On-Policy Deep Reinforcement Learning

The Discounted and Average Reward Criteria

by

Yiming Zhang

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2022

Keith W. Ross

Acknowledgements

First and foremost, I would like to thank my advisor Keith Ross. I am deeply grateful to him for taking me as his PhD student, as well as being a constant source of support and encouragement. His ability to tackle and think about research problems from first principles, his rigor, and his creativity has shaped me into the researcher I am today. I will forever treasure the countless hours of conversations we have had over the past five years. Keith will always be a role model to me as a researcher, a teacher, and a person. I am also grateful to my committee members Kyunghyun Cho, He He, Xi Chen, and Chris Musco for their thoughtful and constructive feedback throughout the entire process.

I am forever grateful to count myself as a member of the NYU community. I want to thank all my friends, colleagues, collaborators, and mentors at NYU who have deeply impacted me over the years: Che (Watcher) Wang, Yanqiu (Autumn) Wu, Sean Welleck, Tianyao Chen, Ziyu Wang, Mehryar Mohri, Ningshan Zhang, Scott Yang, Lingfan Yu, Yixin Hu, Roberta Raileanu, Junbo (Jake) Zhao, Jing Leng, Krzysztof Geras, Ilya Kostrikov, Elman Manslov, Alfredo Canzani. I also want to thank Santiago Pizzini, Hong Tam, and Rosemary Amico for all your hard work in ensuring the smooth operations of the graduate program.

I am extremely thankful for the opportunity to conduct research at NYU Shanghai where I spent a memorable two and a half years. Thank you Eric Mao, Xiaoyun (Vivien) Du, Fangqi (Maggie) Mao, Lin Hong and countless others behind the scenes for making NYU Shanghai feel like home for graduate students such as myself. I am thankful to my friends, collaborators and

colleagues Quan Vuong, Xiaoyue Gong, Kenny Song, Xinyue Chen, Zijian Zhou, Canyu Zhu, Mufei Li, Aiwen Xu, Guangyu (Gus) Xia, Shuyang Ling, and Zheng Zhang (ZZ). Also I want to thank Diane Geng for introducing me to the wonderful community at NYUSH prior to starting my PhD. Thank you Zhiguo Qi for maintaining the NYU Shanghai HPC center and helping us overcome many of the technical difficulties on the compute clusters.

I want to give a special thanks to my friends who have accompanied me throughout this special journey: Zhihao Wu, Wenyu Sheng, Yi Zhang, Bingqing Xu, Ruofan Wu, Wenchao Wu, Ming Cong, Chang Zhao, Yue Sun, Xiaoai Lü, Lu'an Jiang, Qiushi Wang, Yangyang Hu, Xiaoxin Du, Yudi Yang, Bryant Feng, Amata Lee, Joe Wu, Nishant Mohanchandra, Benjamin Xie, Stephen Bates, Alina Zhu, and many many others.

Thank you Michelle for always being an unconditional beacon of support and encouragement.

Finally I want to thank my parents for their unwavering support over the years.

Abstract

Reinforcement Learning (RL) is the study of sequential decision making where an agent attempts to maximize its overall cumulative reward in some given environment. Combined with deep learning, reinforcement learning has made remarkable strides in the past decade in complex tasks such as playing video games [Mnih et al. 2013; Vinyals et al. 2019], playing Go [Silver et al. 2016, 2018], robotics [Lillicrap et al. 2016; Haarnoja et al. 2018], and chip design [Mirhoseini et al. 2021].

However despite these successes, modern RL algorithms often suffer from poor sample efficiency and lack of safety guarantees. In this thesis we tackle these issues in the context of on-policy Deep Reinforcement Learning (DRL), both theoretically and algorithmically. This work addresses both the discounted and average reward criteria. In the first part of this thesis, we develop theory for average reward on-policy reinforcement learning by extending recent results for local policy search. We show that previous work based on the discounted return [Schulman et al. 2015; Achiam et al. 2017] results in a non-meaningful bound in the average-reward setting. By addressing the average-reward criterion directly, we derive a novel bound which depends on the average divergence between the two policies and Kemeny's constant. Based on this bound, we develop an iterative procedure which produces a sequence of monotonically improved policies for the average reward criterion. We show that this iterative procedure can then be combined with classic deep reinforcement learning methods, resulting in practical DRL algorithms that target the long-run average reward criterion. Next, we develop a unifying framework for the on-policy sample efficiency problem. This methodology uses a two-step approach which first learns an

optimal policy in the non-parameterized policy space before projecting said policy back into the parameter space. Our approach is general in that it applies to both discrete and continuous action spaces, and can handle a wide variety of proximity constraints. Finally we address the problem of reinforcement learning with safety constraints. We provide theoretical support that trust region-based methods can be applied to problems with both discounted and non-discounted cost constraints. We then propose a novel first-order algorithm for policy optimization for maximizing an agent's cumulative reward while at the same time satisfying a set of cost constraints. Our algorithm is extremely simple to implement and has an approximate upper bound for worst-case constraint violation throughout training.

Contents

List of Figures

List of Tables

1 Introduction

One of the goals of modern artificial intelligence is to produce agents for sequential decision making which can learn desirable behavior by interacting with the environment through trial and error [Norvig and Russell 2002; Arulkumaran et al. 2017]. Reinforcement Learning (RL), which builds on the theory of Markov Decision Processes (MDP) provides a fundamental framework for this learning paradigm. Although traditional tabular reinforcement learning has enjoyed some moderate success, it suffers from the *curse of dimensionality* [Bellman 1957] and fails on problems with large state and/or action spaces.

Recent advances in deep learning [Goodfellow et al. 2016] has had a significant impact on many areas of machine learning, including reinforcement learning. Parameterizing polices and value functions with deep neural networks allows the agent to generalize over large and also continuous state and action spaces. This gave rise to the modern subfield of Deep Reinforcement Learning (DRL).

One of the key challenges for modern RL is the problem of *sample efficiency*, which can be defined the number of calls to the environment required to attain a specified performance level [Kakade 2003]. Although in practice, sample efficiency is often evaluated empirically by comparing the performance of different algorithms after a fixed number of agent-environment interactions, which will also be the focus of this work. Low sample efficiency is particularly problematic in domains where data collection can be extremely expensive, such as robotics where operating a robot for an extended amount of time can lead to wear and tear of the hardware.

This work will focus on the sample efficiency problem within the scope of *on-policy* deep reinforcement learning, which describes a class of algorithms where the agent attempts to improve or evaluate the policy used for data collection and decision making. This is opposed to *off-policy methods* where the policy that the agent is attempting to evaluate or improve is different from the policy (or policies) used to collect data. More concretely in on-policy algorithms, an agent collects trajectories from the current policy, performs policy evaluation and update, then the previously collected data are immediate discarded and the agent collects new data from the updated policy. A key question for on-policy algorithms is then *after collecting data from the environment, how do we use it in the most efficient manner?* While recent work has shown that well-designed off-policy algorithms are in general more sample-efficient [Haarnoja et al. 2018; Chen et al. 2021], on-policy methods serve as the foundation for most off-policy and/or model-based techniques [Janner et al. 2019], as well as emerging new sub-fields such as offline RL [Levine et al. 2020] and meta-RL [Wang et al. 2016; Duan et al. 2017]. Therefore a deeper understanding of on-policy methods is essential to advancing the field of RL as a whole.

This thesis attempts to address the sample-efficiency issue through the following perspectives:

First we look at the problem of *average reward* deep reinforcement learning. Much of the recent advances in DRL relies on discounting rewards into the future since this allows for a simpler mathematical formulation and more stable behavior in practice [Naik et al. 2019]. However, when the natural objective of the problem is non-discounted, using discounting in the training objective leads to a discrepancy between the problem we are interested in solving and what we are actually solving. Furthermore, we find that using a shorter effective horizon for training can have a significant negative impact in terms of both sample efficiency and asymptotic performance. We address these issues by developing theory and algorithms for engaging the average reward criteria directly.

Next, we develop a unified algorithmic framework for solving the on-policy sample efficiency problem through local policy search, i.e. we aim to answer the question given a current policy,

what is the most sample-efficient policy update within some local region around the current policy?

Lastly, we focus on the special case of reinforcement learning with safety constraints, i.e. how do we develop sample-efficient algorithms given that the agent must satisfy a set of cost constraints?

In terms of the broader impact of this research, the goal of this thesis is to bring RL closer to more practical applications with positive societal impacts. In the real world, higher sample efficiency translates to time saved, lower costs, lower resource consumption, and lower carbon emission. This of central concern and is instrumental for making RL practical for applications in real physical systems such as robots and self-driving cars.

1.1 Outline of Thesis

Chapter 2 provides background material necessary for this thesis, this includes the basic definition and properties of a Markov Decision Process (MDP), how MDPs are classified, a brief introduction to mean passage time and mixing time for Markov chains, the discounted and average criterion and their respective Bellman equations, and finally the concept of Blackwell optimality.

Chapter 3 begins by introducing the policy gradient theorem and the step-size issue for policy gradient algorithms. We then discuss how local policy search is used to address this issue. The core of the chapter is dedicated to extending the monotonic policy improvement bound for a certain class of local policy search problems to accommodate for the average reward case. Here we first showed that the aforementioned policy improvement bound results in a non-meaningful bound in the average reward case. We then derived a novel bound which depends on the average divergence between the two policies and Kemeny's constant. Based on this bound, we develop an iterative procedure which produces a sequence of monotonically improved policies for the average reward criterion.

Chapter 4 gives a practical algorithm for learning policies in continuous and/or high dimensional state and action spaces based on the theoretical results introduced in Chapter 3. The algorithm is an extension of the well-known TRPO algorithm to the average reward criterion and we demonstrated that our new algorithm Average Reward TRPO (ATRPO) significantly outperforms discounted TRPO on the most challenging MuJoCo environments.

Chapter 5 introduces a unifying framework for effectively solving the local policy search problems via a two-step solution. This framework can flexibly accommodate for difference divergence measures and is shown to demonstrate superior sample efficiency compared to other on-policy algorithms on both the MuJoCo and Atari benchmarks.

Chapter 6 develops theory and algorithms for the setting of *constrained RL*. We begin by introducing Constrained Markov Decision Processes (CMDPs) and extending theoretical guarantees for local policy search for CMDPs with discounted costs to the case of non-discounted costs. We then introduce a simple first-order algorithm which can achieve better sample efficiency on a set of continuous control tasks compared to more complex second-order methods while consistently maintaining constraint satisfaction.

Chapter 7 concludes with a discussion and provides directions for future work.

1.2 List of Contributions

- Vuong, Q., Zhang, Y., Ross, K. (2019). Supervised Policy Update for Deep Reinforcement Learning. *International Conference for Learning Representations (ICLR).*

- Zhang, Y., Vuong, Q., Ross, K. (2020). First Order Constrained Optimization in Policy Space. *Neural Information Processing Systems (NeurIPS).*

- Zhang, Y., Ross, K. (2021). On-Policy Deep Reinforcement Learning for the Average Reward Criterion. *International Conference on Machine Learning (ICML).*

2 Preliminaries

In this chapter, we will give a basic introduction to the preliminaries and notations required for this work. The chapter can be roughly divided into three parts. The first two sections present the basic definition of a Markov Decision Process. Sections 2.3 and 2.4 introduces two ideas from Markov chains which are of fundamental importance in the rest of this work. Finally in the last four sections, we present the discounted and average reward criteria used to evaluate performance in Markov Decision Processes and how the two criteria are connected.

2.1 Markov Decision Processes

Consider a Markov Decision Process (MDP) [Sutton and Barto 2018] $(\mathcal{S}, \mathcal{A}, P, r, d_0)$ where the state space $\mathcal{S}$ and action space $\mathcal{A}$ are assumed to be finite[1]. The transition probability is denoted by $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ where $P(s'|s, a)$ is the probability of transitioning into state s' after taking action a in state s. The reward function is $r : \mathcal{S} \times \mathcal{A} \to [r_{\min}, r_{\max}]$ where $r(s, a)$ is the (bounded) immediate reward received by the agent after taking action a while in state s. And $d_0 : \mathcal{S} \to [0, 1]$ is the initial state distribution.

Let $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ be a stationary policy where $\Delta(\mathcal{A})$ is the probability simplex over $\mathcal{A}$, and Π is the set of all stationary policies. We use $\tau = (s_0, a_0, s_1, a_1, \cdots) \sim \pi$ to denote a sample trajectory

[1] With the exception of Section 5.4, the theoretical analysis of this thesis will focus exclusively on finite state and action spaces, however we will demonstrate that our algorithmic contributions can easily be applied to continuous domains.

generated by the policy π, i.e. $s_0 \sim d_0$, $a_t \sim \pi(\cdot|s_t)$, and $s_{t+1} \sim P(\cdot|s_t, a_t)$. For tabular problems, π can be a table of probabilitic values. However for problems with large or continuous state and action spaces, we generally consider a set of parameterized policies $\Pi_\theta = \{\pi_\theta | \theta \in \Theta\} \in \Pi$. In particular for Deep Reinforcement Learning, Π_θ is a set of neural networks with the same architecture.

In summary, the tuple $(\mathcal{S}, \mathcal{A}, P, r, d_0)$ gives the *dynamics* of the environment and the policy π determines how the agent interact with the environment. Using the language of MDPs, the sequence of environment interactions of an RL agent can be described as follows: At time t, the agent is in state s_t, it takes action a_t according to policy π, receives a reward $r(s_t, a_t)$, then transitions to state s_{t+1} according to $P(\cdot|s_t, a_t)$.

A policy π also induces a Markov chain over the state space and the state action-space. Let $P_\pi \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ be the *state transition matrix* under policy π whose (s, s') components are $P_\pi(s'|s) = \sum_a P(s'|s, a)\pi(a|s)$. We use $P^*_\pi := \lim_{N\to\infty} \frac{1}{N} \sum_{t=0}^{N} P^t_\pi$ to denote the limiting distribution of the transition matrix P_π. Let $d_\pi \in \Delta(\mathcal{S})$ be the stationary distributions of P_π, if one exists. For aperiodic chains, we have $P^*_\pi = \lim_{t\to\infty} P^t_\pi = \mathbf{1}d^T_\pi$ where $\mathbf{1}$ is the column vector of appropriate dimensions consisting of all 1's.

2.2 Classification of Markov Decision Processes

We can classify MDPs based on the chain structure of the underlying Markov chain, and whether states within the MDP are accessible from each other under some stationary policy π. More concretely, MDPs can be classified into the following five classes [Puterman 1994]:

- **Ergodic**: For every stationary policy, the induced Markov chain is irreducible and aperiodic.
- **Unichain**: For every stationary policy, the induced Markov chain contains a single recurrent class and a finite but possibly empty set of transient states.

- **Communicating**: For every pair of states s and s' in $\mathcal{S}$, there exists a stationary policy π under which s' is accessible from s, that is, $P_\pi^t(s'|s) > 0$ for some $t \geqslant 1$.

- **Weakly Communicating**: There exists a closed set of states where each state in that set is accessible from every other state in that set under some stationary policy, and a finite but possibly empty set of states which is transient under every policy.

- **Multichain**: The transition matrix corresponding to at least one stationary policy contains two or more closed irreducible recurrent classes.

From the above definitions, we see that unichain MDPs generalize Ergodic MDPs with a set of transient states. Similarly, weakly communicating MDPs may be viewed as communicating MDPs with an additional set of transient states. Ergodic MDPs are always communicating, and unichain MDPs are always weakly communicating. In this work, we will focus on ergodic and unichain MDPs.

2.3 Mean Passage Time and the Fundamental Matrix

In this section and the next, we introduce several concepts important to understanding the long-term behavior of Markov chains. For a more detailed account, please refer to Brémaud [2020].

Let P be the transition matrix of a finite Markov chain with n states[2]. Define the *fundamental matrix of a Markov chain* with transition matrix P as

$$Z = (I - P + P^*)^{-1} \tag{2.1}$$

where as before, we use P^* to denote the limiting distribution of the transition matrix P. The fundamental matrix exists for all finite Markov chains [Bertsekas et al. 1995]. For any aperiodic

[2]These results naturally apply to the policy-induced Markov chain of an MDP P_π.

Markov chain, we can also show that (Theorem 4.3.1, Kemeny and Snell [1960])

$$Z = I + \sum_{t=1}^{\infty} (P^t - P^*) \tag{2.2}$$

For ergodic Markov chains, one important application of the fundamental matrix is the calculation of the *mean first passage time*. Let $M \in \mathbb{R}^{n \times n}$ be the *mean first passage time matrix* whose elements $M(s, s')$ is the expected number of steps it takes to reach state s' from s. For any ergodic MDP, the matrix M can be calculated via (Theorem 4.4.7, Kemeny and Snell [1960])

$$M = (I - Z + EZ_{\text{dg}})D \tag{2.3}$$

where E is a square matrix consisting of all ones. The subscript 'dg' refers to a matrix whose diagonal entries are the diagonals of some square matrix and the rest of the entries are zero. $D \in \mathbb{R}^{n \times n}$ is a diagonal matrix whose entries are $1/d(s)$ where d is the stationary distribution.

One important property of mean first passage time is that for ergodic MDP, the quantity

$$\kappa = \sum_{s'} d(s')M(s, s') = \text{trace}(Z) \tag{2.4}$$

is a constant independent of the starting state (Theorem 4.4.10, Kemeny and Snell [1960].) The constant κ is sometimes referred to as *Kemeny's constant* [Grinstead and Snell 2012]. This constant can be interpreted as the mean number of steps it takes to get to any goal state weighted by the steady-distribution of the goal states. This weighted mean does not depend on the starting state, as mentioned just above.

2.4 Mixing Time

In this section, we delve deeper into the asymptotic behavior of ergodic Markov chains. When the state space is finite, this behavior can be entirely characterized by the eigen-structure of the transition matrix P [Brémaud 2020]. This is formalized in the Perron-Frobenius Theorem [Perron 1907; Frobenius 1912].

Theorem 2.1 (Perron-Frobenius)**.** *Let P be the transition matrix of an irreducible aperiodic Markov chain and its eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$ which are ordered such that $1 = \lambda_1 > |\lambda_2| \geqslant \cdots \geqslant |\lambda_n|$. Then*

$$P^t = \mathbf{1}d^T + O(t^{m_2-1}|\lambda_2|^t) \tag{2.5}$$

where m_j is the algebraic multiplicity of λ_j.

A proof can be found in Seneta [2006]. The Perron-Frobenius Theorem states that the relative speed of convergence is determined by the *Second Largest Eigenvalue Modulo (SLEM)*. Subsequently, we will use λ^* to denote the SLEM.

The idea of *mixing time* characterizes the time it takes for a Markov chain to converge to its stationary distribution. Formally, the mixing time for a transition matrix P is defined as

$$t_{\text{mix}}(\epsilon) := \min\{t : \max_s D_{\text{TV}}(P^t(\cdot|s) \parallel d) \leqslant \epsilon\} \tag{2.6}$$

where we use $P^t(\cdot|s)$ to denote the sth row of P^t and D_{TV} is the *total variation distance* which is defined as

$$D_{\text{TV}}(p \parallel q) := \frac{1}{2}\|p - q\|_1 = \frac{1}{2}\sum_x |p(x) - q(x)| \tag{2.7}$$

for any two finite probability distributions p and q.

