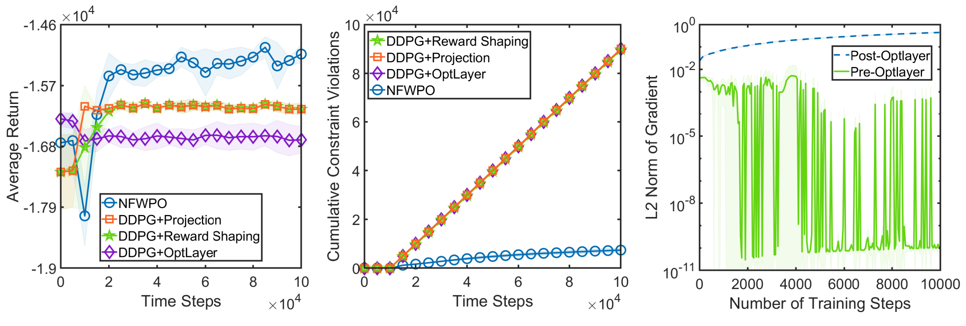
[5]PCC-RL: https://github.com/PCCproject/PCC-RL

4.3 MUJOCO CONTINUOUS CONTROL TASKS

To further validate NFWPO, we consider popular continuous control tasks in MuJoCo (Todorov et al., 2012) with non-linear and state-dependent action constraints. We further compare the proposed algorithm with important benchmarks RL algorithms for MuJoCo control taks, including PPO (Schulman et al., 2017), TRPO (Schulman et al., 2015), and SAC (Haarnoja et al., 2018). To make the comparison even more comprehensive, we also evaluate FOCOPS (Zhang et al., 2020b), which is a recent approach designed to address long-term total discounted cost constraints. As FOCOPS is not designed for handling state-wise action constraints, we relax the action constraints into the long-term total discounted constraints required by FOCOPS. To ensure constraint satisfaction, we use the same technique as DDPG+Projection, i.e., the actions of PPO, TRPO, SAC, and FOCOPS are post-processed by L_2-projection before being applied to the environment.

Reacher with non-linear constraints. In this task, the action space is 2-dimensional (denoted by u_1, u_2), and each action entry corresponds to the torque of a joint of a 2-DoF robot. To validate the applicability of NFWPO, we impose nonlinear action constraints as: ${u_1}^2 + {u_2}^2 \leq 0.05$. From Figure 4(a), we observe a similar trend that NFWPO still converges faster and achieves a larger return than the other baseline methods. In this task, DDPG+OptLayer and DDPG+RewardShaping can achieve a return closer to NFWPO as it has fewer pre-OptLayer (pre-projection) violations as shown in Figure 4(b). The other algorithms still perform poorly as they always produce actions far from the feasible sets and relies heavily on the projection step. This is reflected by the fact that SAC, TRPO, and PPO violate the constraint at almost every training step, as shown in Figure 4(b). Regarding FOCOPS, the violation of the constraint in the second half of training is less than that in the first half. This indicates that FOCOPS needs much more training steps to find a policy that violate lesser, Despite this, the return of FOCOPS remains fairly low as other benchmarks.

Halfcheetah with state-dependent constraints. In this task, an action is 6-dimensional and is denoted by $(v_1, \cdots, v_6)$. We consider a challenging scenario where the constraint is state-dependent. Specifically, the imposed constraint is $\sum_{i=1}^{6} |v_i w_i| \leq 20$, where w_i denotes the angular velocity of the i-th joint and is part of the state. This constraint is meant to capture the limitation of total output power. Similar to the other environments, from Figure 5(a)-(b), we still observe that NFWPO achieves better sample efficiency than the other baselines. Moreover, NFWPO vio-



(a) (b) (c)

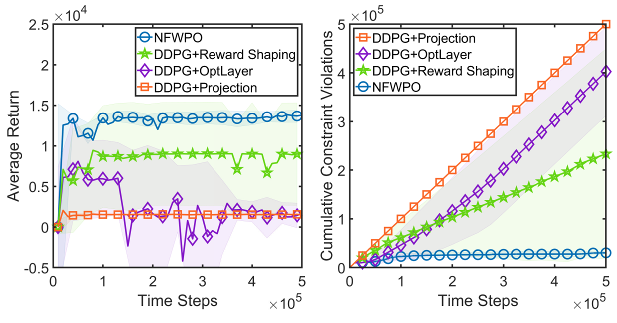
Figure 2: Bike sharing problem with $n = 5$ (BSS-5) under neural policies: (a) Average return over 5 random seeds; (b) Cumulative number of constraint violations of the pre-projection actions during training; (c) L_2-norm of the sample-based gradients with respect to the pre- and post-OptLayer actions of DDPG+OptLayer.

Table 1: Average return over the final 10 evaluations.

Methods	BSS-3	BSS-5	NSFNET
NFWPO	**-1673.04**	**-15132.21**	**13770.67**
DDPG+Projection	-2254.52	-16123.48	1514.44
DDPG+Reward Shaping	-2308.00	-16123.48	9010.46
DDPG+OptLayer	-	-16686.04	1667.59

Table 2: Average return over the final 10 evaluations in the MuJoCo environments.

Methods	Reacher	Halfcheetah
NFWPO	**-4.76**	**6513.26**
DDPG+Projection	-11.15	2746.72
DDPG+Reward Shaping	-8.66	3065.37
DDPG+OptLayer	-7.25	1399.37
SAC+Projection	-10.50	4874.45
TRPO+Projection	-11.04	2247.82
PPO+Projection	-10.68	1459.04
FOCOPS+Projection	-12.23	1916.46



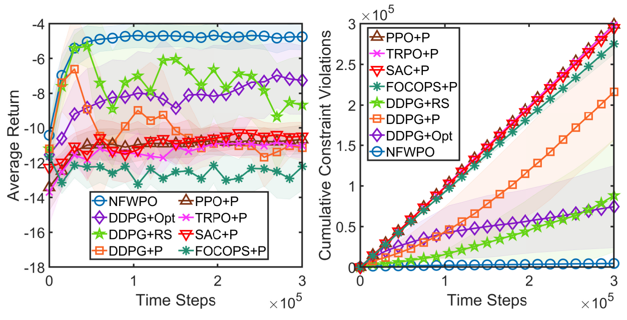
(a) (b)

Figure 3: Utility maximization in NSFNET: (a) Average return over 5 random seeds; (b) Cumulative constraint violations of the pre-projection actions during training.

lates the constraint for only 3% of the steps while getting the highest return. On the other hand, PPO, TRPO, and SAC all violate the constraint for more than 75% of the steps. FOCOPS violates the constraint for about 15% of the time but only achieves a fairly low return. Again, we see that NFWPO outperforms all the baseline methods with much less constraint violation.