By the Perron-Frobenius Theorem, the SLEM determines the convergence speed of P^t. It therefore should not be suprising that $t_{\text{mix}}(\epsilon)$ is also related to SLEM. In fact for any irreducible

and aperiodic Markov chain, We can upper-bound the mixing time by the SLEM (Theorem 12.5, Levin and Peres [2017]):

$$t_{\text{mix}}(\epsilon) \geqslant \left(\frac{1}{1-\lambda^*} - 1\right) \log\left(\frac{1}{2\epsilon}\right) \tag{2.8}$$

For a more in-depth discussion of Markov Chains and mixing times, see Levin and Peres [2017]; Brémaud [2020].

It turns out that a Markov chain's mixing time is related to Kemeny's constant through SLEM:

Proposition 2.2. *For an ergodic Markov chain, let* $1 = \lambda_1 > \lambda_2 \geqslant \cdots \geqslant \lambda_n > -1$ *be the eigenvalues of* P*, we have*

$$\kappa \leqslant 1 + \frac{n-1}{1-\lambda^\star} \tag{2.9}$$

where $\lambda^\star = \max_{i=2,\ldots,n} |\lambda_i|$.

Proof. Let λ be an eigenvalue of P and u its corresponding eigenvector. Since P is aperiodic, $\lambda \neq -1$, we then have

$$\begin{aligned}(I - P + P^\star)u &= u - Pu + \lim_{n\to\infty} P^t u \\ &= (1-\lambda)u + u \lim_{t\to\infty} \lambda^t \\ &= \left(1 - \lambda + \lim_{t\to\infty} \lambda^t\right) u\end{aligned} \tag{2.10}$$

where $\lim_{t\to\infty} \lambda^t = 1$ when $\lambda = 1$ and 0 when $|\lambda| < 1$. Therefore, $(I - P + P^\star)$ has eigenvalues $1, 1-\lambda_2, \ldots, 1-\lambda_n$. The fundamental matrix $Z = (I - P + P^\star)^{-1}$ has eigenvalues $1, \frac{1}{1-\lambda_2}, \cdots, \frac{1}{1-\lambda_n}$. We can then upper bound Kemeny's constant by

$$\kappa = \text{trace}(Z) = 1 + \sum_{i=2}^{n} \frac{1}{1-\lambda_i} \leqslant 1 + \sum_{i=2}^{n} \frac{1}{1-|\lambda_i|} \leqslant 1 + \frac{n-1}{1-\lambda^\star} \tag{2.11}$$

□

2.5 The Discounted Reward Criterion

For some *discount factor* $\gamma \in (0, 1)$, the discounted reward objective is defined as

$$\rho_\gamma(\pi) := \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right] = \frac{1}{1-\gamma}\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi,\gamma}\\ a\sim\pi}}[r(s, a)] \tag{2.12}$$

where

$$d_{\pi,\gamma}(s) := (1-\gamma)\sum_{t=0}^{\infty}\gamma^t P_{\tau\sim\pi}(s_t = s) \tag{2.13}$$

is known as the *future discounted state visitation distribution under policy* π. For any bounded reward function, the discounted objective is guaranteed to converge to a finite value. Note also that the discounted objective depends on the initial state distribution μ. The distribution $d_{\pi,\gamma}(s)$ can be regarded as the discounted analogy to the stationary distribution and it can be easily shown that $d_{\pi,\gamma}(s) \to d_\pi(s)$ for all $s \in \mathcal{S}$ as $\gamma \to 1$ [Kakade and Langford 2002].

The *discounted state-value function*, defined as

$$V_\gamma^\pi(s) := \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\middle| s_0 = s\right], \tag{2.14}$$

is the expected discounted return of policy π starting in state s. It is clear from definition that $\rho_\gamma(\pi) = \mathbb{E}_{s\sim d_0}\left[V_\gamma^\pi(s)\right]$. The *discounted action-value function*

$$Q_\gamma^\pi(s, a) := \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\middle| s_0 = s, a_0 = a\right]. \tag{2.15}$$

is the expected discounted return of policy π after taking action a in state s.

The state-value function $V_\gamma^\pi(s)$ and action-value function $Q_\gamma^\pi(s)$ are related through the fol-

lowing equations [Sutton and Barto 2018]:

$$V_\gamma^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) Q_\gamma^\pi(s,a) \tag{2.16}$$

$$Q_\gamma^\pi(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) V_\gamma^\pi(s'). \tag{2.17}$$

A fundamental property of $V_\gamma^\pi(s)$ and $Q_\gamma^\pi(s)$ is that they can be expressed using the following recursive relations known as the *Bellman equations* [Sutton and Barto 2018]:

$$V_\gamma^\pi(s) = \sum_{a\in\mathcal{A}} \pi(a|s) \left(r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|s,a) V_\gamma^\pi(s') \right) \tag{2.18}$$

$$Q_\gamma^\pi(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s,a) \sum_{a'\in\mathcal{A}} \pi(a'|s') Q_\gamma^\pi(s,a). \tag{2.19}$$

Finally, the *discounted advantage function* is defined as

$$A_\gamma^\pi(s,a) := Q_\gamma^\pi(s,a) - V_\gamma^\pi(s). \tag{2.20}$$

The advantage function can be interpreted as measuring how much better than average a particular action a is in state s.

2.6 The Average Reward Criterion

The average reward objective is defined as:

$$\rho(\pi) := \lim_{N\to\infty} \frac{1}{N} \mathop{\mathbb{E}}_{\tau\sim\pi} \left[\sum_{t=0}^{N-1} r(s_t, a_t) \right] = \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi}} [r(s,a)]. \tag{2.21}$$

The limit is guaranteed to exist when π is stationary [Puterman 1994]. The term $\rho(\pi)$ is also referred to as the *gain* of policy π. In the unichain case, the average reward $\rho(\pi)$ does not depend

on the initial state for any stationary policy π [Bertsekas et al. 1995].

We express the *average-reward bias function* as

$$\bar{V}^{\pi}(s) := \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}(r(s_t,a_t)-\rho(\pi))\middle| s_0=s\right] \tag{2.22}$$

and *average-reward action-bias function* as

$$\bar{Q}^{\pi}(s,a) := \mathop{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}(r(s_t,a_t)-\rho(\pi))\middle| s_0=s, a_0=a\right]. \tag{2.23}$$

The bias functions $\bar{V}^{\pi}(s)$ and $\bar{Q}^{\pi}(s,a)$ are the average reward analogies of $V_{\gamma}^{\pi}(s)$ and $Q_{\gamma}^{\pi}(s,a)$, the extra "$-\rho(\pi)$" term is to guarantee the convergence of the infinite series. Similar to the discounted case, we can show that

$$\bar{V}^{\pi}(s) = \sum_{a\in\mathcal{A}}\pi(a|s)Q_{\gamma}^{\pi}(s,a) \tag{2.24}$$

$$\bar{Q}^{\pi}(s,a) = r(s,a)-\rho(\pi)+\sum_{s'\in\mathcal{S}}P(s'|s,a)V_{\gamma}^{\pi}(s'). \tag{2.25}$$

The average reward bias and action-bias functions also satisfy the Bellman equation [Sutton and Barto 2018]

$$\bar{V}^{\pi}(s) = \sum_{a\in\mathcal{A}}\pi(a|s)\left(r(s,a)-\rho(\pi)+\sum_{s'\in\mathcal{S}}p(s'|s,a)\bar{V}^{\pi}(s')\right) \tag{2.26}$$

$$\bar{Q}^{\pi}(s,a) = r(s,a)-\rho(\pi)+\sum_{s'\in\mathcal{S}}P(s'|s,a)\sum_{a'\in\mathcal{A}}\pi(a'|s')\bar{Q}^{\pi}(s,a). \tag{2.27}$$

Note that these equations take a slightly different form compared to the discounted Bellman equations, there are no discount factors and the rewards are now replaced with $r(s,a)-\rho(\pi)$.[3]

[3] In Sutton and Barto [2018], the term $r(s,a)-\rho(\pi)$ is referred to as the *differential reward*.

Finally, we define the *average-reward advantage function* as

$$\bar{A}^{\pi}(s,a) := \bar{Q}^{\pi}(s,a) - \bar{V}^{\pi}(s). \tag{2.28}$$

2.7 Connecting the Discounted and Average Reward Criteria

We noted in the previous section that the bias function is the average reward analogy of the value function, in fact it can be shown that the bias function and the value function satisfy the following relationship:

Proposition 2.3 (Blackwell 1962)**.** *For a given stationary policy π and discount factor $\gamma \in (0,1)$,*

$$\lim_{\gamma \to 1} \left(V_{\gamma}^{\pi}(s) - \frac{\rho(\pi)}{1-\gamma} \right) = \bar{V}^{\pi}(s) \tag{2.29}$$

for all $s \in \mathcal{S}$.

It is straight forward to extend the above results to the action-bias (value) functions and advantage functions.

Corollary 2.4. *For a given stationary policy π and discount factor $\gamma \in (0,1)$,*

$$\lim_{\gamma \to 1} \left(Q_{\gamma}^{\pi}(s,a) - \frac{\rho(\pi)}{1-\gamma} \right) = \bar{Q}^{\pi}(s,a) \tag{2.30}$$

$$\lim_{\gamma \to 1} A_{\gamma}^{\pi}(s,a) = \bar{A}^{\pi}(s,a) \tag{2.31}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

Proof. From Proposition 2.3, we can rewrite (2.29) as

$$V_\gamma^\pi(s) = \frac{\rho(\pi)}{1-\gamma} + \bar{V}^\pi(s) + g(\gamma, s) \tag{2.32}$$

where $\lim_{\gamma \to 1} g(\gamma, s) = 0$. We then expand $Q_\gamma^\pi(s,a)$ using the Bellman equation

$$\begin{aligned}
Q_\gamma^\pi(s,a) &= r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_\gamma^\pi(s') \\
&= r(s,a) + \gamma \sum_{s'} P(s'|s,a) \left(\frac{\rho(\pi)}{1-\gamma} + \bar{V}^\pi(s') + g^\pi(\gamma, s') \right) \\
&= r(s,a) + \frac{\gamma \rho(\pi)}{1-\gamma} + \gamma \sum_{s'} P(s'|s,a) \left(\bar{V}^\pi(s') + g^\pi(\gamma, s') \right) \\
&= r(s,a) - \rho(\pi) + \frac{\rho(\pi)}{1-\gamma} + \sum_{s'} P(s'|s,a) \bar{V}^\pi(s') \\
&\quad - (1-\gamma) \sum_{s'} P(s'|s,a) \bar{V}^\pi(s') + \gamma \sum_{s'} P(s'|s,a) g^\pi(\gamma, s') \\
&= \bar{Q}^\pi(s,a) + \frac{\rho(\pi)}{1-\gamma} - (1-\gamma) \sum_{s'} P(s'|s,a) \bar{V}^\pi(s') + \gamma \sum_{s'} P(s'|s,a) g^\pi(\gamma, s')
\end{aligned}$$

where we used Proposition 2.3 for the second equality. Note that the last two terms in the last equality approach 0 as $\gamma \to 1$, rearranging the terms and taking the limit for $\gamma \to 1$ gives us Equation (2.30).

We can then similarly rewrite (2.30) as

$$Q_\gamma^\pi(s,a) = \frac{\rho(\pi)}{1-\gamma} + \bar{Q}^\pi(s,a) + h(\gamma, s, a) \tag{2.33}$$

with $\lim_{\gamma \to 1} h(\gamma, s, a) = 0$. This allows us to rewrite the discounted advantage function as

$$\begin{aligned} A_\gamma^\pi(s,a) &= Q_\gamma^\pi(s,a) - V_\gamma^\pi(s) \\ &= \bar{Q}^\pi(s,a) + \frac{\rho(\pi)}{1-\gamma} + h^\pi(s,a,\gamma) - \bar{V}^\pi(s) - \frac{\rho(\pi)}{1-\gamma} - g^\pi(s,\gamma) \\ &= \bar{A}^\pi(s,a) + h^\pi(s,a,\gamma) - g^\pi(s,\gamma) \end{aligned}$$

Since $h^\pi(s,a,\gamma)$ and $g^\pi(s,\gamma)$ both approach 0 as γ approaches 1, taking the limit for $\gamma \to 1$ gives us Equation (2.31). □

2.8 Blackwell Optimality

A direct consequence of Proposition 2.3 is

$$\rho(\pi) = \lim_{\gamma \to 1} (1-\gamma)\rho_\gamma(\pi) \tag{2.34}$$

meaning that the the average reward and the discounted reward objectives are equivalent in the limit. Therefore an equivalent formulation to maximizing the average reward objective would be to instead solve the optimization problem $\operatorname{argmax}_\pi \lim_{\gamma \to 1}(1-\gamma)\rho_\gamma(\pi)$ using the discounted objective.

However this optimization problem contains a limit and is difficult to solve in practice. What is often done in practice is to instead maximize $(1-\gamma)\rho_\gamma(\pi)$ for increasingly larger discount factors, i.e. we switch the order of the limit and the argmax operator $\lim_{\gamma \to 1} \operatorname{argmax}_\pi \lim_{\gamma \to 1}(1-\gamma)\rho_\gamma(\pi)$ [Naik et al. 2019]. In fact, this limiting policy has strong theoretical implications which relate the average and discounted reward criteria.

Definition 2.5. A stationary policy π is said to be *Blackwell optimal* if it is simultaneously optimal for all $\rho_\gamma(\pi)$ with $\gamma \in (\bar{\gamma}, 1)$, where $\bar{\gamma} \in (0,1)$.

In short, a Blackwell optimal policy is optimal w.r.t. $\rho_\gamma(\pi)$ for all sufficiently large discount factors. Blackwell [1962] showed that a Blackwell optimal policy always exits, and that a Blackwell optimal policy is also optimal w.r.t. the average reward. Intuitively, a large discount factor corresponds to a longer time horizon[4], Blackwell optimality theory says that when a policy is optimal for a sufficiently long time horizon, it is also optimal for the average reward.

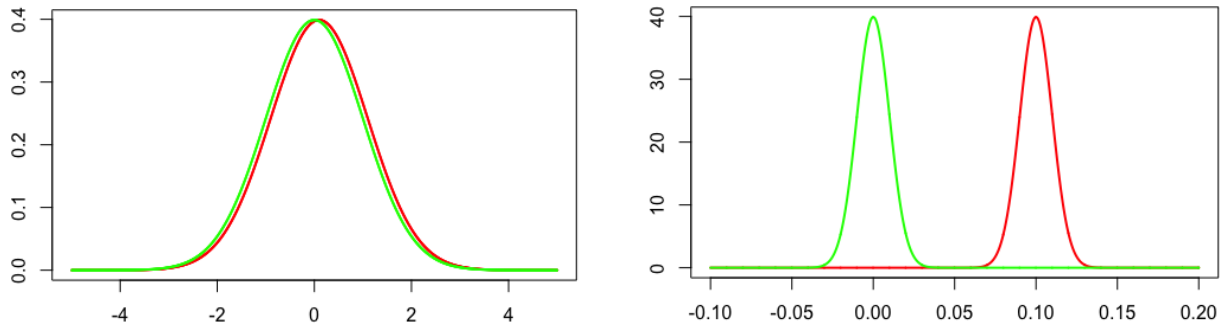
[4]In literature, the term $\frac{1}{1-\gamma}$ is known as the *effective horizon* [Agarwal et al. 2019]

3 | Local Policy Search

3.1 Introduction

One major source of difficulty with modern on-policy DRL algorithms lies in controlling the step-size for policy updates. For on-policy policy update procedure (and to a large extent this applies to off-policy algorithms as well), the quality of data acquired by the agent depends on the quality of the behavioral policy. Therefore an ill-chosen step-size could potentially result in bad behavioral policies which will subsequently affect the trajectories experienced by the agent. This could have potential downstreams effects which makes it very difficult for the agent to recover from suboptimal regions of the policy space. In addition, measuring the "distance" between different policies can be difficult and naïvely using the Euclidean distance between the policy parameters often does not respect the true "distance" between policies. This can be illustrated with the following simple example in Figure 3.1, the two distributions in both Figure 3.1(a) and 3.1(b) have the same Euclidean distance but it is clear by simple inspection that the two distributions in Figure 3.1(b) are further apart.

In order to have better control over step-sizes, Schulman et al. [2015] constructed a lower bound on the difference between the expected discounted return for two arbitrary policies π and π' by building upon the work of Kakade and Langford [2002]. The bound is a function of the divergence between these two policies and the discount factor. Schulman et al. [2015] showed that iteratively maximizing this lower bound generates a sequence of monotonically improved



(a) $\mu_1 = 0, \sigma_1 = 1$ and $\mu_2 = 0.1, \sigma_2 = 1$ **(b)** $\mu_1 = 0, \sigma_1 = 0.01$ and $\mu_2 = 0.1, \sigma_2 = 0.01$

Figure 3.1: Comparing Gaussian distributions with identical Euclidean distance in the parameter space.

policies for their discounted return. This bound was later refined by Achiam et al. [2017].

One important open question is whether these policy improvement bounds can be applied to non-discounted settings, which makes up an important set of reinforcement learning tasks. Broadly speaking, modern RL tackles two kinds of problems: *episodic tasks* and *continuing tasks*. In episodic tasks, the agent-environment interaction can be broken into separate distinct episodes, and the performance of the agent is simply the sum of the rewards accrued within an episode. Examples of episodic tasks include training an agent to learn to play Go [Silver et al. 2016, 2018], where the episode terminates when the game ends. In continuing tasks, such as robotic locomotion [Peters and Schaal 2008b; Schulman et al. 2015; Haarnoja et al. 2018] or in a queuing scenario [Tadepalli and Ok 1994; Sutton and Barto 2018], there is no natural separation of episodes and the agent-environment interaction continues indefinitely. The performance of an agent in a continuing task is more difficult to quantify since the total sum of rewards is typically infinite.

One way of making the long-term reward objective meaningful for continuing tasks is to apply *discounting* so that the infinite-horizon return is guaranteed to be finite for any bounded reward function. However the discounted objective biases the optimal policy to choose actions that lead to high near-term performance rather than to high long-term performance. Such an objective is not appropriate when the goal is to optimize long-term behavior, i.e., when the natural objective underlying the task at hand is non-discounted. In particular, we note that for the vast

majority of benchmarks for reinforcement learning such as Atari games [Mnih et al. 2013] and MuJoCo [Todorov et al. 2012], a non-discounted performance measure is used to evaluate the trained policies.

Although in many circumstances where the non-discounted criteria are more natural, most of the successful DRL algorithms today have been designed to optimize a discounted criterion during training. One possible work-around for this mismatch is to simply train with a discount factor that is very close to one. Indeed, from Blackwell optimality theory [Blackwell 1962] (see also Section 2.8), we know that if the discount factor is very close to one, then an optimal policy for the infinite-horizon discounted criterion is also optimal for the long-run average-reward criterion. However, although Blackwell's result suggests we can simply use a large discount factor to optimize non-discounted criteria, problems with large discount factors are in general more difficult to solve [Petrik and Scherrer 2008; Jiang et al. 2015, 2016; Lehnert et al. 2018]. Researchers have also observed that state-of-the-art DRL algorithms typically break down when the discount factor gets too close to one [Schulman et al. 2016; Andrychowicz et al. 2020].

In this chapter and the next, we seek to develop algorithms for finding high-performing policies for average-reward DRL problems. Instead of trying to simply use standard discounted DRL algorithms with large discount factors, we instead attack the problem head-on, seeking to directly optimize the average-reward criterion. While the average reward setting has been extensively studied in the classical Markov Decision Process literature [Howard 1960; Blackwell 1962; Veinott 1966; Bertsekas et al. 1995], and has to some extent been studied for tabular RL [Schwartz 1993; Mahadevan 1996; Abounadi et al. 2001; Wan et al. 2020], it has received relatively little attention in the DRL community. Here, our focus is on developing average-reward on-policy DRL algorithms.

We will begin by showing that the policy improvement theorem from Schulman et al. [2015] and Achiam et al. [2017] results in a non-meaningful bound in the average reward case. We then derive a novel result which lower bounds the difference of the average long-run rewards. The

bound depends on the average divergence between the policies and on the so-called Kemeny constant, which measures to what degree the irreducible Markov chains associated with the policies are "well-mixed". We show that iteratively maximizing this lower bound guarantees monotonic average reward policy improvement. In the next chapter, we demonstrate how these bounds can be applied to practical algorithms.

3.2 Background

3.2.1 Policy Gradient Methods

In general, reinforcement learning algorithms can be classified as *value-based* and *policy-based*[1]. A value-based algorithms uses estimates of the value function to determine the appropriate policy, examples of which include Q-Learning [Watkins 1989] and SARSA [Rummery and Niranjan 1994]. In constrast, policy-based methods learn the policy directly either with or without a value function. One popular approach is to directly apply stochastic gradient ascent to the performance objective function of interest. More concretely, given the discounted objective $\rho_\gamma(\pi_\theta)$ where π_θ denotes a policy parameterized by θ, the gradient of $\rho_\gamma(\pi_\theta)$ evaluated at a specific $\theta = \theta_k$ can be shown to be [Williams 1992; Sutton et al. 1999]:

$$\nabla_\theta \rho_\gamma(\pi_{\theta_k}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_{\theta_k}(a_t|s_t) A_\gamma^{\pi_{\theta_k}}(s_t, a_t) \right]. \tag{3.1}$$

This result is known as the *policy gradient theorem*. We can approximate (3.1) by sampling N trajectories of length T from π_{θ_k}:

$$\nabla_\theta \rho_\gamma(\pi_{\theta_k}) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_{\theta_k}(a_{it}|s_{it}) A_\gamma^{\pi_{\theta_k}}(s_{it}, a_{it}). \tag{3.2}$$

[1] An algorithm can also be both value-based and policy-based, i.e. actor-critic algorithms.

For the average reward setting, the policy gradient takes a very similar form [Sutton et al. 1999]:

$$\nabla_\theta \rho_\gamma(\pi_{\theta_k}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\theta_k}} \left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_{\theta_k}(a_t|s_t) \bar{A}^{\pi_{\theta_k}}(s_t, a_t) \right]. \tag{3.3}$$

However note here that the advantage function is now the average reward advantage function.

3.2.2 Local Policy Search for Discounted Problems

The policy gradient problem in deep reinforcement learning (DRL) can be defined as seeking a parameterized policy with high expected reward. An issue with policy gradient methods is poor sample efficiency [Kakade 2003; Schulman et al. 2015; Wang et al. 2017; Wu et al. 2017; Schulman et al. 2017b]. In algorithms such as REINFORCE [Williams 1992], new samples are needed for every gradient step. When generating samples is expensive (such as robotic environments), sample efficiency is of central concern.

Thus, given the current policy and a fixed number of trajectories (samples) generated, the goal of the sample efficiency problem is to construct a new policy with the highest performance improvement possible. To do so, it is desirable to limit the search to policies that are close to the original policy π_{θ_k} [Kakade 2001b; Schulman et al. 2015; Wu et al. 2017; Achiam et al. 2017; Schulman et al. 2017b; Tangkaratt et al. 2018]. Intuitively, if the candidate new policy π_θ is far from the original policy π_{θ_k}, it may not perform better than the original policy because too much emphasis is being placed on the relatively small batch of new data generated by π_{θ_k}, and not enough emphasis is being placed on the relatively large amount of data and effort previously used to construct π_{θ_k}. Concretely, we iteratively update policies by maximizing them within a local region, i.e., at iteration k we find a policy $\pi_{\theta_{k+1}}$ by solving the following optimization problem:

$$\begin{aligned} \underset{\pi_\theta \in \Pi_\theta}{\text{maximize}} \quad & J(\pi_\theta) \\ \text{subject to} \quad & D(\pi_\theta, \pi_{\theta_k}) \leqslant \delta \end{aligned} \tag{3.4}$$

where $J(\pi_\theta)$ is the policy performance metric we are interested in (e.g. including but not limited to $\rho_\gamma(\pi_\theta)$ or $\rho(\pi_\theta)$), D is some divergence measure. Different choices of D results in different algorithm, for example:

- Vanilla Policy Gradient/Gradient Ascent [Sutton et al. 1999; Peters and Schaal 2008b]

$$D(\pi_\theta, \pi_{\theta_k}) = \|\theta - \theta_k\|_2$$

  where $\|\cdot\|_2$ denote the Euclidean norm.

- Newton's method [Furmston et al. 2016]

$$D(\pi_\theta, \pi_{\theta_k}) = (\theta - \theta_k)^T H(\theta_k)(\theta - \theta_k)$$

  where $H(\theta_k)$ is the Hessian matrix at $\theta = \theta_k$.

- Natural Policy Gradient (NPG) [Kakade 2001b]/Trust Region Policy Optimization (TRPO) [Schulman et al. 2015]

$$D(\pi_\theta, \pi_{\theta_k}) = D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta_k}) \approx (\theta - \theta_k)^T F(\theta_k)(\theta - \theta_k)$$

  where $F(\theta_k)$ is the Fisher Information Matrix (FIM) at $\theta = \theta_k$.

- Mirror Descent [Montgomery and Levine 2016; Tomar et al. 2020]

$$D(\pi_\theta, \pi_{\theta_k}) = D_B(\pi_\theta, \pi_{\theta_k})$$

  where D_B is the Bregman divergence.

By using different choices of D and δ, this approach allows us to control the step-size of each update, which can lead to better sample efficiency [Peters and Schaal 2008b].

In this chapter we will focus on the case of D is the KL divergence, which we will subsequently refer to as the NPG/TRPO problem[2]. Schulman et al. [2015] showed that this particular case, the performance difference between two successive policies π_{k+1} and π_k can be upper-bounded:

$$\rho_\gamma(\pi_{k+1}) - \rho_\gamma(\pi_k) \geqslant \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_k,\gamma} \\ a\sim\pi_{k+1}}} [A_\gamma^{\pi_k}(s,a)] - \frac{4\epsilon\gamma}{(1-\gamma)^2} \cdot \max_s [D_{\mathrm{TV}}(\pi_{k+1} \parallel \pi_k)[s]] \tag{3.5}$$

where $D_{\mathrm{TV}}(\pi' \parallel \pi)[s] := \frac{1}{2}\sum_a |\pi'(a|s) - \pi(a|s)|$ is the *total variation divergence*[3], and ϵ is some constant. Schulman et al. [2015] showed that by choosing π_{k+1} which maximizes the right hand side of (3.5), we are guaranteed to have $\rho_\gamma(\pi_{k+1}) \geqslant \rho_\gamma(\pi_k)$. This provided the theoretical foundation for an entire class of on-policy DRL algorithms [Schulman et al. 2015, 2017b; Wu et al. 2017; Song et al. 2020; Tomar et al. 2020].

A natural question arises here is whether the iterative procedure described by Schulman et al. [2015] also guarantees improvement for the average reward. Since the discounted and average reward objectives become equivalent as $\gamma \to 1$, one may conjecture that we can also lower bound the policy performance difference of the average reward objective by simply letting $\gamma \to 1$ for the bounds in Schulman et al. [2015]. Unfortunately this results in a non-meaningful bound.

Proposition 3.1. *Consider the bounds in Theorem 1 of Schulman et al. [2015] and Corollary 1 of Achiam et al. [2017]. The right hand side of both bounds times* $(1-\gamma)$ *goes to negative infinity as* $\gamma \to 1$.

Proof. We will give a proof for the case of Corollary 1 in Achiam et al. [2017], a similar argument can be applied to the bound in Theorem 1 of Schulman et al. [2015].

We first state Corollary 1 of Achiam et al. [2017] which says that for any two stationary policies

[2] In the next chapter we will expand upon how NPG and TRPO are related.

[3] Even though the TV divergence is used in the performance difference bound, the KL divergence is often used in practice via Pinsker's inequality (see Section 3.3 for more details)

π and π':

$$\rho_\gamma(\pi') - \rho_\gamma(\pi) \geqslant \frac{1}{1-\gamma}\left[\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi,\gamma}\\ a\sim\pi'}}[A^\pi_\gamma(s,a)] - \frac{2\gamma\epsilon^\gamma}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d_{\pi,\gamma}} D_{\mathrm{TV}}(\pi' \parallel \pi)\right] \tag{3.6}$$

where $\epsilon^\gamma = \max_s \left|\mathbb{E}_{a\sim\pi'}[A^\pi_\gamma(s,a)]\right|$. Since $d_{\pi,\gamma}$ approaches the stationary distribution d_π as $\gamma \to 1$, we can multiply the right hand side of (4.1) by $(1-\gamma)$ and take the limit which gives us:

$$\begin{aligned}
&\lim_{\gamma\to 1}\left(\mathop{\mathbb{E}}_{\substack{s\sim d_{\pi,\gamma}\\ a\sim\pi'}}[A^\pi_\gamma(s,a)] \pm \frac{2\gamma\epsilon^\gamma}{1-\gamma}\mathop{\mathbb{E}}_{s\sim d_{\pi,\gamma}} D_{\mathrm{TV}}(\pi' \parallel \pi)\right)\\
&= \mathop{\mathbb{E}}_{\substack{s\sim d_\pi\\ a\sim\pi'}}[\bar{A}^\pi(s,a)] - 2\epsilon \mathop{\mathbb{E}}_{s\sim d_\pi}[D_{\mathrm{TV}}(\pi' \parallel \pi)]\lim_{\gamma\to 1}\frac{\gamma}{1-\gamma}\\
&= -\infty
\end{aligned}$$

Here $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'}[\bar{A}^\pi(s,a)]\right|$. The first equality is a direct result of Corollary 2.4. □

Since $\lim_{\gamma\to 1}(1-\gamma)(\rho_\gamma(\pi') - \rho_\gamma(\pi)) = \rho(\pi') - \rho(\pi)$, Proposition 3.1 says that the policy improvement guarantee from Schulman et al. [2015] and Achiam et al. [2017] becomes trivial when $\gamma \to 1$ and thus does not generalize to the average reward setting. In the next section, we will derive a novel policy improvement bound for the average reward objective, which in turn can be used to generate monotonically improved policies w.r.t. the average reward.

3.3 Policy Improvement Theorem for the Average Reward Criterion

Suppose we have a new policy π' obtained via some update rule from the current policy π. Similar to the discounted case, we would like to measure their performance difference $\rho(\pi') - \rho(\pi)$ using

an expression which depends on π and some divergence metric between the two policies. The following identity shows that $\rho(\pi') - \rho(\pi)$ can be expressed using the average reward advantange function of π.

Lemma 3.2. *For any aperiodic unichain MDP:*

$$\rho(\pi') - \rho(\pi) = \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi'} \\ a\sim\pi'}} \left[\bar{A}^{\pi}(s,a)\right] \tag{3.7}$$

for any two stochastic policies π and π'.

Lemma 3.2 is an extension of the well-known policy difference lemma from Kakade and Langford [2002] to the average reward case. A similar result was proven by Even-Dar et al. [2009] and Neu et al. [2010]. For completeness, we provide two simple proofs.

Proof. In the first approach, we directly expand the right-hand side using the definition of the advantage function and Bellman equation, which gives us:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}} \left[\bar{A}^{\pi}(s,a)\right] &= \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}} \left[\bar{Q}^{\pi}(s,a) - \bar{V}^{\pi}(s)\right] \\
&= \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi'}} \left[r(s,a) - \rho(\pi) + \mathop{\mathbb{E}}_{s'\sim P(\cdot|s,a)} \left[\bar{V}^{\pi}(s')\right] - \bar{V}^{\pi}(s)\right] \\
&= \rho(\pi') - \rho(\pi) + \mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi' \\ s'\sim P(\cdot|s,a)}} \left[\bar{V}^{\pi}(s')\right] - \mathop{\mathbb{E}}_{s\sim d^{\pi'}} \left[\bar{V}^{\pi}(s)\right]
\end{aligned}$$

Since $d_{\pi'}(s)$ is the stationary distribution:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'} \\ a\sim\pi' \\ s'\sim P(\cdot|s,a)}} \left[\bar{V}^{\pi}(s')\right] &= \sum_s d_{\pi'}(s) \sum_a \pi'(a|s) \sum_{s'} P(s'|s,a) \bar{V}^{\pi}(s') \\
&= \sum_s d_{\pi'}(s) \sum_{s'} P_{\pi'}(s'|s) \bar{V}^{\pi}(s') = \sum_{s'} d_{\pi'}(s') \bar{V}^{\pi}(s')
\end{aligned}$$

Therefore,

$$\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'}\\ a\sim\pi'\\ s'\sim P(\cdot|s,a)}}[\bar{V}^{\pi}(s')] - \mathop{\mathbb{E}}_{s\sim d^{\pi'}}[\bar{V}^{\pi}(s)] = 0$$

which gives us the desired result.

Alternatively, we can directly apply Proposition 2.3 and Corollary 2.4 to Lemma 6.1 of [Kakade and Langford 2002] and take the limit as $\gamma \to 1$. □

Note that the expression (3.7) depends on samples drawn from π'. However we can show through the following lemma that when d_π and $d_{\pi'}$ are "close" w.r.t. the TV divergence, we can evaluate the right hand side of (3.7) using samples from d_π.

Lemma 3.3. *For any aperiodic unichain MDP, the following bound holds for any two stochastic policies π and π':*

$$\left|\rho(\pi') - \rho(\pi) - \mathop{\mathbb{E}}_{\substack{s\sim d_\pi\\ a\sim\pi'}}\left[\bar{A}^{\pi}(s,a)\right]\right| \leqslant 2\epsilon D_{TV}(d_{\pi'} \parallel d_\pi) \tag{3.8}$$

where $\epsilon = \max_s \left|\mathbb{E}_{a\sim\pi'(a|s)}[\bar{A}^{\pi}(s,a)]\right|$.

Proof.

$$\begin{aligned}
\left|\rho(\pi') - \rho(\pi) - \mathop{\mathbb{E}}_{\substack{s\sim d_\pi\\ a\sim\pi'}}\left[\bar{A}^{\pi}(s,a)\right]\right| &= \left|\mathop{\mathbb{E}}_{\substack{s\sim d^{\pi'}\\ a\sim\pi'}}\left[\bar{A}^{\pi}(s,a)\right] - \mathop{\mathbb{E}}_{\substack{s\sim d_\pi\\ a\sim\pi'}}\left[\bar{A}^{\pi}(s,a)\right]\right| \\
&= \left|\sum_s \mathop{\mathbb{E}}_{a\sim\pi'}\left[\bar{A}^{\pi}(s,a)\right](d_{\pi'}(s) - d_\pi(s))\right| \\
&\leqslant \sum_s \left|\mathop{\mathbb{E}}_{a\sim\pi'}\left[\bar{A}^{\pi}(s,a)\right](d_{\pi'}(s) - d_\pi(s))\right| \\
&\leqslant \max_s \left|\mathop{\mathbb{E}}_{a\sim\pi'}\left[\bar{A}^{\pi}(s,a)\right]\right| \|d_{\pi'} - d_\pi\|_1 \\
&= 2\epsilon D_{\mathrm{TV}}(d^{\pi'} \parallel d^{\pi})
\end{aligned}$$

where the last inequality follows from Hölder's inequality. □

Lemma 3.3 implies that

$$\rho(\pi') \approx \rho(\pi) + \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi'}} \left[\bar{A}^\pi(s,a)\right] \tag{3.9}$$

when d_π and $d_{\pi'}$ are "close". However in order to study how policy improvement is connected to changes in the actual policies themselves, we need to analyze the relationship between changes in the policies and changes in stationary distributions. It turns out that the sensitivity of the stationary distributions in relation to the policies is related to the structure of the underlying Markov chain via Kemeny's constant, which was introduced in Chapter 2.3.

The following result connects the sensitivity of the stationary distribution to changes to the policy.

Lemma 3.4. *For any ergodic MDP, the divergence between the stationary distributions d_π and $d_{\pi'}$ can be upper bounded by the average divergence between policies π and π':*

$$D_{TV}(d_{\pi'} \parallel d_\pi) \leqslant (\kappa^* - 1) \mathop{\mathbb{E}}_{s\sim d_\pi} [D_{TV}(\pi' \parallel \pi)[s]] \tag{3.10}$$

where $\kappa^* = \max_\pi \kappa^\pi$

Proof. Our proof is based on Markov chain perturbation theory [Cho and Meyer 2001; Hunter 2005]. Note first that

$$\begin{aligned}(d_{\pi'}^T - d_\pi^T)(I - P_{\pi'} + P_{\pi'}^*) &= d_{\pi'}^T - d_\pi^T - d_{\pi'}^T + d_\pi^T P_{\pi'} \\ &= d_\pi^T P_{\pi'} - d_\pi^T \\ &= d_\pi^T (P_{\pi'} - P_\pi)\end{aligned} \tag{3.11}$$

Right multiplying (3.11) by $(I - P_{\pi'} + P_{\pi'}^*)^{-1}$ gives us:

$$d_{\pi'}^T - d_\pi^T = d_\pi^T (P_{\pi'} - P_\pi)(I - P_{\pi'} + P_{\pi'}^*)^{-1} \tag{3.12}$$

Recall that $Z^{\pi'} = (I - P_{\pi'} + P^*_{\pi'})^{-1}$ and $M^{\pi'} = (I - Z^{\pi'} + EZ^{\pi'}_{\text{dg}})D^{\pi'}$. Rearranging the terms we find that

$$Z^{\pi'} = I + EZ^{\pi'}_{\text{dg}} - M^{\pi'}(D^{\pi'})^{-1} \tag{3.13}$$

Plugging (3.13) into (3.12) gives us

$$\begin{aligned} d^T_{\pi'} - d^T_{\pi} &= d^T_{\pi}(P_{\pi'} - P_{\pi})(I + EZ^{\pi'}_{\text{dg}} - M^{\pi'}(D^{\pi'})^{-1}) \\ &= d^T_{\pi}(P_{\pi'} - P_{\pi})(I - M^{\pi'}(D^{\pi'})^{-1}) \end{aligned} \tag{3.14}$$

where the last equality is due to $(P_{\pi'} - P_{\pi})E = 0$.