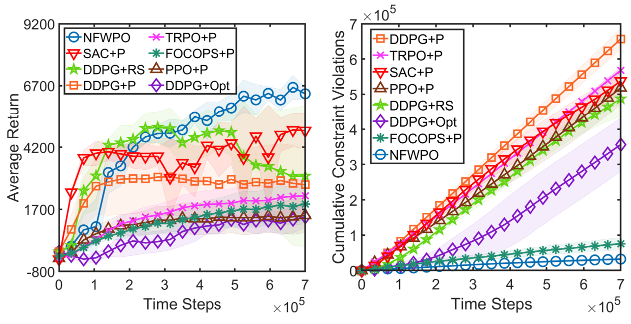
5 RELATED WORK

The constrained RL problems have been extensively studied from two main perspectives. The first category encodes the constraints via cost signals, which are incurred at each step along with the reward signals, and accordingly focuses on the average-cost constraints. This line of research works borrows a variety of ideas from the optimization literature. For example, (Chow et al., 2015) addressed chance



(a) (b)

Figure 4: Reacher-v2 with a non-linear action constraint: (a) Average return over 5 random seeds; (b) Cumulative constraint violations of the pre-projection actions during training ("+P" stands for "+Projection", "+RS" stands for "Reward Shaping", and "+Opt" stands for "+OptLayer").



(a) (b)

Figure 5: Halfcheetah-v2 with a state-dependent action constraint: (a) Average return over 5 random seeds; (b) Cumulative constraint violations of the pre-projection actions during training ("+P" stands for "+Projection", "+RS" stands for "Reward Shaping", and "+Opt" stands for "+OptLayer").

constraints by using a primal-dual approach to achieve a trade-off between return and risk. Similarly, (Tessler et al., 2018) proposed Reward Constrained Policy Optimization which applied Lagrangian relaxation and converted the constraints into penalty terms in the objective. (Achiam et al., 2017) proposed Constrained Policy Optimization to achieve strict policy improvement under the average cost constraints by using the trust-region approach. In (Chow et al., 2018), Lyapunov-based safe reinforcement learning was proposed to address the constraints by solving a linear program. (Yang et al., 2019) proposed Projection-Based Constrained Policy Optimization to achieve no constraint violation by taking a projection step after the local reward improvement update. In (Liu et al., 2020b), Interior-point Policy Optimization was proposed to handle the average cost constraints by augmenting the objective with logarithmic barrier functions. (Satija et al., 2020) took a different approach by converting the trajectory-level constraints into per-step state-wise constraints and accordingly defining a safe policy improvement step. (Zhang et al., 2020b) proposed an approach to solving the constrained policy optimization problem by first finding a target policy directly in the policy space and thereafter converting the target policy to a parameterized one through a projection step onto the parameter space. Different from all the above prior works, this paper considers the RL settings with state-wise action constraints.

The second category is on the state-wise constraints that need to be satisfied on a step-by-step basis. (Pham et al., 2018) studied the state-wise action constraints of robotic systems and proposed a projection-based OptLayer to enforce the constraints. (Dalal et al., 2018) also considered state-wise safety constraints under linearization and proposed a projection-based safety layer to handle the constraints. Similarly, (Bhatia et al., 2019) considered resource constraints and proposed variants of OptLayer to improve the computational efficiency. (Shah et al., 2020) proposed a more efficient projection scheme for enforcing linear action constraints. Despite the similarity in the problem setting, we take a different approach and propose a decoupling framework by leveraging Frank-Wolfe to address the action constraints and completely avoid the zero-gradient issue.

6 CONCLUSION

This paper revisits action-constrained RL to tackle the zero-gradient issue and ensure zero constraint violation simultaneously. We achieve this goal by developing a learning framework that decouples the policy parameter update from constraint satisfaction by leveraging state-wise Frank-Wolfe and a regression argument. Our theoretical and experimental results demonstrate that the proposed learning algorithm is indeed a promising approach for action-constrained RL.

Acknowledgements

This material is based upon work partially supported by the Ministry of Science and Technology of Taiwan under Contract No. MOST 108-2636-E-009-014 and Contract No. MOST 109-2636-E-009-012.

References

Joshua Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31. PMLR, 2017.

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *arXiv:1908.00261*, 2019.

Alekh Agarwal, Sham M Kakade, Jason D Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in Markov decision processes. In *Conference on Learning Theory*, pages 64–66, 2020.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

Abhinav Bhatia, Pradeep Varakantham, and Akshat Kumar. Resource constrained deep reinforcement learning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pages 610–620, 2019.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Sébastien Bubeck et al. Convex Optimization: Algorithms and Complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

Yinlam Chow, Aviv Tamar, Shie Mannor, and Marco Pavone. Risk-sensitive and robust decision-making: a CVaR optimization approach. In *Advances in Neural Information Processing Systems*, pages 1522–1530, 2015.

Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8103–8112, 2018.

Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv:1801.08757*, 2018.

Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596, 2018.

Yasuhiro Fujita and Shin-ichi Maeda. Clipped action policy gradient. In *International Conference on Machine Learning*, pages 1597–1606, 2018.

Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. Chainerrl: A deep reinforcement learning library. *Journal of Machine Learning Research*, 22(77):1–14, 2021. URL `http://jmlr.org/papers/v22/20-376.html`.

Supriyo Ghosh and Pradeep Varakantham. Incentivizing the use of bike trailers for dynamic repositioning in bike sharing systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 27, 2017.

Lin Gu, Deze Zeng, Wei Li, Song Guo, Albert Y Zomaya, and Hai Jin. Intelligent VNF orchestration and flow scheduling via model-assisted deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 38(2):279–291, 2019.

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396, 2017.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2021. URL `http://www.gurobi.com`.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR, 2018.

Léonard Jaillet and Josep M Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Transactions on Robotics*, 29(1):105–117, 2012.

Nathan Jay, Noga H. Rotman, P. Brighten Godfrey, Michael Schapira, and Aviv Tamar. Internet congestion control via deep reinforcement learning, 2019.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

Frank Kelly. Charging and rate control for elastic traffic. *European transactions on Telecommunications*, 8(1):33–37, 1997.