Let $\|\cdot\|_p$ denote the the operator norm of a matrix, in particular $\|\cdot\|_1$ and $\|\cdot\|_\infty$ are the maximum absolute column sum and maximum absolute row sum respectively. By the submultiplicative property of operator norms [Horn and Johnson 2012], we have:

$$\begin{aligned} \|d_{\pi'} - d_{\pi}\|_1 &= \left\|(I - M^{\pi'}(D^{\pi'})^{-1})^T(P^T_{\pi'} - P^T_{\pi})d_{\pi}\right\|_1 \\ &\leqslant \left\|(I - M^{\pi'}(D^{\pi'})^{-1})^T\right\|_1 \left\|(P^T_{\pi'} - P^T_{\pi})d_{\pi}\right\|_1 \\ &= \left\|(I - M^{\pi'}(D^{\pi'})^{-1})\right\|_\infty \left\|(P^T_{\pi'} - P^T_{\pi})d_{\pi}\right\|_1 \end{aligned} \tag{3.15}$$

We can rewrite $\left\|I - M^{\pi'}(D^{\pi'})^{-1}\right\|_\infty$ as

$$\begin{aligned} \left\|I - M^{\pi'}(D^{\pi'})^{-1}\right\|_\infty &= \max_s \left(\sum_{s'} M^{\pi'}(s, s')d_{\pi'}(s') - 1\right) \\ &= \kappa^{\pi'} - 1 \end{aligned} \tag{3.16}$$

Finally we bound $\left\|(P_{\pi'}^T - P_\pi^T)d_\pi\right\|_1$ by

$$\begin{aligned}
\left\|(P_{\pi'}^T - P_\pi^T)d_\pi\right\|_1 &= \sum_{s'}\left|\sum_s\left(\sum_a P(s'|s,a)\pi'(a|s) - P(s'|s,a)\pi(a|s)\right)d_\pi(s)\right| \\
&\leqslant \sum_{s',s}\left|\sum_a P(s'|s,a)(\pi'(a|s) - \pi(a|s))\right| d_\pi(s) \\
&\leqslant \sum_{s,s',a} P(s'|s,a)\left|\pi'(a|s) - \pi(a|s)\right| d_\pi(s) \\
&\leqslant \sum_{s,a}\left|\pi'(a|s) - \pi(a|s)\right| d_\pi(s) \\
&= 2\mathop{\mathbb{E}}_{s\sim d^\pi}[D_{\mathrm{TV}}(\pi' \parallel \pi)]
\end{aligned} \tag{3.17}$$

Plugging back into (3.15) and setting $\kappa^* = \max_\pi \kappa^\pi$ gives the desired result. □

By Proposition 2.2, for Markov chains with a small mixing time, where an agent can quickly get to any state, Kemeny's constant is relatively small and Lemma 3.4 shows that the stationary distributions are not highly sensitive to small changes in the policy. On the other hand, for Markov chains that that have high mixing times, the factor can become very large. In this case Lemma 3.4 shows that small changes in the policy can have a large impact on the resulting stationary distributions.

Combining the bounds in Lemma 3.3 and Lemma 3.4 gives us the following result:

Theorem 3.5. *For any ergodic MDP, the following bounds hold for any two stochastic policies π and π', :*

$$\rho(\pi') - \rho(\pi) \leqslant \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}\bar{A}^\pi(s,a)\right] + 2\xi \mathop{\mathbb{E}}_{s\sim d_\pi}[D_{TV}(\pi' \parallel \pi)[s]] \tag{3.18}$$

$$\rho(\pi') - \rho(\pi) \geqslant \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi}}\left[\frac{\pi'(a|s)}{\pi(a|s)}\bar{A}^\pi(s,a)\right] - 2\xi \mathop{\mathbb{E}}_{s\sim d_\pi}[D_{TV}(\pi' \parallel \pi)[s]] \tag{3.19}$$

where $\xi = (\kappa^* - 1)\max_s \mathbb{E}_{a\sim\pi'}|\bar{A}^\pi(s,a)|$.

The bounds in Theorem 3.5 are guaranteed to be finite. Analogous to the discounted case, the multiplicative factor ξ provides theoretical guidance on the step-sizes for policy updates.

The bound in Theorem 3.5 is given in terms of the TV divergence; however the KL divergence is more commonly used in practice. The relationship between the TV divergence and KL divergence is given by Pinsker's inequality [Tsybakov 2008], which says that for any two distributions p and q: $D_{\mathrm{TV}}(p \parallel q) \leqslant \sqrt{D_{\mathrm{KL}}(p\|q)/2}$. We can then show that

$$\begin{aligned} \mathop{\mathbb{E}}_{s\sim d_\pi}[D_{\mathrm{TV}}(\pi' \parallel \pi)[s]] &\leqslant \mathop{\mathbb{E}}_{s\sim d_\pi}[\sqrt{D_{\mathrm{KL}}(\pi'\|\pi)[s]/2}] \\ &\leqslant \sqrt{\mathop{\mathbb{E}}_{s\sim d_\pi}[D_{\mathrm{KL}}(\pi'\|\pi)][s]]/2} \end{aligned} \tag{3.20}$$

where the second inequality comes from Jensen's inequality. The inequality in (3.20) shows that the bounds in Theorem 3.5 still hold when $\mathbb{E}_{s\sim d_\pi}[D_{\mathrm{TV}}(\pi' \parallel \pi)[s]]$ is substituted with $\sqrt{\mathbb{E}_{s\sim d_\pi}[D_{\mathrm{KL}}(\pi'\|\pi)][s]/2}$. This parallels the bound for the discounted case introduced in [Achiam et al. 2017].

Note that Theorem holds for any ergodic MDP; we can also show that a similar result can be derived for the more general aperiodic unichain case.

For an aperioidic unichain MDP, we can similarly bound the sensitivity of the stationary distribution to changes in the policy.

Lemma 3.6. *For any aperiodic unichain MDP:*

$$D_{TV}(d_{\pi'} \parallel d_\pi) \leqslant \zeta^* \mathop{\mathbb{E}}_{s\sim d_\pi}[D_{TV}(\pi' \parallel \pi)[s]] \tag{3.21}$$

where $\zeta^* = \max_\pi \|Z^\pi\|_\infty$.

Proof. Note that

$$d^T_{\pi'} - d^T_\pi = d^T_\pi (P_{\pi'} - P_\pi)(I - P_{\pi'} + P^*_{\pi'})^{-1}$$

from Equation 3.12 still holds in the general aperiodic unichain case. By the submultiplicative

property, we have:

$$\begin{aligned}\|d_{\pi'} - d_\pi\|_1 &= \left\|((I - P_{\pi'} + P^*_{\pi'})^{-1})^T (P^T_{\pi'} - P^T_\pi) d_\pi\right\|_1 \\ &\leqslant \left\|((I - P_{\pi'} + P^*_{\pi'})^{-1})^T\right\|_1 \left\|(P^T_{\pi'} - P^T_\pi) d_\pi\right\|_1 \\ &= \left\|(I - P_{\pi'} + P^*_{\pi'})^{-1}\right\|_\infty \left\|(P^T_{\pi'} - P^T_\pi) d_\pi\right\|_1\end{aligned} \tag{3.22}$$

Using the same argument as (3.17) to bound $\left\|(P^T_{\pi'} - P^T_\pi) d_\pi\right\|_1$ and setting $\zeta^* = \max_\pi \|Z^\pi\|_\infty$ gives the desired result. □

Combining Lemma 3.3 and Lemma 3.6 gives us the following result:

Theorem 3.7. *For any aperiodic unichain MDP, the following bounds hold for any two stochastic policies π and π':*

$$\rho(\pi') - \rho(\pi) \leqslant \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi'}} \left[\bar{A}^\pi(s,a)\right] + 2\tilde{\xi} \mathop{\mathbb{E}}_{s\sim d_\pi} \left[D_{TV}(\pi' \,\|\, \pi)[s]\right] \tag{3.23}$$

$$\rho(\pi') - \rho(\pi) \geqslant \mathop{\mathbb{E}}_{\substack{s\sim d_\pi \\ a\sim\pi'}} \left[\bar{A}^\pi(s,a)\right] - 2\tilde{\xi} \mathop{\mathbb{E}}_{s\sim d_\pi} \left[D_{TV}(\pi' \,\|\, \pi)[s]\right] \tag{3.24}$$

where $\tilde{\xi} = \zeta^* \max_s \mathbb{E}_{a\sim\pi'} |\bar{A}^\pi(s,a)|$.

The constant ζ^* is always finite therefore we can similarly apply the approximate policy iteration procedure from Algorithm 1 to generate a sequence of monotonically improving policies.

3.4 Approximate Policy Iteration

Parallel to the discounted cases [Schulman et al. 2015; Achiam et al. 2017], one direct consequence of Theorem 3.5 is that iteratively maximizing the right hand side of (3.19) generates a monotonically improving sequence of policies w.r.t. the average reward objective. Algorithm 1 gives an approximate policy iteration algorithm that produces such a sequence of policies.

Algorithm 1 Approximate Average Reward Policy Iteration

Initialize: π_0

1: **for** $k = 0, 1, 2, \ldots$ **do**

2: Policy Evaluation: Evaluate $\bar{A}^{\pi_k}(s,a)$ for all s, a

3: Policy Improvement:

$$\pi_{k+1} = \operatorname*{argmax}_{\pi} \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_k} \\ a\sim\pi}} \left[\bar{A}^{\pi_k}(s,a)\right] - \xi\sqrt{2 \mathop{\mathbb{E}}_{s\sim d_{\pi_k}} \left[D_{\mathrm{KL}}\left(\pi \| \pi_k\right)[s]\right]} \tag{3.25}$$

where $\xi = (\kappa^* - 1)\max_s \mathbb{E}_{a\sim\pi}\left|\bar{A}^{\pi_k}(s,a)\right|$

Proposition 3.8. *Given an initial policy π_0, Algorithm 1 is guaranteed to generate a sequence of policies $\pi_1, \pi_2, \ldots$ such that $\rho(\pi_0) \leqslant \rho(\pi_1) \leqslant \rho(\pi_2) \leqslant \cdots$.*

Proof. At iteration k, $\mathbb{E}_{s\sim d_{\pi_k}, a\sim\pi}[\bar{A}^{\pi_k}(s,a)] = 0$, $\mathbb{E}_{s\sim d_{\pi_k}}[D_{\mathrm{KL}}(\pi\|\pi_k)[s]] = 0$ for $\pi = \pi_k$. By Theorem 3.5 and (3.25), $\rho(\pi_{k+1}) - \rho(\pi_k) \geqslant 0$. □

However, Algorithm 1 is difficult to implement in practice since it requires exact knowledge of $\bar{A}^{\pi_k}(s,a)$ and the transition matrix. Furthermore, calculating the term ξ is impractical for high-dimensional problems. In the next two chapters, we will introduce simple sample-based algorithms which approximates the update rule in Algorithm 1 which can be practically applied to problems with large state and action spaces.

3.5 Related Work

Dynamic programming algorithms for finding the optimal average reward policies have been well-studied [Howard 1960; Blackwell 1962; Veinott 1966]. Several tabular Q-learning-like algorithms for problems with unknown dynamics have been proposed, such as R-Learning [Schwartz 1993], RVI Q-Learning [Abounadi et al. 2001], CSV-Learning [Yang et al. 2016], and Differential Q-Learning [Wan et al. 2020]. Mahadevan [1996] conducted a thorough empirical analysis of the R-Learning algorithm. We note that much of the previous work on average reward RL focuses on

the tabular setting without function approximations, and the theoretical properties of many of these Q-learning-based algorithm are not well understood (in particular R-learning). More recently, POLITEX updates policies using a Boltzmann distribution over the sum of action-value function estimates of the previous policies [Abbasi-Yadkori et al. 2019] and Wei et al. [2020] introduced a model-free algorithm for optimizing the average reward of weakly-communicating MDPs.

For policy gradient methods, Baxter and Bartlett [2001] showed that if $1/(1-\gamma)$ is large compared to the mixing time of the Markov chain induced by the MDP, then the gradient of $\rho_\gamma(\pi)$ can accurately approximate the gradient of $\rho(\pi)$. Kakade [2001a] extended upon this result and provided an error bound on using an optimal discounted policy to maximize the average reward. In contrast, our work directly deals with the average reward objective and provides theoretical guidance on the optimal step size for each policy update.

Policy improvement bounds have been extensively explored in the discounted case. The results from Schulman et al. [2015] are extensions of Kakade and Langford [2002]. Pirotta et al. [2013] also proposed an alternative generalization to Kakade and Langford [2002]. Achiam et al. [2017] improved upon Schulman et al. [2015] by replacing the maximum divergence with the average divergence.

3.6 Conclusion

In this chapter, we introduced a novel policy improvement bound for the average reward criterion. The bound is based on the average divergence between two policies and Kemeny's constant or mixing time of the Markov chain. We show that previous existing policy improvement bounds for the discounted case results in a non-meaningful bound for the average reward objective. These results provides the theoretical justification and the means to generalize the popular trust-region based algorithms to the average reward setting. In the next chapter, we will apply these theoretical results to practical algorithms.

4 | Average Reward TRPO

4.1 Background

Formally our goal is to obtain the best possible $\pi_{\theta_{k+1}}$ in a parameterized policy class Π_θ given the current policy π_{θ_k} at iteration k. Recall first that in the discounted case, iteratively maximizing the right hand side of (4.1) generates a sequence of monotonically improving policies in w.r.t. the discounted criteria, i.e.

$$\pi_{\theta_{k+1}} = \underset{\pi_\theta \in \Pi_\theta}{\operatorname{argmax}} \frac{1}{1-\gamma} \left[\underset{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi_\theta}}{\mathbb{E}} \left[A_\gamma^{\pi_{\theta_k}}(s,a) \right] - \frac{\gamma\epsilon^\gamma}{1-\gamma} \sqrt{2 \underset{s\sim d_{\pi_{\theta_k},\gamma}}{\mathbb{E}} \left[D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right) \right]} \right] \tag{4.1}$$

where we replaced the TV divergence with the KL divergence using Pinsker's inequality. Similar to the average reward case (see Section 3.4), solving this unconstrained problem can be challenging and often infeasible [Schulman et al. 2015; Achiam 2017]. However, we note that when $D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right) \approx 0$, i.e. when π_θ and π_{θ_k} are close, we can approximate the performance difference using the following surrogate function:

$$\rho_\gamma(\pi_\theta) - \rho_\gamma(\pi_{\theta_k}) \approx \widetilde{J}_{\gamma,\pi_{\theta_k}}(\pi_\theta) := \frac{1}{1-\gamma} \left[\underset{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi_\theta}}{\mathbb{E}} \left[A_\gamma^{\pi_{\theta_k}}(s,a) \right] \right]. \tag{4.2}$$

It can also be easily shown that the surrogate function (4.2) matches $\rho_\gamma(\pi_\theta) - \rho_\gamma(\pi_{\theta_k})$ to the first order w.r.t. the parameter θ [Achiam 2017]. Using these facts, Schulman et al. [2015] proposed solving the following constrained version of Problem (4.1):

$$\begin{aligned} \underset{\pi_\theta \in \Pi_\theta}{\text{maximize}} \quad & \underset{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi_\theta}}{\mathbb{E}} \left[A_\gamma^{\pi_{\theta_k}}(s,a)\right] \\ \text{subject to} \quad & \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) \leqslant \delta. \end{aligned} \tag{4.3}$$

Note here that the objective is the surrogate function defined in (4.2) and $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) := \mathbb{E}_{s\sim d_{\pi_{\theta_k},\gamma}}[D_{\mathrm{KL}}(\pi_\theta \| \pi_{\theta_k})[s]]$. The constraint set $\{\pi_\theta \in \Pi_\Theta : \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) \leqslant \delta\}$ is called the *trust region set*. The step-size δ is treated as a hyperparamter in practice and should ideally be tuned for each specific task.

However for problems with high dimensional parameter spaces such as deep neural networks, solving (4.3) is highly impractical. We can approximate this problem by performing first-order Taylor approximation on the objective and second-order approximation on the KL constraint[1] around θ_k which gives us:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & g_\gamma^T(\theta - \theta_k) \\ \text{subject to} \quad & \frac{1}{2}(\theta - \theta_k)^T H_\gamma (\theta - \theta_k) \leqslant \delta \end{aligned} \tag{4.4}$$

where

$$g_\gamma := \underset{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi_{\theta_k}}}{\mathbb{E}} \left[\nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_k} A_\gamma^{\pi_{\theta_k}}(s,a)\right] \tag{4.5}$$

and

$$H_\gamma := \underset{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi_{\theta_k}}}{\mathbb{E}} \left[\nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_k} \nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_k}^T\right] \tag{4.6}$$

Note that this approximation is good provided that the step-size δ is small. The term g is the

[1] The gradient and first-order Taylor approximation of $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})$ at $\theta = \theta_k$ is zero.

discounted reward policy gradient at $\theta = \theta_k$ with an additional baseline term [Sutton et al. 1999] and H_γ is the *Fisher Information Matrix* (FIM) [Lehmann and Casella 2006] where we use the subscript γ to denote its dependency on the discount factor. The FIM is a symmetrical matrix and always positive semi-definite. If we assume H_γ is always positive definite, we can solve (4.4) analytically with a Lagrange duality argument which yields the solution:

$$\theta = \theta_k + \sqrt{\frac{2\delta}{g_\gamma^T H_\gamma^{-1} g}} H_\gamma^{-1} g \tag{4.7}$$

The update rule in (4.7) has the same form as that of natural policy gradients [Kakade 2001b] and both g_γ and H_γ can be approximated using samples drawn from the policy π_{θ_k}.

Up until now, the update rule for TRPO and NPG are identical. But one issue is that due to the use of Taylor approximations, the KL constraint may no longer be satisfied. Hence after applying the update term (4.7), we use backtracking linesearch to find an update term which has a positive advantage value and also maintains KL constraint satisfaction. We also apply the conjugate gradient method [Strang 2007] to estimate H_γ^{-1}.

In the next several sections, we will focus on how to convert the TRPO problem to the average reward setting. At the end of this chapter, we will also demonstrate that the non-discounted version of the TRPO algorithm significantly outperforms its discounted variants on a set of high dimensional continuing control tasks.

4.2 Trust Region Methods for the Average Reward Criterion

Following a similar line of logic to the discounted case, we can rewrite the unconstrained optimization problem in (3.25) as a constrained problem:

$$\begin{aligned} \underset{\pi_\theta \in \Pi_\theta}{\text{maximize}} \quad & \underset{\substack{s\sim d_{\pi_{\theta_k}} \\ a\sim\pi_\theta}}{\mathbb{E}} \left[\bar{A}^{\pi_{\theta_k}}(s,a)\right] \\ \text{subject to} \quad & \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) \leqslant \delta. \end{aligned} \tag{4.8}$$

Here $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) := \mathbb{E}_{s\sim d_{\pi_{\theta_k}}}\left[D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s]\right]$ where s is drawn from the stationary distribution (as opposed to $d_{\pi_{\theta_k},\gamma}$ in the discounted case). Importantly, the advantage function $\bar{A}^{\pi_{\theta_k}}(s,a)$ appearing in (4.8) is the average-reward advantage function, defined as the bias minus the action-bias, and not the discounted advantage function. Another crucial difference in the average reward setting is that the choice of step-size is related to the mixing time of the underlying Markov chain (since it is related to the multiplicative factor ξ in Theorem 3.5). When the mixing time is small, a larger step-size can be chosen and vice versa. While it is impractical to calculate the optimal step-size, in certain applications domain knowledge on the mixing time can be used to serve as a guide for tuning δ.