Simon Lacoste-Julien. Convergence rate of Frank-Wolfe for non-convex objectives. *arXiv:1607.00345*, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

Anqi Liu, Guanya Shi, Soon-Jo Chung, Anima Anandkumar, and Yisong Yue. Robust regression for safe exploration in control. In *Learning for Dynamics and Control*, pages 608–619. PMLR, 2020a.

Yongshuai Liu, Jiaxin Ding, and Xin Liu. IPO: Interior-point policy optimization under constraints. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 4940–4947, 2020b.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. Optlayer - practical constrained optimization for deep reinforcement learning in the real world. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6236–6243, 2018.

Sashank J Reddi, Suvrit Sra, Barnabás Póczos, and Alex Smola. Stochastic frank-wolfe methods for nonconvex optimization. In *Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1244–1251, 2016.

Harsh Satija, Philip Amortila, and Joelle Pineau. Constrained markov decision processes via backward value functions. In *International Conference on Machine Learning*, pages 8502–8511. PMLR, 2020.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Sanket Shah, Sinha Arunesh, Varakantham Pradeep, Perrault Andrew, and Tambe Milind. Solving online threat screening games using constrained action space reinforcement learning. In *AAAI Conference on Artificial Intelligence*, volume 34, pages 2226–2235, 2020.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395, 2014.

Richard S Sutton. Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063, 2000.

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

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

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Vassilios Tsounis, Mitja Alge, Joonho Lee, Farbod Farshidian, and Marco Hutter. Deepgait: Planning and control of quadrupedal gaits using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 5(2):3699–3706, 2020.

Zhiyuan Xu, Jian Tang, Jingsong Meng, Weiyi Zhang, Yanzhi Wang, Chi Harold Liu, and Dejun Yang. Experience-driven networking: A deep reinforcement learning based approach. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1871–1879, 2018.

Tsung-Yen Yang, Justinian Rosca, Karthik Narasimhan, and Peter J Ramadge. Projection-based constrained policy optimization. In *International Conference on Learning Representations*, 2019.

Junjie Zhang, Minghao Ye, Zehua Guo, Chen-Yu Yen, and H Jonathan Chao. CFR-RL: Traffic engineering with reinforcement learning in SDN. *IEEE Journal on Selected Areas in Communications*, 38(10):2249–2259, 2020a.

Yiming Zhang, Quan Vuong, and Keith Ross. First order constrained optimization in policy space. *Advances in Neural Information Processing Systems*, 2020b.

A PROOFS OF THE THEORETICAL RESULTS

In this section, we provide the proof of the convergence result of the tabular FWPO in Proposition 2 by leveraging the proof technique of smooth optimization. We start by establishing the key smoothness result of Proposition 1 and then proceed to the convergence analysis.

A.1 SMOOTHNESS RESULTS AND PROOF OF PROPOSITION 1

In this section, we show the smoothness results of the value functions as well as the performance objective under the regularity assumptions (A1)-(A2). Notably, given the regularity conditions of r and p in action, it remains non-trivial to establish the smoothness of $V(\cdot;\theta)$ and $J_\mu(\pi(\cdot;\theta))$ in θ due to the multi-step compound effect of the changes in policy parameters on the value functions. Despite this challenge, we are still able to characterize the required smoothness conditions.

Recall from Definition 1 that a differentiable function $f : \text{dom}f \to \mathbb{R}$ is L_0-smooth if there exists $L_0 \geq 0$ such that for any $x, y \in \text{dom}f$, we have $\|\nabla f(x) - \nabla f(y)\|_2 \leq L_0\|x-y\|_2$. One useful property is that if f is L_0-smooth, then f satisfies

$$|f(y) - f(x) - \langle \nabla f(x), y-x\rangle| \leq \frac{L_0}{2}\|y-x\|_2^2, \;\; \forall x, y \in \text{dom}f. \tag{16}$$

For notational convenience, we explicitly number the state space as $\mathcal{S} = \{1, \cdots, M\}$. By slightly abusing the notation, we use π to denote the $M \times N$ matrix where the s-th row represents the action selected by a deterministic policy π at each state $s \in \{1, \cdots, M\}$. Let $\mathbf{P}^\pi$ denote a $M \times M$ matrix where the (i,j)-th entry is $p(j|i, \pi(i))$. Given a deterministic policy π, consider a "perturbed" version of the policy defined by $\pi_\delta = \pi + \delta\,\mathbf{W}$, where $\delta \in \mathbb{R}$ and $\mathbf{W} \in \mathbb{R}^{M\times N}$ is some fixed matrix with $\|\mathbf{W}\|_F = 1$. Moreover, we simplify the notations by letting $\mathbf{P}(\delta) \equiv \mathbf{P}^{\pi_\delta}$ and $V_s(\delta) \equiv V(s;\pi_\delta)$ and then define $\mathbf{G}(\delta) := (\mathbf{I}_M - \gamma\,\mathbf{P}(\delta))^{-1}$, where $\mathbf{I}_M$ denotes the identity matrix of size $M \times M$. To show the smoothness result of $V(s;\theta)$, it is sufficient to show that $|\frac{d^2 V_s(\delta)}{d\delta^2}|$ is bounded for any $\mathbf{W} \in \mathbb{R}^{M\times N}$ with $\|\mathbf{W}\|_F = 1$, for any state s. We first introduce several useful lemmas.