Similar to the discounted case [Schulman et al. 2015; Achiam et al. 2017]. When we set $\pi_{\theta_{k+1}}$ to be the optimal solution to (4.8), the policy improvement guarantee no longer holds. However we can show that $\pi_{\theta_{k+1}}$ has the following worst-case performance degradation guarantee:

Proposition 4.1. *Let $\pi_{\theta_{k+1}}$ be the optimal solution to (4.8) for some $\pi_{\theta_k} \in \Pi_\theta$. The policy performance difference between $\pi_{\theta_{k+1}}$ and π_{θ_k} can be lower bounded by*

$$\rho(\pi_{\theta_{k+1}}) - \rho(\pi_{\theta_k}) \geqslant -\xi^{\pi_{\theta_{k+1}}}\sqrt{2\delta} \tag{4.9}$$

where $\xi^{\pi_{\theta_{k+1}}} = (\kappa^{\pi_{\theta_{k+1}}} - 1) \max_s \mathbb{E}_{a\sim\pi_{\theta_{k+1}}} |\bar{A}^{\pi_{\theta_k}}(s,a)|$.

Proof. Since $\bar{D}_{\mathrm{KL}}(\pi_{\theta_k} \,\|\, \pi_{\theta_k}) = 0$, π_{θ_k} is feasible. The objective value is 0 for $\pi_\theta = \pi_{\theta_k}$. The bound follows from (3.19) and (3.20) where the average KL is bounded by δ. □

Several algorithms have been proposed for efficiently solving the discounted version of (4.8) [Schulman et al. 2015; Wu et al. 2017; Tangkaratt et al. 2018; Song et al. 2020]. These algorithms can also be adapted for the average reward case and are theoretically justified via Theorem 3.5 and Proposition 4.1. In the next section, we will provide as a specific example how this can be done for one such algorithm.

4.3 Average Reward TRPO

In this section, we introduce ATRPO, which is an average-reward modification of TRPO algorithm [Schulman et al. 2015]. Similar to TRPO, we apply Taylor approximations to (4.8). This gives us a new optimization problem which can be solved exactly using Lagrange duality. The solution to this approximate problem gives an explicit update rule for the policy parameters which then allows us to perform policy updates using an actor-critic framework. The algorithm is similar to TRPO in the discount case but with several notable distinctions.

Algorithm 2 provides a basic outline of ATRPO.

The major differences between ATRPO and TRPO are as follows:

(i) The critic network in Algorithm 2 approximates the average-reward bias rather than the discounted value function.

(ii) ATRPO must estimate the average return ρ of the current policy.

(iii) The target for the bias function and the advantage are calculated without discount factors and the average return ρ is subtracted from the reward. Simply setting the discount factor to 1 in TRPO does not lead to Algorithm 2.

Algorithm 2 Average Reward TRPO (ATRPO)

Initialize: Policy parameters θ_0, critic net parameters ϕ_0, learning rate α, trajectory truncation parameter N.

for $k = 0, 1, 2, \cdots$ **do**

  Collect a truncated trajectory $\{s_t, a_t, s_{t+1}, r_t\}_{t=1:N}$ from the environment using π_{θ_k}.

  Calculate sample average reward of π_{θ_k} via $\rho = \frac{1}{N}\sum_{t=1}^{N} r_t$.

  for $t = 1, 2, \ldots, N$ **do**

    Get target

$$\bar{V}_t^{\text{target}} = r_t - \rho + \bar{V}_{\phi_k}(s_{t+1})$$

    Get advantage estimate:

$$\hat{A}(s_t, a_t) = r_t - \rho + \bar{V}_{\phi_k}(s_{t+1}) - \bar{V}_{\phi_k}(s_t)$$

  Update critic by

$$\phi_{k+1} \leftarrow \phi_k - \alpha \nabla_\phi \mathcal{L}(\phi_k)$$

  where

$$\mathcal{L}(\phi_k) = \frac{1}{N}\sum_{t=1}^{N} \left\| \bar{V}_{\phi_k}(s_t) - \bar{V}_t^{\text{target}} \right\|^2$$

  Use $\hat{A}(s_t, a_t)$ to update θ_k using TRPO policy update [Schulman et al. 2015].

(iv) For ATRPO, the policy gradient term g is now the average reward policy gradient which takes the form:

$$g := \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}} \\ a\sim\pi_{\theta_k}}} \left[\nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_k} \bar{A}^{\pi_{\theta_k}}(s,a) \right]. \tag{4.10}$$

This term also requires estimating the average reward advantage function.

(v) Another subtle distinction is that in theory both g_γ and H_γ should be estimated via samples collected from the *future discounted state visitation distribution* for π_{θ_k} but this is often ignored in practice [Sutton and Barto 2018]. In fact it can be shown that this practice results in the estimated g not being an actual gradient [Nota and Thomas 2020]. In contrast for the average reward case both g and H are estimated using samples from the stationary distribution.

(vi) ATRPO also assumes that the underlying task is a continuing infinite-horizon task. But since in practice we cannot run infinitely long trajectories, all trajectories are truncated at some

large truncation value N. Unlike TRPO, during training we do not allow for episodic tasks where episodes terminate early (before N). For the MuJoCo environments, we will address this by having the agent not only resume locomotion after falling but also incur a penalty for falling (see Section 4.5 for more details.)

Also in Algorithm 2, for illustrative purposes, we use the average reward one-step bootstrapped estimate for the target of the critic and the advantage function. In practice, we instead develop and use an average-reward version of the Generalized Advantage Estimator (GAE) from Schulman et al. [2016]. We provide more details on how GAE can be generalized to the average-reward case in the next section.

4.4 Critic Estimation for the Average Reward

Suppose the agent collects a batch of data consisting of a trajectories each of length N $\{s_t, a_t, r_t, s_{t+1}\}$ $(t = 1, \ldots, N)$ using policy π. Similar to what is commonly done for critic estimation in on-policy methods, we fit some value function V_ϕ^π parameterized by ϕ using data collected with the policy.

We will first review how this is done in the discounted case. Two of the most common ways of calculating the regression target for V_ϕ^π are the *Monte Carlo* target denoted by

$$V_t^{\text{target}} = \sum_{t'=t}^{N} \gamma^{t'-t} r_t, \tag{4.11}$$

or the *bootstrapped* target

$$V_t^{\text{target}} = r_t + \gamma \bar{V}_\phi^\pi(s_{t+1}). \tag{4.12}$$

Using the dataset $\{s_t, V_t^{\text{target}}\}$, we can fit V_ϕ^π with supervised regression by minimizing the MSE between $V_\phi^\pi(s_t)$ and V_t^{target}. With the fitted value function, we can estimate the advantage

function either with the Monte Carlo estimator

$$\hat{A}^{\pi}_{\mathrm{MC}}(s_t, a_t) = \sum_{t'=t}^{N} \gamma^{t'-t} r_t - \bar{V}^{\pi}_{\phi}(s_t)$$

or the bootstrap estimator

$$\hat{A}^{\pi}_{\mathrm{BS}}(s_t, a_t) = r_t + \gamma \bar{V}^{\pi}_{\phi}(s_{t+1}) - \bar{V}^{\pi}_{\phi}(s_t).$$

When the Monte Carlo advantage estimator is used to approximate the policy gradient, it does not introduce a bias but tends to have a high variance whereas the bootstrapped estimator introduces a bias but tends to have lower variance. These two estimators are seen as the two extreme ends of the bias-variance trade-off. In order to have better control over the bias and variance, Schulman et al. [2016] used the idea of eligibility traces [Sutton and Barto 2018] and introduced the Generalized Advantage Estimator (GAE). The GAE takes the form

$$\hat{A}_{\mathrm{GAE}}(s_t, a_t) = \sum_{t'=t}^{N} (\gamma\lambda)^{t'-t} \delta_{t'} \tag{4.13}$$

where

$$\delta_{t'} = r_{t'} + \gamma \bar{V}^{\pi}_{\phi}(s_{t'+1}) - \bar{V}^{\pi}_{\phi}(s_{t'}) \tag{4.14}$$

and $\lambda \in [0, 1]$ is the eligibility trace parameter. We can then use the parameter λ to tune the bias-variance trade-off. It is worth noting two special cases corresponding to the bootstrap and Monte Carlo estimator:

$$\begin{aligned}
\lambda = 0: &\quad \hat{A}_{\mathrm{GAE}}(s_t, a_t) = r_t + \gamma \bar{V}^{\pi}_{\phi}(s_{t+1}) - \bar{V}^{\pi}_{\phi}(s_t) \\
\lambda = 1: &\quad \hat{A}_{\mathrm{GAE}}(s_t, a_t) = \sum_{t'=t}^{N} \gamma^{t'-t} r_{t'} - \bar{V}^{\pi}_{\phi}(s_t)
\end{aligned}$$

For infinite horizon tasks, the discount factor γ is used to reduce variance by downweighting rewards far into the future [Schulman et al. 2016]. Also noted in Schulman et al. [2016] is that for any $l \gg 1/(1-\gamma)$, γ^l decreases rapidly and any effects resulting from actions after $l \approx 1/(1-\gamma)$ are effectively "forgotten". This approach in essence converts a continuing control task into an episodic task where any rewards received after $l \approx 1/(1-\gamma)$ becomes negligible. This undermines the original continuing nature of the task and could prove to be especially problematic for problems where effects of actions are delayed far into the future. However, increasing γ would lead to an increase in variance. Thus in practice γ is often treated as a hyperparameter to balance the effective horizon of the task and the variance of the gradient estimator.

To mitigate this, we introduce how we can formulate critics for the average reward. A key difference is that in the discounted case we use V_ϕ^π to approximate the *discounted value function* whereas in the average reward case $\bar{V}_\phi^\pi$ is used to approximate the *average reward bias function.*

Let

$$\hat{\rho}_\pi = \frac{1}{N}\sum_{t=1}^{N} r_t$$

denote the estimated average reward. The Monte Carlo target for the average reward value function is

$$\bar{V}_t^{\text{target}} = \sum_{t'=t}^{N} (r_t - \hat{\rho}_\pi) \tag{4.15}$$

and the bootstrapped target is

$$\bar{V}_t^{\text{target}} = r_t - \hat{\rho}_\pi + \bar{V}_\phi^\pi(s_{t+1}). \tag{4.16}$$

Note that our targets (4.15-4.16) are distinctly different from the traditional discounted targets (4.11-4.12).

The Monte Carlo and Bootstrap estimators for the average reward advantage function are:

$$\hat{A}^{\pi}_{\mathrm{MC}}(s_t, a_t) = \sum_{t'=t}^{N} (r_t - \hat{\rho}_{\pi}) - \bar{V}^{\pi}_{\phi}(s_t)$$

$$\hat{A}^{\pi}_{\mathrm{BS}}(s_t, a_t) = r_{i,t} - \hat{\rho}_{\pi} + \bar{V}^{\pi}_{\phi}(s_{t+1}) - \bar{V}^{\pi}_{\phi}(s_t)$$

We can similarly extend the GAE to the average reward setting:

$$\hat{A}_{\mathrm{GAE}}(s_t, a_t) = \sum_{t'=t}^{N} \lambda^{t'-t} \delta_{t'} \tag{4.17}$$

where

$$\delta_{t'} = r_{t'} - \hat{\rho}_{\pi} + \bar{V}^{\pi}_{\phi}(s_{t'+1}) - \bar{V}^{\pi}_{\phi}(s_{t'}). \tag{4.18}$$

and set the target for the value function to

$$\bar{V}^{\mathrm{target}}_{t} = r_t - \hat{\rho}_{\pi} + \bar{V}^{\pi}_{\phi}(s_{t+1}) + \sum_{t'=t+1}^{N} \lambda^{t'-t} \delta_{t'} \tag{4.19}$$

The two special cases corresponding to $\lambda = 0$ and $\lambda = 1$ are

$$\lambda = 0: \quad \hat{A}_{\mathrm{GAE}}(s_t, a_t) = r_t - \hat{\rho}_{\pi} + \bar{V}^{\pi}_{\phi}(s_{t+1}) - \bar{V}^{\pi}_{\phi}(s_t)$$

$$\lambda = 1: \quad \hat{A}_{\mathrm{GAE}}(s_t, a_t) = \sum_{t'=t}^{N} (r_{t'} - \hat{\rho}_{\pi}) - \bar{V}^{\pi}_{\phi}(s_t)$$

We note again that the average reward advantage estimator is distinct from the discounted case. To summarize, in the average reward setting:

- The parameterized value function is used to fit the *average reward bias function.*
- The reward term r_t in the discounted formulation is replaced by $r_t - \hat{\rho}_{\pi}$.
- Without any discount factors, recent and future experiences are weighed equally thus

respecting the continuing nature of the task.

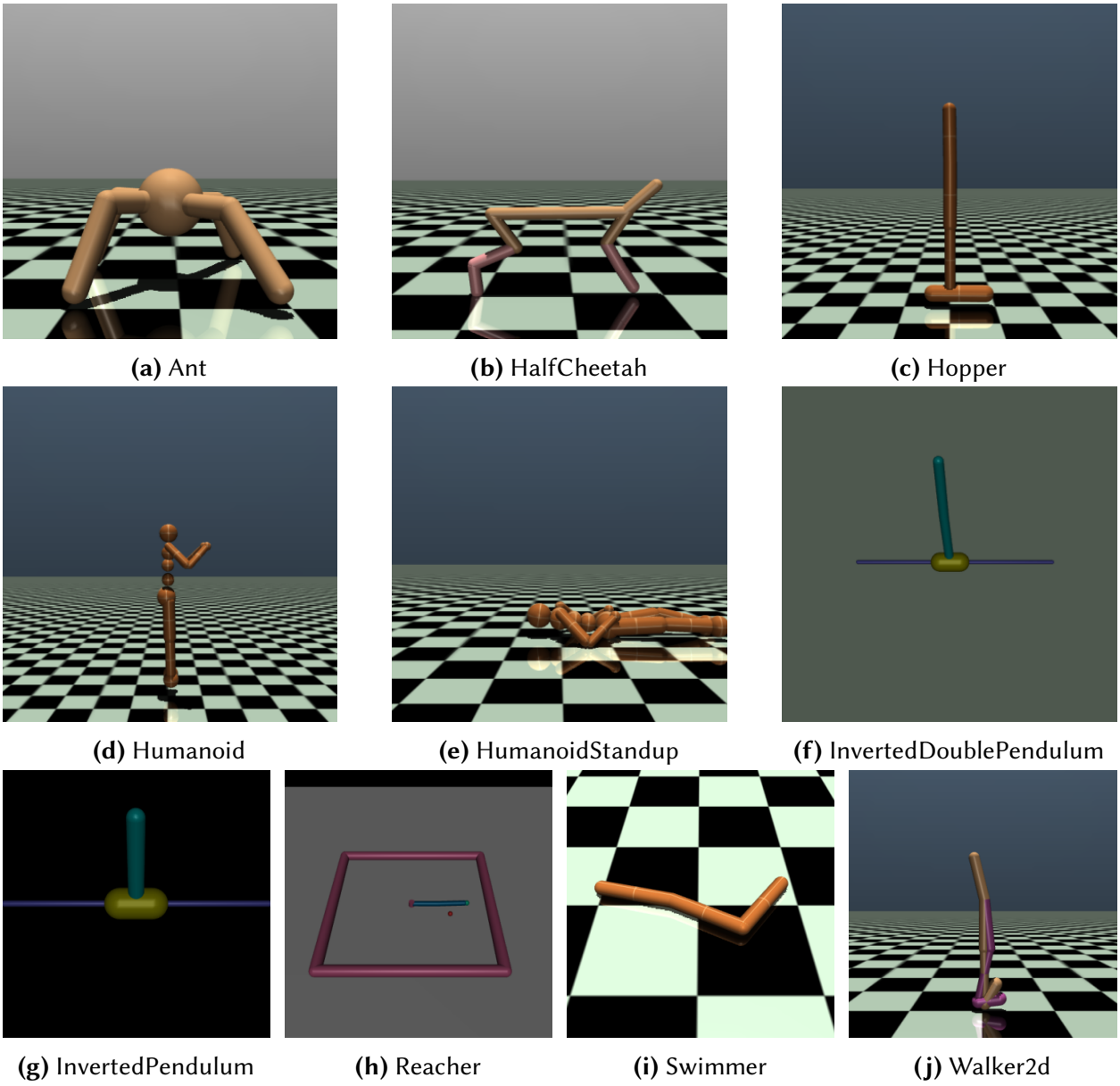
4.5 Experiments

We conducted experiments comparing the performance of ATRPO and TRPO on continuing control tasks. For most experiments in this chapter and subsequent chapters, we will be using the MuJoCo physical simulator [Todorov et al. 2012] implemented using OpenAI gym [Brockman et al. 2016].

Table 4.1: The MuJoCo environments. Description taken from https://gym.openai.com/envs/#mujoco, the last two columns are the dimensions of the state and action space respectively

Name	Description	State Space	Action Space
Ant	Make a 3D four-legged robot walk	111	8
HalfCheetah	Make a 2D cheetah robot run	17	6
Hopper	Make a 2D robot hop	11	3
Humanoid	Make a 3D two-legged robot walk	376	17
HumanoidStandup	Make a 3D two-legged robot standup	376	17
InvertedDoublePendulum	Balance a pole on a pole on a cart	11	1
InvertedPendulum	Balance a pole on a cart	4	1
Reacher	Make a 2D robot reach to a randomly located target	11	2
Swimmer	Make a 2D robot swim	8	2
Walker2d	Make a 2D robot walk	17	6

For experiments in this chapter, We consider the three tasks Ant, HalfCheetah, and Humanoid — which are considered the most challenging of the MuJoCo environments — where the natural goal is to train the agents to run as fast as possible without falling.



(a) Ant **(b)** HalfCheetah **(c)** Hopper

(d) Humanoid **(e)** HumanoidStandup **(f)** InvertedDoublePendulum

(g) InvertedPendulum **(h)** Reacher **(i)** Swimmer **(j)** Walker2d

Figure 4.1: The MuJoCo Environments

4.5.1 Evaluation Protocol

Even though the MuJoCo benchmark is commonly trained using the *discounted* objective (see e.g. Schulman et al. [2015], Wu et al. [2017], Lillicrap et al. [2016], Schulman et al. [2017b], Haarnoja et al. [2018]), it is *always* evaluated without discounting. Similarly, we also evaluate performance using the undiscounted total-reward objective for both TRPO and ATRPO.

Specifically for each environment, we train a policy for 10 million environment steps. During training, every 100,000 steps, we run 10 separate evaluation trajectories with the current policy without exploration (i.e., the policy is kept fixed and deterministic). For each evaluation trajectory we calculate the undiscounted return of the trajectory until the agent falls or until 1,000 steps, whichever comes first. We then report the average undiscounted return over the 10 trajectories. *Note that this is the standard evaluation metric for the MuJoCo environments.* In order to understand the performance of the agent for long time horizons, we also report the performance of the agent evaluated on trajectories of maximum length 10,000.

4.5.2 Comparing ATRPO and TRPO

To simulate an infinite-horizon setting during training, we do the following: when the agent falls, the trajectory does not terminate; instead the agent incurs a large reset cost for falling, and then continues the trajectory from a random start state. The reset cost is set to 100. However, we show in the supplementary material. We note that this modification does not change the underlying goal of the task. We also point out that the reset cost is only applied during training and is not used in the evaluation phase described in the previous section.

We plot the performance for ATRPO and TRPO trained with different discount factors in Figure 4.2. We see that TRPO with its best discount factor can perform as well as ATRPO for the simplest environment HalfCheetah, but ATRPO provides dramatic improvements in Ant and Humanoid. In particular for the most challenging environment Humanoid, ATRPO performs on average 50.1%

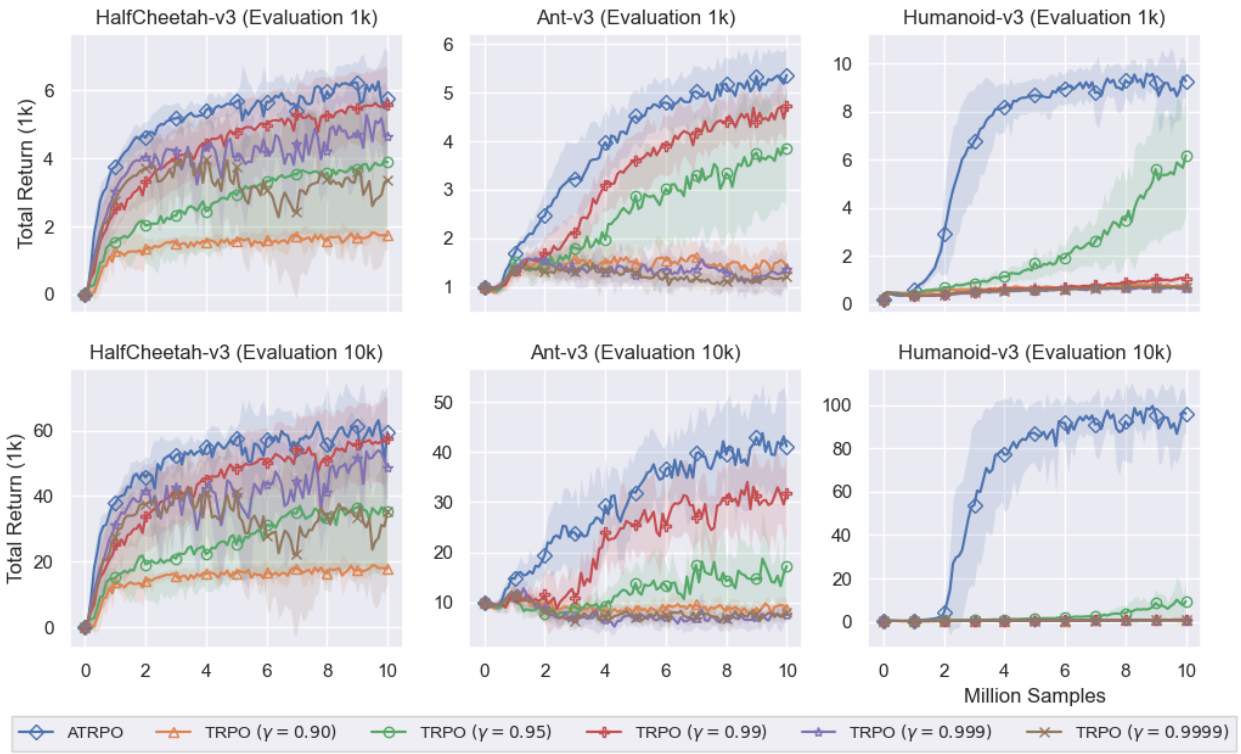


Figure 4.2: Comparing performance of ATRPO and TRPO with different discount factors. The x-axis is the number of agent-environment interactions and the y-axis is the total return averaged over 10 seeds. The solid line represents the agents' performance on evaluation trajectories of maximum length 1,000 (top row) and 10,000 (bottom row). The shaded region represents one standard deviation.

better than TRPO with its best discount factor when evaluated on trajectories of maximum length 1000. The improvement is even greater when the agents are evaluated on trajectories of maximum length 10,000 where the performance boost jumps to 913%.

Next, we show that ATRPO also significantly outperforms TRPO when TRPO is trained without the reset scheme described at the beginning of this section (i.e. the standard MuJoCo setting.). The results are shown in Figure 4.3. Note that these results are largely consistent with those reported in Figure 4.2

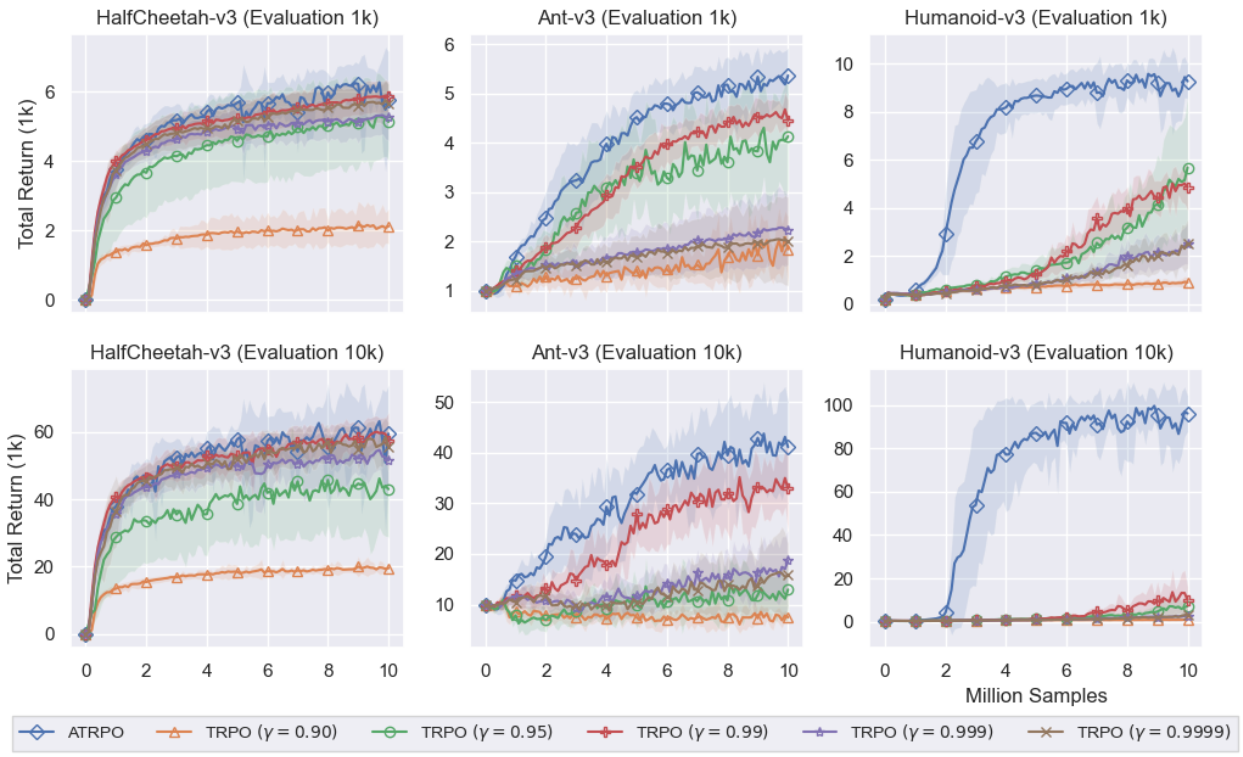


Figure 4.3: Comparing performance of ATRPO and TRPO with different discount factors. TRPO is trained without the reset scheme. The x-axis is the number of agent-environment interactions and the y-axis is the total return averaged over 10 seeds. The solid line represents the agents' performance on evaluation trajectories of maximum length 1,000 (top row) and 10,000 (bottom row). The shaded region represent one standard deviation.

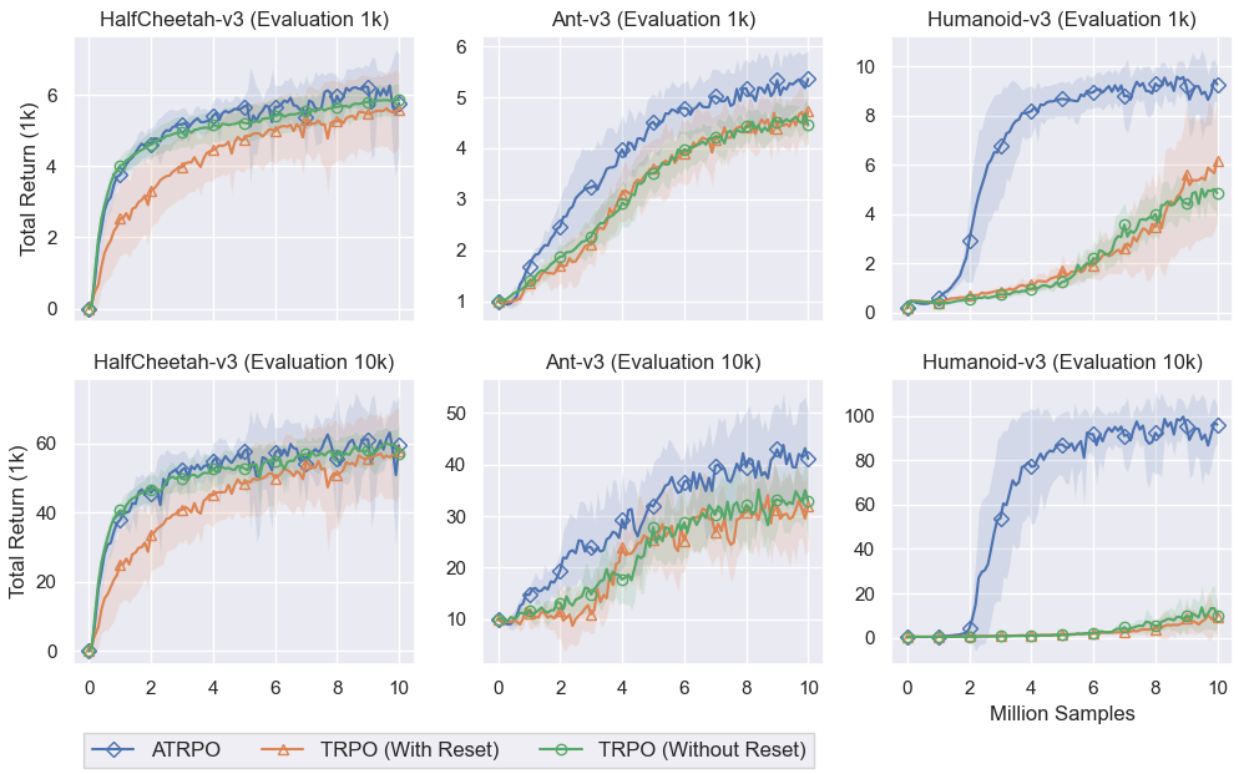


Figure 4.4: Comparing performance of ATRPO and TRPO trained with and without the reset costs. The curves for TRPO are for the best discount factor for each environment. The x-axis is the number of agent-environment interactions and the y-axis is the total return averaged over 10 seeds. The solid line represents the agents' performance on evaluation trajectories of maximum length 1,000 (top row) and 10,000 (bottom row). The shaded region represent one standard deviation.

In Figure 4.4 we plotted the performance of the best discount factor for each environment for TRPO trained with and without the reset scheme (i.e. the best performing TRPO curves from Figure 4.2 and Figure 4.3.) ATRPO is also plotted for comparison.

We make two observations regarding discounting. First, we note that increasing the discount factor does not necessarily lead to better performance for TRPO. A larger discount factor in principle enables the algorithm to seek a policy that performs well for the average-reward criterion [Blackwell 1962]. Unfortunately, a larger discount factor can also increase the variance of the gradient estimator [Zhao et al. 2011; Schulman et al. 2016], increase the complexity of the policy space [Jiang et al. 2015], lead to slower convergence [Bertsekas et al. 1995; Agarwal et al. 2020],

and degrade generalization in limited data settings [Amit et al. 2020]. Moreover, algorithms with discounting are known to become unstable as $\gamma \to 1$ [Naik et al. 2019]. Secondly, for TRPO the best discount factor is different for each environment (0.99 for HalfCheetah and Ant, 0.95 for Humanoid). The discount factor therefore serves as a hyperparameter which can be tuned to improve performance, choosing a suboptimal discount factor can have significant consequences. Both of these observation are consistent with what was seen in the literature [Andrychowicz et al. 2020]. We have shown here that using the average reward criterion directly not only delivers superior performance but also obviates the need to tune the discount factor.

4.5.3 Sensitivity Analysis on Reset Cost

For the experiments we presented so far, we introduced a reset cost in order to simulate an infinite horizon setting. Here we analyze the sensitivity of the results with respect to this reset cost.

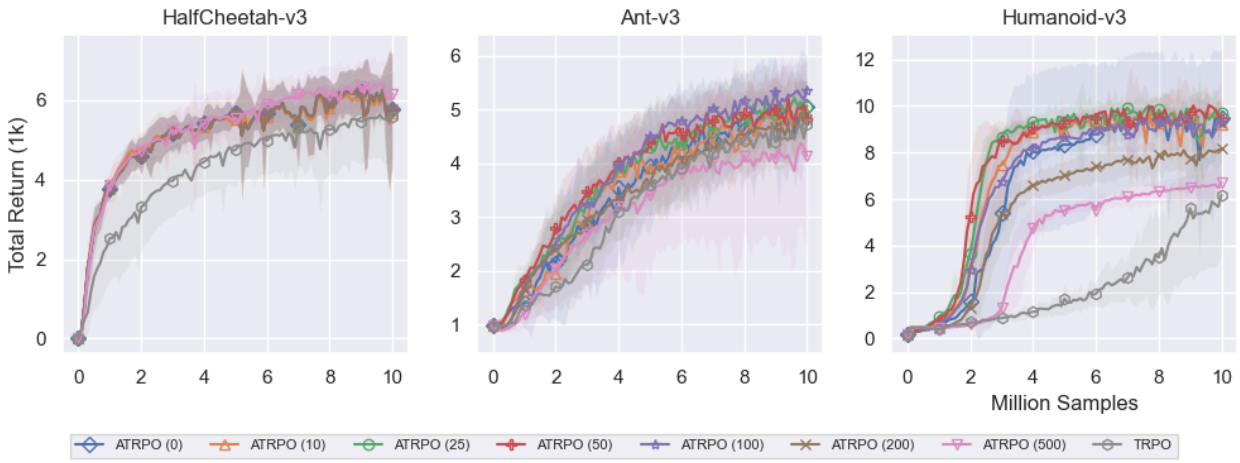


Figure 4.5: Comparing ATRPO trained with different reset costs to discounted TRPO with the best discount factor for each environment. The x-axis is the number of agent-environment interactions and the y-axis is the total return averaged over 10 seeds. The solid line represents the agents' performance on evaluation trajectories of maximum length 1,000. The shaded region represent one standard deviation.

Figure 4.5 shows that ATRPO is largely insensitive to the choice of reset cost. Though we note that for Humanoid, extremely large reset costs (200 and 500) does negatively impact performance but the result is still above that of TRPO.

4.5.4 Understanding Long Run Performance

Next, we demonstrate that agents trained using the average reward criterion are better at optimizing for long-term returns. Here, we first train Humanoid with 10 million samples with ATRPO and with TRPO with a discount factor of 0.95 (shown to be the best discount factor in the previous experiments).

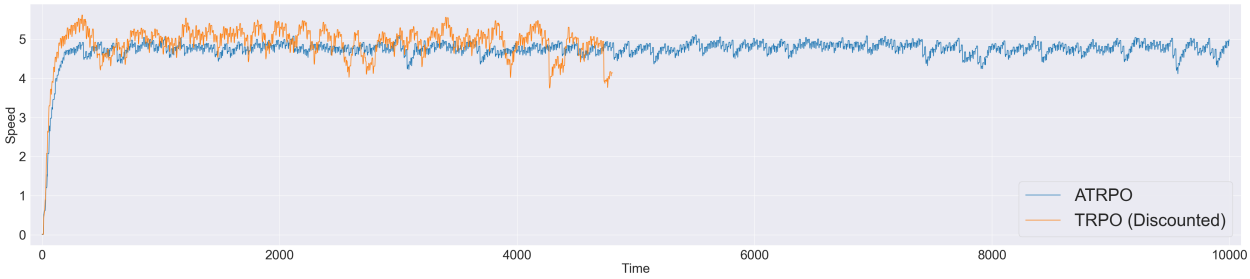


Figure 4.6: Speed-time plot of a single trajectory (maximum length 10,000) for ATRPO and Discounted TRPO in the Humanoid-v3 environment. The solid line represents the speed of the agent at the corresponding timesteps.

Then for evaluation, we run the trained ATRPO and TRPO policies for a trajectory of 10,000 timesteps (or until the agent falls). We use the same random seeds for the two algorithms. Figure 4.6 is a plot of the speed of the agent at each time step of the trajectory, using the *seed that gives the best performance for discounted TRPO*. We see in Figure 4.6 that the discounted algorithm gives a higher initial speed at the beginning of the trajectory. However its overall speed is much more erratic throughout the trajectory, resulting in the agent falling over after approximately 5000 steps. This coincides with the notion of discounting where more emphasis is placed at the beginning of the trajectory and ignores longer-term behavior. On the other hand, the average-reward policy — while having a slightly lower velocity overall throughout its trajectory — is able to sustain the trajectory much longer, thus giving it a higher total return. In fact, we observed that for all 10 random seeds we tested, the average reward agent is able to finish the entire 10,000 time step trajectory without falling. In Table 4.2 we present the summary statistics of trajectory length for all trajectories using discounted TRPO we note that the median trajectory length for the TRPO

discounted agent is 452.5, meaning that on average TRPO performs significantly worse than what is reported in Figure. 4.6.

Table 4.2: Summary statistics for all 10 trajectories using a Humanoid-v3 agent trained with TRPO

Min	Max	Average	Median	Std
108	4806	883.1	452.5	1329.902

4.5.5 Implementation Details

All experiments in this secton were implemented in Pytorch 1.3.1[Paszke et al. 2019] and Python 3.7.4 on Intel Xeon Gold 6230 processors. We based our TRPO implementation on https://github.com/ikostrikov/pytorch-trpo and https://github.com/Khrylx/PyTorch-RL. Our CPO implementation is our own Pytorch implementation based on https://github.com/jachiam/cpo and https://github.com/openai/safety-starter-agents. Our hyperparameter selections were also based on these implementations. Our choice of hyperparameters were based on the motivation that we wanted to put discounted TRPO in the best possible light and compare its performance with ATRPO. Our hyperparameter choices for ATRPO mirrored the discounted case since we wanted to understand how performance for the average reward case differs while controlling for all other variables.