Lemma 1 (Matrix-by-scalar derivatives)

$$\frac{d\,\mathbf{G}(\delta)}{d\delta} = \gamma\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta), \tag{17}$$

$$\frac{d^2\,\mathbf{G}(\delta)}{d\delta^2} = 2\gamma\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta) + \mathbf{G}(\delta)\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}\,\mathbf{G}(\delta). \tag{18}$$

Proof (Lemma 1) By the definition of $\mathbf{G}(\delta)$, we know

$$\mathbf{I}_M = \mathbf{G}(\delta)(\mathbf{I}_M - \gamma\,\mathbf{P}(\delta)) = \mathbf{G}(\delta) - \gamma\,\mathbf{G}(\delta)\,\mathbf{P}(\delta). \tag{19}$$

Note that the product rule of matrix-by-scalar derivative suggests that for two matrices $\mathbf{U}_1(\delta), \mathbf{U}_2(\delta)$, we have $\frac{d}{d\delta}(\mathbf{U}_1\,\mathbf{U}_2) = \mathbf{U}_1\,\frac{d\,\mathbf{U}_2}{d\delta} + \frac{d\,\mathbf{U}_1}{d\delta}\,\mathbf{U}_2$. By taking the matrix-by-scalar derivative with respect to δ on both sides of (19) and using the product rule, we have

$$\frac{d\,\mathbf{G}(\delta)}{d\delta} - \Big(\gamma\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta} + \gamma\frac{d\,\mathbf{G}(\delta)}{d\delta}\,\mathbf{P}(\delta)\Big) = \mathbf{0}_M, \tag{20}$$

where $\mathbf{0}_M$ denotes the $M \times M$ zero matrix. By reorganizing the terms in (20), it is straightforward to verify that (17) holds. Based on (17), we can obtain the second-order derivative of $\mathbf{G}(\delta)$ as

$$\frac{d^2\,\mathbf{G}(\delta)}{d\delta^2} = \frac{d}{d\delta}\Big(\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\Big) \tag{21}$$

$$= \mathbf{G}(\delta)\frac{d}{d\delta}\Big(\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\Big) + \frac{d\,\mathbf{G}(\delta)}{d\delta}\Big(\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\Big) \tag{22}$$

$$= 2\gamma\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta) + \mathbf{G}(\delta)\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}\,\mathbf{G}(\delta). \tag{23}$$

Hence, we conclude that (17) and (18) indeed hold. □

Lemma 2 *For any $x \in \mathbb{R}^M$, $\mathbf{G}(\delta)$ satisfies that*

$$\|\mathbf{G}(\delta)x\|_\infty \leq \frac{1}{1-\gamma}\|x\|_\infty. \tag{24}$$

Proof (Lemma 2) Note that $\mathbf{G}(\delta) = (\mathbf{I}_n - \gamma\,\mathbf{P}(\delta))^{-1} = \sum_{m=0}^{\infty} \gamma^m\,\mathbf{P}(z)^m$. Given that $\mathbf{P}(\delta)$ is a stochastic matrix where all the elements are non-negative and each row sums to 1, we know $\mathbf{P}(\delta)^m$ is also a stochastic matrix, for any $m \in \mathbb{N}$. For each i, let e_i denote an n-dimensional one-hot vector with the i-th entry equal to 1. Then, it is easy to verify that $\|\mathbf{G}(\delta)e_i\|_\infty \leq \frac{1}{1-\gamma}$, for any $i \in \{1, \cdots, M\}$. This thereby implies that $\|\mathbf{G}(\delta)x\|_\infty \leq \frac{1}{1-\gamma}\|x\|_\infty$, for any $x \in \mathbb{R}^M$. □

Lemma 3 *Under the regularity assumptions (A1)-(A2), there exist constants $C_1, C_2 > 0$ such that for any $\mathbf{W} \in \mathbb{R}^{M\times N}$ with $\|\mathbf{W}\|_F = 1$ and for any $x \in \mathbb{R}^M$, we have*

$$\left\| \frac{d\,\mathbf{P}(\delta)}{d\delta} x \right\|_\infty < C_1 \|x\|_\infty, \tag{25}$$

$$\left\| \frac{d^2\,\mathbf{P}(\delta)}{d\delta^2} x \right\|_\infty < C_2 \|x\|_\infty. \tag{26}$$

Proof (Lemma 3) For convenience, we use $\mathbf{W}_i$ to denote the i-th row of $\mathbf{W}$. For any pair of $i, j \in \{1, \cdots, M\}$, the (i,j)-th element of $\frac{d\,\mathbf{P}(\delta)}{d\delta}$ evaluated at $\delta = 0$ satisfies

$$\begin{aligned}
\left| \left[\frac{d\,\mathbf{P}(\delta)}{d\delta}\Big|_{\delta=0} \right]_{(i,j)} \right| &= \left| \lim_{h\to 0} \frac{p(j|i, \pi_h(i)) - p(j|i, \pi(i))}{h} \right| &(27)\\
&= \left| \langle \nabla_a p(j|i,a)\big|_{a=\pi(i)}, \mathbf{W}_i^\top \rangle \right| &(28)\\
&\leq \left\| \nabla_a p(j|i,a)\big|_{a=\pi(i)} \right\|_2 \cdot \|\mathbf{W}_i\|_2 &(29)\\
&< C_p, &(30)
\end{aligned}$$

where (28) follows from the differentiability of p and the property of directional derivatives, (29) holds by the Cauchy-Schwarz inequality, and (30) follows from the regularity assumptions and that $\|\mathbf{W}\|_F = 1$. Then, by (30), it is straightforward to verify that (25) indeed holds.

Regarding (26), we first let $\mathbf{H}_a(i,j)$ denote the Hessian of $p(j|i,a)$ with respect to a. The second directional derivative $\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}$ satisfies that

$$\begin{aligned}
\left| \left[\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}\Big|_{\delta=0} \right]_{(i,j)} \right| &= \left| \mathbf{W}_i\,\mathbf{H}_a(i,j)\big|_{a=\pi(i)}\,\mathbf{W}_i^\top \right| &(31)\\
&\leq \|\mathbf{W}_i\|_2 \cdot \left\| \mathbf{H}_a(i,j)\big|_{a=\pi(i)}\,\mathbf{W}_i^\top \right\|_2 &(32)\\
&\leq \left\| \mathbf{H}_a(i,j)\big|_{a=\pi(i)} \right\|_F &(33)\\
&< L_p M^2, &(34)
\end{aligned}$$

where (31) holds by the basic property of second directional derivatives, (32) is due to the Cauchy-Schwarz inequality, (33) follows from that $\|\mathbf{W}\|_F \leq 1$ and $\left\|\mathbf{H}_a(i,j)|_{a=\pi(i)}\right\|_2 \leq \left\|\mathbf{H}_a(i,j)|_{a=\pi(i)}\right\|_F$, and (34) holds by the L_p-smoothness of p. Therefore, by (34) we conclude that (26) holds. □

Now we are ready to establish the smoothness conditions of the value functions. Define $L_Q = L_r + ML_p\frac{\gamma}{1-\gamma}$.

Lemma 4 *Under the regularity assumptions (A1)-(A2) and tabular direct policy parameterization, we have the following smoothness properties of $Q(s,a;\pi)$ and $V(s;\pi)$:*

(i) Under any fixed policy π, $Q(s,a;\pi)$ is L_Q-smooth in action a, for any state $s \in \mathcal{S}$.

(ii) There exists some constant $L > 0$ such that $V(s;\pi_\theta)$ is L-smooth in the policy parameters θ, for any state $s \in \mathcal{S}$.

Proof (Lemma 4) **For (i):** Recall that the action-value function can be expressed as

$$Q(s,a;\pi) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V(s';\pi). \tag{35}$$

By taking the derivative of (35) with respect to a, we have

$$\nabla_a Q(s,a;\pi) = \nabla_a r(s,a) + \gamma \sum_{s'\in\mathcal{S}} \nabla_a p(s'|s,a) V(s';\pi). \tag{36}$$

By the regularity assumptions of r and p, we know that for any state s and any two actions a' and a'',

$$\|\nabla_a Q(s,a;\pi)|_{a=a'} - \nabla_a Q(s,a;\pi)|_{a=a''}\|_2 \tag{37}$$

$$\leq \|\nabla_a r(s,a)|_{a=a'} - \nabla_a r(s,a)|_{a=a''}\|_2 + \gamma \sum_{s'\in\mathcal{S}} \Big(\|\nabla_a p(s'|s,a)|_{a=a'} - \nabla_a p(s'|s,a)|_{a=a''}\|_2 \cdot |V(s';\pi)|\Big) \tag{38}$$

$$\leq L_r\|a'-a''\|_2 + ML_p\frac{\gamma}{1-\gamma}\|a'-a''\|_2, \tag{39}$$

where (38) follows from (36) and the triangle inequality, and (39) holds by the regularity assumptions of r and p as well as the fact that $|V(s';\pi)| \leq \frac{1}{1-\gamma}$. This implies that $Q(s,a;\pi)$ is L_Q-smooth in a.

For (ii): We adapt the proof technique in (Agarwal et al., 2019, 2020) and show the smoothness result of $V(s;\pi_\theta)$ in θ. Recall that by the Bellman equation, under a deterministic policy, we have

$$V(s;\pi) = r(s,\pi(s)) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a)V(s';\pi), \quad \forall s\in\mathcal{S}\,. \tag{40}$$

Let r^π be an M-dimensional column vector where the i-th entry is $r(i,\pi(i))$, and V^π denote an M-dimensional column vector where the i-th entry is $V(i;\pi)$. Then, we can rewrite (40) in matrix form as

$$V^\pi = r^\pi + \gamma\,\mathbf{P}^\pi\, V^\pi. \tag{41}$$

By (41), we immediately know that

$$V^\pi = (\mathbf{I}_M - \gamma\,\mathbf{P}^\pi)^{-1} r^\pi. \tag{42}$$

For each $s\in\{1,\cdots,M\}$, let e_s denote an M-dimensional one-hot vector with the s-th entry equal to 1. Hence, for each $s\in\mathcal{S}$, we know $V(s;\pi) = e_s^\top(\mathbf{I}_M - \gamma\,\mathbf{P}^\pi)^{-1}r^\pi$. By Lemma 1, we have

$$\frac{dV_s(\delta)}{d\delta} = \gamma e_s^\top\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)r^\pi, \tag{43}$$

$$\frac{d^2V_s(\delta)}{d\delta^2} = 2\gamma e_s^\top\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)r^\pi + e_s^\top\,\mathbf{G}(\delta)\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}\,\mathbf{G}(\delta)r^\pi. \tag{44}$$

Then, for any $\mathbf{W}\in\mathbb{R}^{M\times N}$ with $\|\mathbf{W}\|_F = 1$, we have

$$\Big|\frac{dV_s(\delta)}{d\delta}\Big| \leq \gamma\Big\|\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)r^\pi\Big\|_\infty \tag{45}$$

$$\leq \frac{\gamma C_1}{(1-\gamma)^2}, \tag{46}$$

where (45) is a direct result of (43), and (46) follows from Lemmas 2-3 and $\|r^\pi\|_\infty \leq 1$. Similarly, we know

$$\Big|\frac{d^2V_s(\delta)}{d\delta^2}\Big| \leq 2\gamma\Big\|\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)\frac{d\,\mathbf{P}(\delta)}{d\delta}\,\mathbf{G}(\delta)r^\pi\Big\|_\infty + \Big\|\mathbf{G}(\delta)\frac{d^2\,\mathbf{P}(\delta)}{d\delta^2}\,\mathbf{G}(\delta)r^\pi\Big\|_\infty \tag{47}$$

$$\leq \frac{2\gamma C_1^2}{(1-\gamma)^3} + \frac{C_2}{(1-\gamma)^2}, \tag{48}$$

where (47) is a direct result of (44), and (48) holds by Lemmas 2-3 and $\|r^\pi\|_\infty \leq 1$. Then, since (47)-(48) hold for any $\mathbf{W}\in\mathbb{R}^{M\times N}$ with $\|\mathbf{W}\|_F = 1$, then we know for every state s, $V(s;\pi_\theta)$ is L-smooth in θ, where $L = \frac{2\gamma C_1^2}{(1-\gamma)^3} + \frac{C_2}{(1-\gamma)^2}$. □

Now, we are ready to prove Proposition 1. For convenience, we restate Proposition 1 below.

Proposition *Under the regularity assumptions (A1)-(A2), there exists some constant $L > 0$ such that for any restarting state distribution μ, $J_\mu(\pi(\cdot;\theta))$ is L-smooth in θ.*

Proof (Proposition 1) By (ii) in Lemma 4 and the definition that $J_\mu(\pi_\theta) = \mathbb{E}_{s\sim\mu}[V(s;\pi_\theta)]$, we know $J_\mu(\pi_\theta)$ is L-smooth, for any restarting state distribution μ. □

A.2 PROOF OF PROPOSITION 2

For convenience, we restate Proposition 2 as follows.

Proposition *Under the FWPO algorithm with $\alpha_k(s) = \frac{(1-\gamma)\mu_{\min}}{LD_s^2} g_k(s)$, $\{\pi(\cdot;\theta_k)\}$ form a non-decreasing sequence of policies in the sense that $\pi(\cdot;\theta_{k+1}) \geq \pi(\cdot;\theta_k)$, for all k. Moreover, the effective Frank-Wolfe gap of FWPO converges to zero as $k \to \infty$, and the convergence rate can be quantified as*

$$\sum_{k=0}^{\infty} \mathcal{G}_k^2 \leq \frac{2LD_{\max}^2}{(1-\gamma)^3\mu_{\min}^2}, \tag{49}$$

which implies that $\bar{\mathcal{G}}_T = O(T^{-1/2})$.