With the exception of the sensitivity analysis experiments, the reset cost is set to 100 on all three environments. In the original implementation of the MuJoCo environments in OpenAI gym, the maximum episode length is set to 1000[2], we removed this restriction in our experiments in order to study long-run performance.

We used a two-layer feedforward neural network with a tanh activation for both our policy and critic networks. The policy is Gaussian with a diagonal covariance matrix. The policy networks outputs a mean vector and a vector containing the state-independent log standard deviations.

[2]See https://github.com/openai/gym/blob/master/gym/envs/__init__.py

States are normalized by the running mean and the running standard deviation before being fed to any network. We used the GAE for advantage estimation (see Section 4.4). The advantage values are normalized by its batch mean and batch standard deviation before being used for policy updates. Learning rates are linearly annealed to 0 over the course of training. Note that these settings are common in most open-source implementations of TRPO and other on-policy algorithms. For training and evaluation, we used different random seeds (i.e. the random seeds we used to generate the evaluation trajectories are different from those used during training). We optimize both networks using Adam [Kingma and Ba 2015]. Table 6.6 summarizes the hyperparameters used in our experiments.

Table 4.3: Hyperparameter Setup for Experiments in Chapter 4

Hyperparameter	TRPO/ATRPO
No. of hidden layers	2
No. of hidden nodes	64
Activation	tanh
Initial log std	-0.5
Batch size	5000
GAE parameter (reward)	0.95
Learning rate for policy	3×10^{-4}
Learning rate for critic net	3×10^{-4}
$L2$-regularization coeff. for critic net	3×10^{-3}
Damping coeff.	0.01
Backtracking coeff.	0.8
Max backtracking iterations	10
Max conjugate gradient iterations	10
Trust region bound δ	0.01

4.6 Conclusion

Motivate by the theoretical results in the previous chapter, we derived practical algorithms. We propose ATRPO, a modification of the TRPO algorithm for on-policy DRL. We demonstrate through

a series of experiments that ATRPO is highly effective on high-dimensional continuing control tasks. To the best of our knowledge, this is the first DRL algorithm for the average reward setting.

5 | Supervised Policy Update

5.1 Introduction

In this chapter, we introduce an alternative paradigm for solving the local policy search problem. As we have extensively discussed in Chapter 3, the guideline of limiting the search to nearby policies seems reasonable in principle, but requires a distance $D(\pi_\theta, \pi_{\theta_k})$ between the current policy π_{θ_k} and the candidate new policy π_θ, and then attempt to solve the constrained optimization problem:

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{J}(\pi_\theta \mid \pi_{\theta_k}, \text{ new data}) \\ \text{subject to} \quad & D(\pi_\theta, \pi_{\theta_k}) \leqslant \delta \end{aligned} \tag{5.1}$$

where $\hat{J}(\pi_\theta \mid \pi_{\theta_k}, \text{ new data})$ is an estimate of $J(\pi_\theta)$, the performance of policy π_θ, based on the previous policy π_{θ_k} and the batch of fresh data generated by π_{θ_k}. The objective in (5.1) attempts to maximize the performance of the updated policy, and the constraint in (5.1) ensures that the updated policy is not too far from the policy π_{θ_k} that was used to generate the data that can be possibly annealed over time.

To summarize, the general local policy update problem gives rise to three questions:

- What is a good estimator $\hat{J}(\pi_\theta \mid \pi_{\theta_k}, \text{ new data})$ for the performance of π_θ?
- Given that there are many possible ways to define closeness $D(\pi_\theta, \pi_{\theta_k})$, which definitions lead to the most sample-efficient algorithms?

- Given an estimator $\hat{J}(\pi_\theta|\pi_{\theta_k}, \text{ new data})$ and a constraint $D(\pi_\theta, \pi_{\theta_k}) \leqslant \delta$, what is a good algorithm for solving the optimization problem in (5.1)?

It is our belief that these three sub-problems should be solved in a unified manner.

We propose a new methodology, called Supervised Policy Update (SPU), for this sample efficiency problem. The methodology is general in that it applies to both discrete and continuous action spaces, and can address a wide variety of constraint types for the constrained optimization problem in (5.1). Starting with data generated by the current policy, SPU optimizes over a proximal policy space to find an optimal *non-parameterized policy*. It then solves a supervised regression problem to convert the non-parameterized policy to a parameterized policy, from which it draws new samples. We develop a general methodology for finding an optimal policy in the non-parameterized policy space, and then illustrate the methodology for three different definitions of proximity. We also show how the Natural Policy Gradient and Trust Region Policy Optimization (NPG/TRPO) problems and the Proximal Policy Optimization (PPO) problem can be addressed by this methodology. While SPU is substantially simpler than NPG/TRPO in terms of mathematics and implementation, our extensive experiments show that SPU is more sample efficient than TRPO in Mujoco simulated robotic tasks and PPO in Atari video game tasks.

Our work also strikes the right balance between performance and simplicity. The implementation is only slightly more involved than PPO [Schulman et al. 2017b]. Simplicity in RL algorithms has its own merits. This is especially useful when RL algorithms are used to solve problems outside of traditional RL testbeds.

5.2 The SPU Framework

The SPU methodology has two steps. In the first step, for a given constraint criterion $D(\pi, \pi_{\theta_k}) \leqslant \delta$, we find the optimal solution to the non-parameterized problem:

$$\begin{aligned} \underset{\pi \in \Pi}{\text{maximize}} \quad & \hat{J}_{\pi_{\theta_k}}(\pi) \\ \text{subject to} \quad & D(\pi, \pi_{\theta_k}) \leqslant \delta \end{aligned} \tag{5.2}$$

Note that π is not restricted to the set of parameterized policies Π_θ. As commonly done, we approximate the objective function. However, unlike PPO/TRPO, we are not approximating the constraint. We will show below the optimal solution π^* for the non-parameterized problem (5.2) can be determined nearly in closed form for many natural constraint criteria $D(\pi, \pi_{\theta_k}) \leqslant \delta$.

In the second step, we attempt to find a policy π_θ in the parameterized space Π_θ that is close to the target policy π^*. Concretely, to advance from θ_k to θ_{k+1}, we perform the following steps:

(i) We first sample N trajectories using policy π_{θ_k}, giving sample data (s_i, a_i, A_i), $i = 1, .., m$. Here A_i is an estimate of the advantage value $A^{\pi_{\theta_k}}(s_i, a_i)$. (For simplicity, we index the samples with i rather than with (i, t) corresponding to the tth sample in the ith trajectory.)

(ii) For each s_i, we define the target distribution π^* to be the optimal solution to the constrained optimization problem (5.2) for a specific constraint D.

(iii) We then fit the policy network π_θ to the target distributions $\pi^*(\cdot|s_i)$, $i = 1, .., m$. Specifically, to find θ_{k+1}, we minimize the following supervised loss function:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{m} D_{\text{KL}}\left(\pi_\theta \| \pi^*\right)[s_i]. \tag{5.3}$$

For this step, we initialize with the weights for π_{θ_k}. We minimize the loss function $L(\theta)$ with stochastic gradient descent methods. Overfitting to the fresh data (s_i, a_i, A_i), $i = 1, .., m$ can

be controlled by dynamic early stopping [Goodfellow et al. 2016]. The resulting θ becomes our θ_{k+1}.

5.3 SPU Applied to Specific Criteria

To illustrate the SPU methodology, for three different but natural types of proximity constraints, we solve the corresponding non-parameterized optimization problem and derive the resulting gradient for the SPU supervised learning problem. We also demonstrate that different constraints lead to very different but intuitive forms of the gradient update. Although we consider three classes of proximity constraint, there may be yet another class that leads to even better performance. The methodology allows researchers to explore other proximity constraints in the future.

5.3.1 Forward KL Constraints

We first consider constraint criteria of the form:

$$\begin{aligned} \underset{\pi\in\Pi}{\text{maximize}} \quad & \sum_s d_{\pi_{\theta_k},\gamma}(s) \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)} \left[A_\gamma^{\pi_{\theta_k}}(s,a)\right] \\ \text{subject to} \quad & \sum_s d_{\pi_{\theta_k},\gamma}(s) D_{\mathrm{KL}}\left(\pi\|\pi_{\theta_k}\right)[s] \leqslant \delta \\ & D_{\mathrm{KL}}\left(\pi\|\pi_{\theta_k}\right)[s] \leqslant \epsilon \text{ for all } s \end{aligned} \tag{5.4}$$

We refer to the first constraint in (5.4) as the *aggregated KL constraint* and the second constraint as the *disaggregated KL constraint*. These two constraints taken together restrict π from deviating too much from π_{θ_k}. We shall refer to (5.4) as the forward-KL non-parameterized optimization problem.

Note that this problem without the disaggregated constraints is analogous to the TRPO problem [Schulman et al. 2015]. The TRPO paper actually prefers enforcing the disaggregated constraint to enforcing the aggregated constraints. However, for mathematical conveniences, they worked

with the aggregated constraints: "While it is motivated by the theory, this problem is impractical to solve due to the large number of constraints. Instead, we can use a heuristic approximation which considers the average KL divergence" [Schulman et al. 2015]. The SPU framework allows us to solve the optimization problem with the disaggregated constraints exactly. Experimentally, we compared against TRPO in a controlled experimental setting, e.g. using the same advantage estimation scheme, etc. Since we clearly outperform TRPO, we argue that SPU's two-process procedure has significant potentials.

For each $\lambda > 0$, define:

$$\pi^{\lambda}(a|s) = \frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_{\lambda}(s)} \exp\left(\frac{A_{\gamma}^{\pi_{\theta_k}}(s,a)}{\lambda}\right) \tag{5.5}$$

where $\mathcal{Z}_{\lambda}(s)$ is the normalization term. Note that $\pi^{\lambda}(a|s)$ is a function of λ. Further, for each s, let λ_s be such that $D_{\mathrm{KL}}\left(\pi^{\lambda_s} \| \pi_{\theta_k}\right)[s] = \epsilon$. Also let $\Gamma^{\lambda} = \{s : D_{\mathrm{KL}}\left(\pi^{\lambda} \| \pi_{\theta_k}\right)[s] \leqslant \epsilon\}$.

Theorem 5.1. *The optimal solution to the problem* (5.4) *is given by:*

$$\widetilde{\pi}^{\lambda}(a|s) = \begin{cases} \pi^{\lambda}(a|s) & s \in \Gamma^{\lambda} \\ \pi^{\lambda_s}(a|s) & s \notin \Gamma^{\lambda} \end{cases} \tag{5.6}$$

where λ is chosen so that $\sum_s d^{\pi_{\theta_k}}(s) D_{KL}\left(\widetilde{\pi}^{\lambda} \| \pi_{\theta_k}\right)[s] = \delta$.

Proof. We first show that (5.4) is a convex optimization. To this end, first note that the objective is a linear function of the decision variables $\pi = \{\pi(a|s) : \quad s \in \mathcal{S},\ a \in \mathcal{A}\}$. The LHS of the disaggregated KL constraint can be rewritten as: $\sum_{a \in \mathcal{A}} \pi(a|s) \log \pi(a|s) - \sum_{a \in \mathcal{A}} \pi \log \pi_{\theta_k}(a|s)$. The second term is a linear function of π. The first term is a convex function since the second derivative of each summand is always positive. The LHS of the disaggregated KL constraint is thus a convex function. By extension, the LHS of the aggregated KL constraint is also a convex function since it is a nonnegative weighted sum of convex functions. The problem (5.4) is thus a

convex optimization problem. According to Slater's constraint qualification, strong duality holds since is a feasible solution to (5.4) where the inequality holds strictly.

We can therefore solve (5.4) by solving the related Lagrangian problem. For a fixed λ consider:

$$\begin{aligned} \underset{\pi \in \Pi}{\text{maximize}} \quad & \sum_s d_{\pi_{\theta_k}, \gamma}(s) \left[\mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [A_\gamma^{\pi_{\theta_k}}(s,a)] - \lambda D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \right] \\ \text{subject to} \quad & D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \leqslant \epsilon \text{ for all } s \end{aligned} \tag{5.7}$$

The above problem decomposes into separate problems, one for each state s:

$$\begin{aligned} \underset{\pi(\cdot|s)}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} [A_\gamma^{\pi_{\theta_k}}(s,a)] - \lambda D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \\ \text{subject to} \quad & D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \leqslant \epsilon \end{aligned} \tag{5.8}$$

Further consider the objective in (5.8) without the constraint:

$$\begin{aligned} \underset{\pi(\cdot|s)}{\text{maximize}} \quad & \sum_a \pi(a|s) \left[A_\gamma^{\pi_{\theta_k}}(s,a) - \lambda \log\left(\frac{\pi(a|s)}{\pi_{\theta_k}(a|s)}\right) \right] \\ \text{subject to} \quad & \sum_a \pi(a|s) = 1 \\ & \pi(a|s) \geqslant 0, \qquad \text{for all } a \end{aligned} \tag{5.9}$$

A simple Lagrange-multiplier argument shows that the optimal solution to (5.9) is given by:

$$\pi^\lambda(a|s) = \frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_\lambda(s)} \exp\left[\frac{A_\gamma^{\pi_{\theta_k}}(s,a)}{\lambda}\right] \tag{5.10}$$

where $\mathcal{Z}_\lambda(s)$ is defined so that π^λ is a valid distribution. Now returning to the decomposed constrained problem (5.8), there are two cases to consider. The first case is when $D_{\mathrm{KL}}\left(\pi^\lambda \| \pi_{\theta_k}\right)[s] \leqslant \epsilon$. In this case, the optimal solution to (5.8) is $\pi^\lambda(a|s)$. The second case is when $D_{\mathrm{KL}}\left(\pi^\lambda \| \pi_{\theta_k}\right)[s] > \epsilon$. In this case the optimal is $\pi^\lambda(a|s)$ with λ replaced with λ_s, where λ_s is the solution to

$D_{\mathrm{KL}}\left(\pi^{\lambda} \| \pi_{\theta_k}\right)[s] = \epsilon$. Thus, an optimal solution to (5.8) is given by:

$$\widetilde{\pi}^{\lambda}(a|s) = \begin{cases} \dfrac{\pi_{\theta_k}(a|s)}{\mathcal{Z}(s)} \exp\left[\dfrac{A_{\gamma}^{\pi_{\theta_k}}(s,a)}{\lambda}\right] & s \in \Gamma^{\lambda} \\ \dfrac{\pi_{\theta_k}(a|s)}{\mathcal{Z}(s)} \exp\left[\dfrac{A_{\gamma}^{\pi_{\theta_k}}(s,a)}{\lambda^{s}}\right] & s \notin \Gamma^{\lambda} \end{cases} \tag{5.11}$$

where $\Gamma^{\lambda} = \{s : D_{\mathrm{KL}}\left(\pi^{\lambda} \| \pi_{\theta_k}\right)[s] \leqslant \epsilon\}$.

To find the Lagrange multiplier λ, we can then do a line search to find the λ that satisfies:

$$\sum_{s} d^{\pi_{\theta_k}}(s) D_{\mathrm{KL}}\left(\widetilde{\pi}^{\lambda} \middle\| \pi_{\theta_k}\right)[s] = \delta \tag{5.12}$$

□

Equation (5.6) provides the structure of the optimal non-parameterized policy. As part of the SPU framework, we then seek a parameterized policy π_{θ} that is close to $\widetilde{\pi}^{\lambda}(a|s)$, that is, minimizes the loss function (5.3). To this end, we apply stochastic gradient descent, we can show that the gradient of the loss function takes the following form:

Proposition 5.2. *For each sampled state s_i:*

$$\nabla_{\theta} D_{KL}\left(\pi_{\theta} \middle\| \widetilde{\pi}^{\lambda}\right)[s_i] = \nabla_{\theta} D_{KL}\left(\pi_{\theta} \| \pi_{\theta_k}\right)[s_i] - \frac{1}{\widetilde{\lambda}_{s_i}} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(\cdot|s_i)}\left[\nabla_{\theta} \log \pi_{\theta}(a|s_i) A_{\gamma}^{\pi_{\theta_k}}(s_i, a)\right] \tag{5.13}$$

where $\widetilde{\lambda}_{s_i} = \lambda$ for $s_i \in \Gamma^{\lambda}$ and $\widetilde{\lambda}_{s_i} = \lambda_{s_i}$ for $s_i \notin \Gamma^{\lambda}$.

Proof. We first calculate the cross entropy $H(\pi_\theta, \tilde{\pi}^{\tilde{\lambda}_s})$ which gives us:

$$\begin{aligned}
H(\pi_\theta, \tilde{\pi}^{\tilde{\lambda}_s}) &= -\sum_a \pi_\theta(a|s) \log \widetilde{\pi}^{\widetilde{\lambda}_s}(a|s) \\
&= -\sum_a \pi_\theta(a|s) \log \left(\frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_{\widetilde{\lambda}_s}(s)} \exp\left(\frac{A_\gamma^{\pi_{\theta_k}}(s,a)}{\widetilde{\lambda}_s} \right) \right) \\
&= -\sum_a \pi_\theta(a|s) \log \left(\frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_{\widetilde{\lambda}_s}(s)} \right) - \sum_a \pi_\theta(a|s) \frac{A_\gamma^{\pi_{\theta_k}}(s,a)}{\widetilde{\lambda}_s} \\
&= -\sum_a \pi_\theta(a|s) \log \pi_{\theta_k}(a|s) + \sum_a \pi_\theta(a|s) \log \mathcal{Z}_{\widetilde{\lambda}_s}(s) - \frac{1}{\widetilde{\lambda}_s} \sum_a \pi_{\theta_k}(a|s) \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a) \\
&= H(\pi_\theta, \pi_{\theta_k})[s] + \log \mathcal{Z}_{\widetilde{\lambda}_s}(s) - \frac{1}{\widetilde{\lambda}_s} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(.|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a) \right]
\end{aligned} \tag{5.14}$$

Taking the gradient of the above cross entropy term:

$$\nabla_\theta H(\pi_\theta, \tilde{\pi}^{\tilde{\lambda}_s}) = \nabla_\theta H(\pi_\theta, \pi_{\theta_k})[s] - \frac{1}{\widetilde{\lambda}_s} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(.|s)} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a) \right] \tag{5.15}$$

Finally we add the gradient of the entropy on both sides and collapse the sum of gradients of cross entropy and entropy into the gradient of the KL:

$$\nabla_\theta D_{\mathrm{KL}} \left(\pi_\theta \middle\| \widetilde{\pi}^{\widetilde{\lambda}_s} \right)[s] = \nabla_\theta D_{\mathrm{KL}} \left(\pi_\theta \| \pi_{\theta_k} \right)[s] - \frac{1}{\widetilde{\lambda}_s} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(.|s)} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a) \right] \tag{5.16}$$

□

We estimate the expectation in (5.13) with the sampled action a_i and approximate $A_\gamma^{\pi_{\theta_k}}(s_i, a_i)$ as A_i (obtained from the critic network), giving:

$$\nabla_\theta D_{\mathrm{KL}} \left(\pi_\theta \middle\| \widetilde{\pi}^{\lambda} \right)[s_i] \approx \nabla_\theta D_{\mathrm{KL}} \left(\pi_\theta \| \pi_{\theta_k} \right)[s_i] - \frac{1}{\widetilde{\lambda}_{s_i}} \frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i \tag{5.17}$$

To simplify the algorithm, we slightly modify (5.17). We replace the hyperparameter δ with the hyper-parameter λ and tune λ rather than δ. Further, we set $\widetilde{\lambda}_{s_i} = \lambda$ for all s_i in (5.17) and introduce

per-state acceptance to enforce the disaggregated constraints, giving the approximate gradient:

$$\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \middle\| \widetilde{\pi}^{\lambda}\right) \approx \frac{1}{m}\sum_{i=1}^{m}\left[\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i] - \frac{1}{\lambda}\frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i\right] \mathbf{1}_{D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i] \leqslant \epsilon} \tag{5.18}$$

We make the approximation that the disaggregated constraints are only enforced on the states in the sampled trajectories. We use (5.18) as our gradient for supervised training of the policy network. The equation (5.18) has an intuitive interpretation: the gradient represents a trade-off between the approximate performance of π_θ (as captured by $\frac{1}{\lambda}\frac{\nabla_\theta \pi_\theta(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} A_i$) and how far π_θ diverges from π_{θ_k} (as captured by $\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i]$). For the stopping criterion, we train until $\frac{1}{m}\sum_i D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i] \approx \delta$.