To show the non-decreasing property in Proposition 2, we summarize useful properties on policy improvement as follows. We use $A(\cdot,\cdot;\pi)$ to denote the advantage function of a policy π.

Lemma 5 (Performance difference lemma, (Kakade and Langford, 2002)) *For any two policies π and π', for any restarting state distribution μ, the performance difference between the two policies at any state s is*

$$V(s;\pi') - V(s;\pi) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d_\mu^{\pi'}, a\sim\pi'(\cdot|s)}\big[A(s,a;\pi)\big]. \tag{50}$$

By Lemma 5, we can directly obtain a sufficient condition of state-wise policy improvement.

Corollary 1 *For any two policies π and π', we have $\pi' \geq \pi$ if the following condition holds for every state $s \in \mathcal{S}$:*

$$\mathbb{E}_{a\sim\pi'}[A(s,a;\pi)] \geq \mathbb{E}_{a\sim\pi}[A(s,a;\pi)] = 0. \tag{51}$$

Now we are ready to prove Proposition 2.

Proof (Proposition 2) For ease of notation, in this proof we use $\pi_k \equiv \pi(\cdot;\theta_k)$ and $\pi_k(s) \equiv \pi(s;\theta_k)$. Recall from (8) that under the policy π_k, the state-wise Frank-Wolfe gap is defined as $g_k(s) := \langle c_k(s) - \theta_k(s), \nabla_a Q(s,a;\pi_k)|_{a=\pi_k(s)}\rangle$. By Lemma 4, the $Q(s,a;\pi)$ is L-smooth in a, for any state s and any policy π. Then, under FWPO, we have

$$Q(s,\pi_{k+1}(s);\pi_k) \geq Q(s,\pi_k(s);\pi_k) + \alpha_k(s)\langle c_k(s)-\theta_k(s), \nabla_a Q(s,a;\pi_k)|_{a=\pi_k(s)}\rangle - \frac{L}{2}\alpha_k(s)^2\|c_k(s)-\theta_k(s)\|_2^2, \tag{52}$$

$$\geq Q(s,\pi_k(s);\pi_k) + \alpha_k(s)g_k(s) - \frac{L}{2}\alpha_k(s)^2 D_s^2, \tag{53}$$

where (53) follows from the definitions of the state-wise Frank-Wolfe gap and the diameter D_s. It is easy to verify that $\alpha_k(s)g_k(s) - \frac{L}{2}\alpha_k(s)^2D_s^2$ is positive for all $\alpha_k(s) \in (0, \frac{2g_k(s)}{LD_s^2})$ and attains a maximum of $\frac{g_k(s)^2}{2LD_s^2}$ at $\alpha_k(s) = \frac{g_k(s)}{LD_s^2}$. Therefore, if the learning rate $\alpha_k(s) \in (0, \frac{2g_k(s)}{LD_s^2})$, then $Q(s,\pi_{k+1}(s);\pi_k) > Q(s,\pi_k(s);\pi_k)$ and hence $A(s,\pi_{k+1}(s);\pi_k) > 0$. By Corollary 1, we know $\pi_{k+1} \geq \pi_k$, for all k. Hence, $\{\pi(\cdot;\theta_k)\}$ indeed form a non-decreasing sequence of policies.

Next, we characterize the rate of convergence of the objective $J_\mu(\pi_k)$ of the proposed FWPO algorithm. By Proposition 1, we know the objective $J_\mu(\pi_k)$ is L-smooth. Therefore, we have

$$J_\mu(\pi_{k+1}) \geq J_\mu(\pi_k) + \langle\nabla_\theta J_\mu(\pi_k), \theta_{k+1}-\theta_k\rangle - \frac{L}{2}\|\theta_{k+1}-\theta_k\|_2^2 \tag{54}$$

$$= J_\mu(\pi_k) + \sum_{s\in\mathcal{S}} d_\mu^{\pi_k}(s)\alpha_k(s)\cdot\langle c_k(s)-\theta_k(s), \nabla_a Q(s,a;\pi_k)|_{a=\pi_k(s)}\rangle - \frac{L}{2}\sum_{s\in\mathcal{S}}\alpha_k(s)^2\|c_k(s)-\theta_k(s)\|_2^2 \tag{55}$$

$$\geq J_\mu(\pi_k) + (1-\gamma)\mu_{\min}\sum_{s\in\mathcal{S}}\alpha_k(s)g_k(s) - \frac{L}{2}\sum_{s\in\mathcal{S}}\alpha_k(s)^2D_s^2, \tag{56}$$

where (54) follows from that $J_\mu(\pi_k)$ is L-smooth, (55) holds by the update scheme of FWPO, and (56) follows from that $d_\mu^{\pi_k}(s)\alpha_k(s) \geq (1-\gamma)\mu_{\min}$ and the definition of the diameters. By using (56) and letting $\alpha_k(s) = \frac{g_k(s)}{LD_s^2}(1-\gamma)\mu_{\min}$,

$$J_\mu(\pi_{k+1}) \geq J_\mu(\pi_k) + \sum_{s\in\mathcal{S}} \frac{g_k(s)^2}{2LD_s^2}(1-\gamma)^2\mu_{\min}^2 \tag{57}$$

$$\geq J_\mu(\pi_k) + \frac{(1-\gamma)^2\mu_{\min}^2}{2LD_{\max}^2}\sum_{s\in\mathcal{S}} g_k(s)^2. \tag{58}$$

Recall that $\mathcal{G}_k^2 := \sum_{s\in\mathcal{S}} g_k(s)^2$ denotes the effective Frank-Wolfe gap of the k-th iteration. By taking the telescoping sum of (58), we know

$$J_\mu(\pi_{k+1}) \geq J_\mu(\pi_0) + \frac{(1-\gamma)^2\mu_{\min}^2}{2LD_{\max}^2}\sum_{t=0}^{k} \mathcal{G}_t^2\,. \tag{59}$$

Let π^* denote an optimal policy, i.e., $\pi^* \geq \pi$, for any policy π. Hence, we have

$$\sum_{t=0}^{k}\mathcal{G}_t^2 \leq \frac{2LD_{\max}^2}{(1-\gamma)^2\mu_{\min}^2}(J_\mu(\pi_{k+1}) - J_\mu(\pi_0)) \leq \frac{2LD_{\max}^2}{(1-\gamma)^2\mu_{\min}^2}(J_\mu(\pi^*) - J_\mu(\pi_0)) \leq \frac{2LD_{\max}^2}{(1-\gamma)^3\mu_{\min}^2}, \tag{60}$$

where the last inequality follows from the fact that the value functions are upper bounded by $(1-\gamma)^{-1}$. Recall that $\bar{\mathcal{G}}_T := \min_{0\leq k\leq T}\mathcal{G}_k$. Therefore, (60) implies that

$$\bar{\mathcal{G}}_T \leq \sqrt{\frac{1}{T+1}\sum_{t=0}^{T}\mathcal{G}_t^2} \leq \sqrt{\frac{1}{T+1}\cdot\frac{2LD_{\max}^2}{(1-\gamma)^3\mu_{\min}^2}} = O(T^{-1/2}). \tag{61}$$

This completes the proof. □

B PROOF OF PROPOSITION 3

For convenience, we restate Proposition 3 as follows.

Proposition *If there is no action constraints, then the policy update scheme of NFWPO is equivalent to the vanilla DDPG by (Lillicrap et al., 2016).*

Proof We prove this result by reinterpreting the DDPG from a state-wise perspective. Let $\bar{\theta}$ and $\bar{\phi}$ be the current parameters of the actor and the critic, respectively. As proposed in (Lillicrap et al., 2016), the policy update under DDPG is done by approximating the true gradient $\nabla_\theta J_\mu(\pi(\cdot;\theta))$ by the sample-based gradient $\nabla_\theta \hat{J}_\mu(\pi(\cdot;\theta)) \approx \nabla_\theta J_\mu(\pi(\cdot;\theta))$ as

$$\nabla_\theta \hat{J}_\mu(\pi(\cdot;\theta)) = \frac{1}{|\mathcal{B}|}\sum_{s\in\mathcal{B}} \nabla_a Q(s,a;\phi)|_{a=\pi(s;\bar{\theta})}\nabla_\theta\pi(s;\theta), \tag{62}$$

where $\mathcal{B}$ denotes a mini-batch of states drawn from the replay buffer. Note that this update rule can be reinterpreted from the perspective of regression by the following steps:

- **Reference actions.** For each s in the mini-batch $\mathcal{B}$, compute the reference action at state s under the guidance of the critic

$$a_s^* = \pi(s;\bar{\theta}) + \eta_1\nabla_a Q(s,a;\bar{\phi})|_{a=\pi(s;\bar{\theta})}, \tag{63}$$

  where $\eta_1 > 0$ is the step size.
- **Loss function.** Construct a loss function $L(\theta;\bar{\theta})$ as the mean-squared error (MSE) between the actions of the current policy and the reference actions, i.e.,

$$\mathcal{L}(\theta;\bar{\theta}) = \frac{1}{2|\mathcal{B}|}\sum_{s\in\mathcal{B}}\big(\pi(s;\theta) - a_s^*\big)^2. \tag{64}$$

- **Gradient update.** Accordingly, update the policy parameter by minimizing the MSE loss by applying gradient descent for one step, i.e.,

$$\theta \leftarrow \theta - \eta_2 \nabla_\theta \mathcal{L}(\theta; \bar{\theta}). \tag{65}$$

We can observe that the update scheme characterized by (64)-(65) is equivalent to the DDPG update with a learning rate of $\eta_1\eta_2$. Therefore, DDPG can be viewed as an actor-critic algorithm where both the actor and critic are trained by using regression as subroutines. Note that (63) is equivalent to (12) if there is no action constraints. Hence, we conclude that NFWPO is equivalent to DDPG if there is no action constraints. $\square$

C PSEUDO CODE OF NFWPO

For completeness, we provide the pseudo code of NFWPO in Algorithm 2.

Algorithm 2 Frank-Wolfe Policy Optimization With Neural Policy Parameterization (NFWPO)

```
1: **Input:** Frank-Wolfe learning rate $\alpha$, actor learning rate $\beta$, critic learning rate $\beta_c$, target update ratio $\tau$, and variance of exploration noise $\sigma^2$
2: Randomly initialize the actor $\pi(\cdot;\theta)$ and the critic $Q(\cdot,\cdot;\phi)$ with parameters $\theta, \phi$
3: Initialize the target networks with parameters $\theta^\dagger, \phi^\dagger$
4: **for** each episode $i = 0, 1, \cdots$ **do**
5:     Let $\bar{\theta}$ and $\bar{\phi}$ be the snapshots of the current actor and critic parameters
6:     Receive initial state $s_0$ of the current episode
7:     **for** $t = 0, 1, \cdots$ **do**
8:         Select action $a_t = \pi(s_t;\bar{\theta}) + \mathcal{N}(0, \sigma^2)$
9:         Apply action $a_t$ and observe reward $r_t$ as well as the next state $s_{t+1}$
10:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in the replay buffer
11:        Randomly sample a mini-batch $\mathcal{B}$ of transitions $(s, a, r, s')$ from the replay buffer
12:        // update the actor by state-wise Frank-Wolfe
13:        **for** each state $s \in \mathcal{B}$ **do**
14:            $\bar{c}(s) = \operatorname{argmax}_{c\in\mathcal{C}(s)} \langle c, \nabla_a Q(s,a;\bar{\phi})|_{a=\Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta}))}\rangle$
15:            $\tilde{a}_s = \Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta})) + \alpha\big(\bar{c}(s) - \Pi_{\mathcal{C}(s)}(\pi(s;\bar{\theta}))\big)$
16:        **end for**
17:        $\mathcal{L}_{\text{NFWPO}}(\theta;\bar{\theta}) = \sum_{s\in\mathcal{B}} \big(\pi(s;\theta) - \tilde{a}_s\big)^2$
18:        $\theta \leftarrow \theta - \beta\nabla_\theta \mathcal{L}_{\text{NFWPO}}(\theta;\bar{\theta})|_{\theta=\bar{\theta}}$
19:        // update the Q-learning-like critic as vanilla DDPG
20:        $\mathcal{L}_{\text{critic}}(\phi) = \frac{1}{|\mathcal{B}|}\sum_{(s,a,r,s')\in\mathcal{B}} \big(r + \gamma Q(s', \pi(s';\theta^\dagger);\phi^\dagger) - Q(s,a;\phi)\big)^2$
21:        $\phi \leftarrow \phi - \beta_c \nabla_\phi \mathcal{L}_{\text{critic}}(\phi)|_{\phi=\bar{\phi}}$
22:        Update the target networks: $\theta^\dagger \leftarrow \tau\theta + (1-\tau)\theta^\dagger$, $\phi^\dagger \leftarrow \tau\phi + (1-\tau)\phi^\dagger$
23:    **end for**
24: **end for**
```

D DETAILED EXPERIMENTAL SETUP AND ADDITIONAL EXPERIMENTAL RESULTS

D.1 TRAINING CONFIGURATIONS

Random seeds. For each task, each algorithm is trained under the common set of random seeds of $\{0, 1, 2, 3, 4\}$.

Exploration. The training process starts after some number of time steps (1000 steps for Reacher-v2 and 10000 steps for the other environments), and we use a purely exploratory policy in this initial phase to collect samples for the replay buffer for all the algorithms. During training, Gaussian noise is added to each action for exploration for the neural cases. In the tabular case, we use the ϵ-greedy policy as the behavior policy instead.

Update frequency. For NFSNET, due to the longer computation time required by the network simulator, we speed up the simulations by updating the actor every 50 steps for all the algorithms. For the other environments, as DDPG+OptLayer is particularly time-consuming, we also update its actor every 50 steps to speed up the training process.

Implementation of OptLayer. As there is no off-the-shelf implementation of DDPG+OptLayer available, we leverage the open-source packages cvxpylayer (Agrawal et al., 2019) as well as the OptNet proposed by (Amos and Kolter, 2017) to implement the differentiable projection-based OptLayer for end-to-end training. Note that in DDPG+OptLayer there are two scenarios where projection is needed for the actor output: (i) producing actions for interacting with the environment (under a behavior policy) and (ii) evaluating the actions produced by the current policy for calculating the deterministic policy gradient as in (62). Since OptLayer is computationally inefficient, we use it only for the scenario (ii), where back-propagation is required for gradient descent. For scenario (i), we use Gurobi optimization solver instead to speed up the training process.

D.2 A SUMMARY OF THE HYPERPARAMETERS

In this section, we provide a summary of the key hyperparameters in Tables 3-4. We highlight some key design choices:

- For both the actor and critic networks, we use two hidden layers with 400 and 300 hidden units with ReLU as the activation, as suggested by Fujimoto et al. (2018). For the activation function of the actor output, in order to better accommodate the various action ranges of different environments, we choose tanh for the MuJoCo control tasks and ReLU for the other tasks, respectively.

- We choose a smaller batch size for DDPG+OptLayer since its computation time is much larger than the other approaches. As DDPG+OptLayer is a strong baseline in some of the environments, we also set the batch size of NFWPO to be 16 for a fair comparison.

- For SAC algorithm, we use the open-source Spinning up (Achiam, 2018) and its default setting. For PPO and TRPO, we use the open-source provide by (Fujita and Maeda, 2018), which is based on ChainerRL (Fujita et al., 2021) with its default setting.

D.3 ADDITIONAL EXPERIMENTAL RESULTS

In this section, we further compare NFWPO with the baseline methods designed to address special types of action constraints in the HalfCheetah-v2 environment.

Comparison with Clipped Action Policy Gradient (Fujita and Maeda, 2018). We further compare NFWPO with the Clipped Action Policy Gradient (CAPG) algorithms. As CAPG can only handle bound constraints, we evaluate them with the intrinsic bound constraints in HalfCheetah (i.e., each action entry must be in $[-1, +1]$). The results in Figure 6 show that NFWPO outperforms the two versions of CAPG, namely TRPO-CAPG and PPO-CAPG, under the bound constraints.

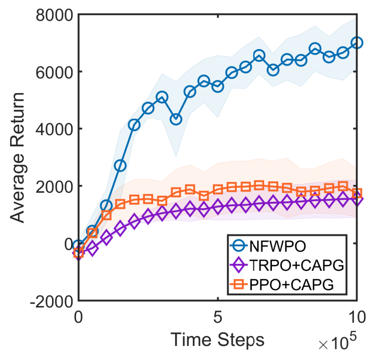


Figure 6: Halfcheetah-v2 with bound constraints: Average return at each evaluation over 5 random seeds

Table 3: A summary of the hyper-parameters of NFWPO and the baseline methods.

Hyper-parameters	Reacher-v2	HalfCheetah-v2	NSFNET	BSS-5
Learning Rate (For Frank-Wolfe)	0.05	0.01	0.05	0.05
Actor Learning Rate (For others)	0.0001	0.0001	0.0001	0.0001
Critic Learning Rate	0.001	0.001	0.001	0.001
Discount Factor	0.99	0.99	0.99	0.99
Target Update Ratio (τ)	0.001	0.001	0.001	0.001
Replay Buffer Size	10^4	10^6	5×10^4	10^6
Evaluation Frequency	5000	5000	10000	5000
Total Training Steps	3×10^5	7×10^5	5×10^5	100000
Starting Time of Training	1000	10000	10000	10000
Additive Action Noise for Exploration	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 0.1)$	$\mathcal{N}(0, 3)$	$\mathcal{N}(0, 5)$
Weight of Reward Shaping	$\frac{1}{7}$	3 (state-dependent), 2 (quadratic)	$\frac{1}{4}$	4
Batch Size (For DDPG+RewardShaping)	64	64	64	64
Batch Size (For DDPG+Projection)	64	64	64	64
Batch Size (For DDPG+OptLayer)	16	16	16	16
Batch Size (For FWPO/NFWPO)	16	16	16	16

Table 4: The configurations of the hyper-parameters for tabular policies in the bike-sharing environment.

Hyper-parameters	BSS-3
Critic Network	(30, out)
Learning Rate (For Frank-Wolfe)	0.05
Actor Learning Rate (For Others)	0.001
Critic Learning Rate	0.002
Discount Factor	0.9
Target Update Ratio for Critic (τ)	0.01
Target Update Frequency for Actor	100
Replay Buffer Size	10^4
Evaluation Frequency	5000
Total Training Steps	10^6
Starting Time of Training	10000
Exploratory Behavior Policy	ϵ-greedy, $\epsilon = 0.1$
Weight of Reward Shaping	1
Batch Size (For All Methods)	64