We present an outline of the Forward KL SPU algorithm in Algorithm 3. We note also that similar to what we have done with TRPO in Chapter 4, SPU can be similarly modified to directly optimize for the average reward.

Algorithm 3 SPU (Forward KL)

Initialize: Policy network π_{θ_0}, Value networks V_{ϕ_0}.

while Stopping criteria not met **do**
  Generate trajectories $\tau \sim \pi_{\theta_k}$.
  for K epochs **do**
    for each minibatch **do**
      Update value networks by minimizing MSE of V_{ϕ_k}, $V_{\phi_k}^{\text{target}}$.
      Update policy network using Equation (5.18)
    if $\frac{1}{N}\sum_{j=1}^{N} D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_j] > \delta$ **then**
      Break out of inner loop

5.3.2 Backward KL Constraints

In a similar manner, we can derive the structure of the optimal policy when using the reverse KL-divergence as the constraint. For simplicity, we provide the result for when there are only

disaggregated constraints. We seek to find the non-parameterized optimal policy by solving:

$$\begin{aligned} \underset{\pi \in \Pi}{\text{maximize}} \quad & \sum_s d^{\pi_{\theta_k}}(s) \mathop{\mathbb{E}}_{a \sim \pi(\cdot|s)} \left[A_\gamma^{\pi_{\theta_k}}(s,a) \right] \\ & D_{\text{KL}}\left(\pi_{\theta_k} \| \pi\right)[s] \leqslant \epsilon \text{ for all } s \end{aligned} \tag{5.19}$$

Theorem 5.3. *The optimal solution to the problem* (5.19) *is given by:*

$$\pi^*(a|s) = \pi_{\theta_k}(a|s) \frac{\lambda(s)}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)} \tag{5.20}$$

where $\lambda(s) > 0$ *and* $\lambda'(s) > \max_a A_\gamma^{\pi_{\theta_k}}(s,a)$

Proof. The problem (5.19) decomposes into separate problems, one for each state $s \in \mathcal{S}$:

$$\begin{aligned} \underset{\pi(\cdot|s)}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a) \right] \\ \text{subject to} \quad & \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\log \frac{\pi_{\theta_k}(a|s)}{\pi(a|s)} \right] \leqslant \epsilon \end{aligned} \tag{5.21}$$

After some algebra, we see that above optimization problem is equivalent to:

$$\begin{aligned} \underset{\pi(\cdot|s)}{\text{maximize}} \quad & \sum_a A_\gamma^{\pi_{\theta_k}}(s,a) \pi(a|s) \\ \text{subject to} \quad & -\sum_a \pi_{\theta_k}(a|s) \log \pi(a|s) \leqslant \epsilon + H(\pi_{\theta_k}) \\ & \sum_a \pi(a|s) = 1 \\ & \pi(a|s) \geqslant 0, \text{ for all } a \end{aligned} \tag{5.22}$$

where $H(\pi_{\theta_k})$ denotes the entropy of π_{θ_k}. Problem (5.22) is a convex optimization problem with Slater's condition holding. Strong duality thus holds for the problem (5.22). Applying standard

Lagrange multiplier arguments, it is easily seen that the solution to (5.22) is

$$\pi^*(a|s) = \pi_{\theta_k}(a|s)\frac{\lambda(s)}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}$$

where $\lambda(s)$ and $\lambda'(s)$ are constants chosen such that the disaggregegated KL constraint is binding and the sum of the probabilities equals 1. It is easily seen $\lambda(s) > 0$ and $\lambda'(s) > \max_a A_\gamma^{\pi_{\theta_k}}(s,a)$. □

Note that the structure of the optimal policy with the backward KL constraint is quite different from that with the forward KL constraint. Similar to the forward KL case, the gradient of the loss function can also be evaluated with some straightforward calculation:

Proposition 5.4. *The gradient of the loss function w.r.t. the backward KL optimal policy is:*

$$\nabla_\theta D_{KL}\left(\pi_\theta \| \pi^*\right)[s] = \nabla_\theta D_{KL}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] - \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log\left(\frac{1}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}\right)\right] \tag{5.23}$$

Proof. Similar to the forward KL case, we first calculate the cross entropy term:

$$\begin{aligned}
H(\pi_\theta, \pi^*)[s] &= -\sum_a \pi_\theta(a|s) \log \pi^*(a|s) \\
&= -\sum_a \pi_\theta(a|s) \log\left(\pi_{\theta_k}(a|s)\frac{\lambda(s)}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}\right) \\
&= -\sum_a \pi_\theta(a|s) \log \pi_{\theta_k}(a|s) - \sum_a \pi_\theta(a|s) \log \lambda(s) + \sum_a \pi_\theta(a|s) \log(\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)) \\
&= H(\pi_\theta, \pi_{\theta_k})[s] - \lambda(s) + \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log(\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a))\right] \\
&= H(\pi_\theta, \pi_{\theta_k})[s] - \lambda(s) - \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log\frac{1}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}\right]
\end{aligned} \tag{5.24}$$

Taking the gradient of both sides gives us:

$$\nabla_\theta H(\pi_\theta, \pi^*)[s] = \nabla_\theta H(\pi_\theta, \pi_{\theta_k})[s] - \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log\frac{1}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}\right] \tag{5.25}$$

Finally we add the gradient of the entropy on both sides and collapse the sum of gradients of cross entropy and entropy into the gradient of the KL:

$$\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s] = \nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] - \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}}\left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \log \frac{1}{\lambda'(s) - A_\gamma^{\pi_{\theta_k}}(s,a)}\right] \tag{5.26}$$

□

The equation (5.23) has an intuitive interpretation. It increases the probability of action a if $A_\gamma^{\pi_{\theta_k}}(s,a) > \lambda'(s) - 1$ and decreases the probability of action a if $A_\gamma^{\pi_{\theta_k}}(s,a) < \lambda'(s) - 1$. (5.23) also tries to keep π_θ close to π_{θ_k} by minimizing their KL divergence.

5.3.3 L^∞ Constraints

In this section we show how a PPO-like objective [Schulman et al. 2017b] can be formulated in the context of SPU. PPO-clipped updates policy by

$$\theta_{k+1} = \operatorname*{argmax}_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_{\theta_k}}\left[L_{clip}(\pi_\theta, \pi_{\theta_k})\right] \tag{5.27}$$

where

$$L_{clip}(\pi_\theta, \pi_{\theta_k}) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_\gamma^{\pi_{\theta_k}}(s,a),\ \mathrm{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A_\gamma^{\pi_{\theta_k}}(s,a)\right) \tag{5.28}$$

We note that this objective can be estimated using sample trajectory collected from the environment and can be easily optimized via stochastic gradient ascent. The clipping in PPO can be seen as an attempt at keeping $\pi_\theta(a_i|s_i)$ from becoming neither much larger than $(1+\epsilon)\pi_{\theta_k}(a_i|s_i)$ nor much smaller than $(1-\epsilon)\pi_{\theta_k}(a_i|s_i)$ for $i = 1, \ldots, m$. This is illustrated in Figure 5.1. When the advantage is positive, the objective function increases if the current state action pair (s_i, a_i) is more likely. However the objective has an upper limit in how much it is allowed to increase, i.e.

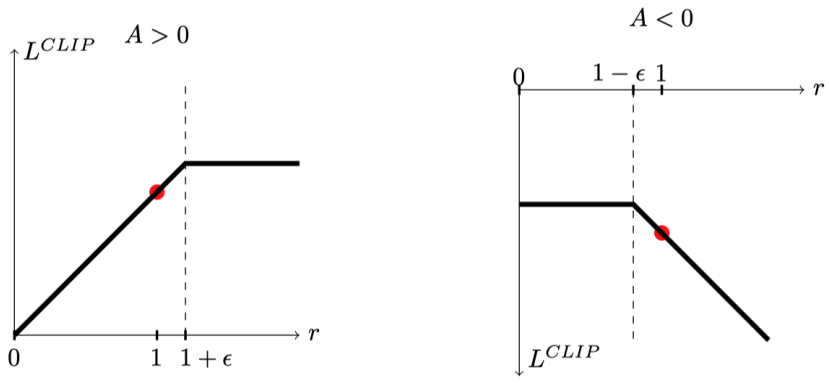


Figure 5.1: Plot of L_{clip} against the importance sampling ratio $r = \dfrac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ for when the advantage is positive and negative (Image from [Schulman et al. 2017b])

the ceiling $1+\epsilon$. By similar logic, when the advantage is negative, the objective increase when (s_i, a_i) is less likely and this value is capped by $1-\epsilon$.

In this subsection, we consider the constraint function

$$D(\pi, \pi_{\theta_k}) = \max_{i=1,\ldots,m} \frac{|\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)|}{\pi_{\theta_k}(a_i|s_i)} \tag{5.29}$$

which leads us to the following optimization problem:

$$\begin{aligned} \underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad & \sum_{i=1}^{m} A_\gamma^{\pi_{\theta_k}}(s_i, a_i) \frac{\pi(a_i|s_i)}{\pi_k(a_i|s_i)} \\ \text{subject to} \quad & \left| \frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \right| \leqslant \epsilon \quad i = 1, \ldots, m \\ & \sum_{i=1}^{m} \left(\frac{\pi(a_i|s_i) - \pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \right)^2 \leqslant \delta \end{aligned} \tag{5.30}$$

Note that here we are using a variation of the SPU methodology described in Section 5.2 since here we first create estimates of the expectations in the objective and constraints and then solve the optimization problem (rather than first solve the optimization problem and then take samples as done for Theorems 5.1 and 5.3). Note that we have also included an aggregated constraint

(constraint 2 of (5.31)) in addition to the PPO-like constraint (constraint 1 of (5.31)), which further ensures that the updated policy is close to π_{θ_k}.

Theorem 5.5. *The optimal solution to the optimization problem* (5.31) *is given by:*

$$\pi^*(a_i|s_i) = \begin{cases} \pi_{\theta_k}(a_i|s_i)\min\{1+\lambda A_i, 1+\epsilon\} & A_i \geqslant 0 \\ \pi_{\theta_k}(a_i|s_i)\max\{1+\lambda A_i, 1-\epsilon\} & A_i < 0 \end{cases} \tag{5.31}$$

for some $\lambda > 0$ *where* $A_i := A_\gamma^{\pi_{\theta_k}}(s_i, a_i)$.

Proof. The problem (5.31) is equivalent to:

$$\begin{aligned} \underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad & \sum_{i=1}^m A_\gamma^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \\ \text{subject to} \quad & 1-\epsilon \leqslant \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leqslant 1+\epsilon \quad i=1,\ldots,m \\ & \sum_{i=1}^m \left(\frac{\pi(a_i|s_i)-\pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2 \leqslant \delta \end{aligned} \tag{5.32}$$

This problem is clearly convex. $\pi_{\theta_k}(a_i|s_i), i = 1,\ldots,m$ is a feasible solution where the inequality constraint holds strictly. Strong duality thus holds according to Slater's constraint qualification. To solve (5.32), we can therefore solve the related Lagrangian problem for fixed λ:

$$\begin{aligned} \underset{\pi(a_1|s_1),\ldots,\pi(a_m|s_m)}{\text{maximize}} \quad & \sum_{i=1}^m \left[A_\gamma^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} - \lambda\left(\frac{\pi(a_i|s_i)-\pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2\right] \\ \text{subject to} \quad & 1-\epsilon \leqslant \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leqslant 1+\epsilon \quad i=1,\ldots,m \end{aligned} \tag{5.33}$$

which is separable and decomposes into m separate problems, one for each s_i:

$$\begin{aligned} \underset{\pi(a_i|s_i)}{\text{maximize}} \quad & A_\gamma^{\pi_{\theta_k}}(s_i,a_i)\frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} - \lambda\left(\frac{\pi(a_i|s_i)-\pi_{\theta_k}(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)}\right)^2 \\ \text{subject to} \quad & 1-\epsilon \leqslant \frac{\pi(a_i|s_i)}{\pi_{\theta_k}(a_i|s_i)} \leqslant 1+\epsilon \end{aligned} \tag{5.34}$$

The solution to the unconstrained problem without the constraint is:

$$\pi^*(a_i|s_i) = \pi_{\theta_k}(a_i|s_i)\left(1 + \frac{A_\gamma^{\pi_{\theta_k}}(s_i, a_i)}{2\lambda}\right) \tag{5.35}$$

Now consider the full constrained problem . If $A_\gamma^{\pi_{\theta_k}}(s_i, a_i) \geqslant 0$ and $\pi^*(a_i|s_i) > \pi_{\theta_k}(a_i|s_i)(1+\epsilon)$, the optimal solution is $\pi_{\theta_k}(a_i|s_i)(1+\epsilon)$. Similarly, If $A_\gamma^{\pi_{\theta_k}}(s_i, a_i) < 0$ and $\pi^*(a_i|s_i) < \pi_{\theta_k}(a_i|s_i)(1-\epsilon)$, the optimal solution is $\pi_{\theta_k}(a_i|s_i)(1-\epsilon)$. Rearranging the terms gives (5.31). To obtain λ, we can perform a line search over λ so that the second constraint (5.32) is binding. □

To simplify the algorithm, we treat λ as a hyper-parameter rather than δ. After solving for π^*, we seek a parameterized policy π_θ that is close to π^* by minimizing their mean square error over sampled states and actions, i.e. by updating θ in the negative direction of $\nabla_\theta \sum_i (\pi_\theta(a_i|s_i) - \pi^*(a_i|s_i))^2$. This loss is used for supervised training instead of the KL because we take estimates before forming the optimization problem. Thus, the optimal values for the decision variables do not completely characterize a distribution. We refer to this approach as SPU with the L^∞ constraint.

5.4 Extension to Continuous State and Action Spaces

The methodology developed in the previous section also applies to continuous state and action spaces. In this section, we outline the modifications that are necessary for the continuous case.

We first modify the definition of $d_{\pi,\gamma}$ by replacing $P_\pi(s_t = s)$ with $\frac{d}{ds}P_\pi(s_t \leqslant s)$ so that $d_{\pi,\gamma}(s)$ becomes a density function over the state space. With this modification, the definition of $\bar{D}_{\mathrm{KL}}(\pi \parallel \pi_k)$ and the approximation (4.2) are unchanged. The SPU framework described in Section 4 is also unchanged.

Consider now the non-parameterized optimization problem with aggregate and disaggregate

constraints (5.4), but with continuous state and action space:

$$\begin{aligned} \underset{\pi\in\Pi}{\text{maximize}} \quad & \int d_{\pi_{\theta_{k'},\gamma}}(s) \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}[A^{\pi_{\theta_k}}(s,a)]ds \\ \text{subject to} \quad & \int d_{\pi_{\theta_{k'},\gamma}}(s) D_{\mathrm{KL}}\left(\pi\|\pi_{\theta_k}\right)[s]ds \leqslant \delta \\ & D_{\mathrm{KL}}\left(\pi\|\pi_{\theta_k}\right)[s] \leqslant \epsilon \text{ for all } s \end{aligned} \tag{5.36}$$

Theorem 5.1 holds although its proof needs to be slightly modified as follows. It is straightforward to show that (5.36) remains a convex optimization problem. We can therefore solve (5.36) by solving the Lagrangian (5.7) with the sum replaced with an integral. This problem again decomposes with separate problems for each $s \in \mathcal{S}$ giving exactly the same equations (5.8). The proof then proceeds as in the remainder of the proof of Theorem 5.1.

Theorem 5.3 and 5.5 are also unchanged for continuous action spaces. Their proofs require slight modifications, as in the proof of Theorem 5.1.

5.5 Experiments

Extensive experimental results demonstrate SPU outperforms recent state-of-the-art methods for environments with continuous or discrete action spaces. We provide ablation studies to show the importance of the different algorithmic components, and a sensitivity analysis to show that SPU's performance is relatively insensitive to hyper-parameter choices. There are two definitions we use to conclude A is more sample efficient than B: (i) A takes fewer environment interactions to achieve a pre-defined performance threshold [Kakade 2003]; (ii) the averaged final performance of A is higher than that of B given the same number environment interactions [Schulman et al. 2017b]. .

5.5.1 Results on Mujoco

We will first present experimental results on the MuJoCo robotic simulation environments [Todorov et al. 2012]. In terms of final performance averaged over all available ten Mujoco environments and ten different seeds in each, SPU with L^∞ constraint (Section 5.3.3) and SPU with forward KL constraints (Section 5.3.1) outperform TRPO by 6% and 27% respectively. Since the forward-KL approach is our best performing approach, we focus subsequent analysis on it and hereafter refer to it as SPU, an algorithmic outline of which was presented in Algorithm 3. SPU also outperforms PPO by 17%. Figure 5.2 illustrates the performance of SPU versus TRPO, PPO.

To ensure that SPU is not only better than TRPO in terms of performance gain early during training, we further retrain both policies for 3 million timesteps (Figure 5.3). Again here, SPU outperforms TRPO by 28% . .

5.5.2 Ablation Studies for Mujoco

The indicator variable in (5.18) enforces the disaggregated constraint. We refer to it as *per-state acceptance*. Removing this component is equivalent to removing the indicator variable. We refer to using $\sum_i D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i]$ to determine the number of training epochs as *dynamic stopping*. Without this component, the number of training epochs is a hyperparameter. We also tried removing $\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i]$ from the gradient update step in (5.18). Table 5.1 illustrates the contribution of the different components of SPU to the overall performance. The third row shows that the term $\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_i]$ makes a crucially important contribution to SPU. Furthermore, per-state acceptance and dynamic stopping are both also important for obtaining high performance, with the former playing a more central role. When a component is removed, the hyperparameters are re-tuned to ensure that the best possible performance is obtained with the alternative (simpler) algorithm.

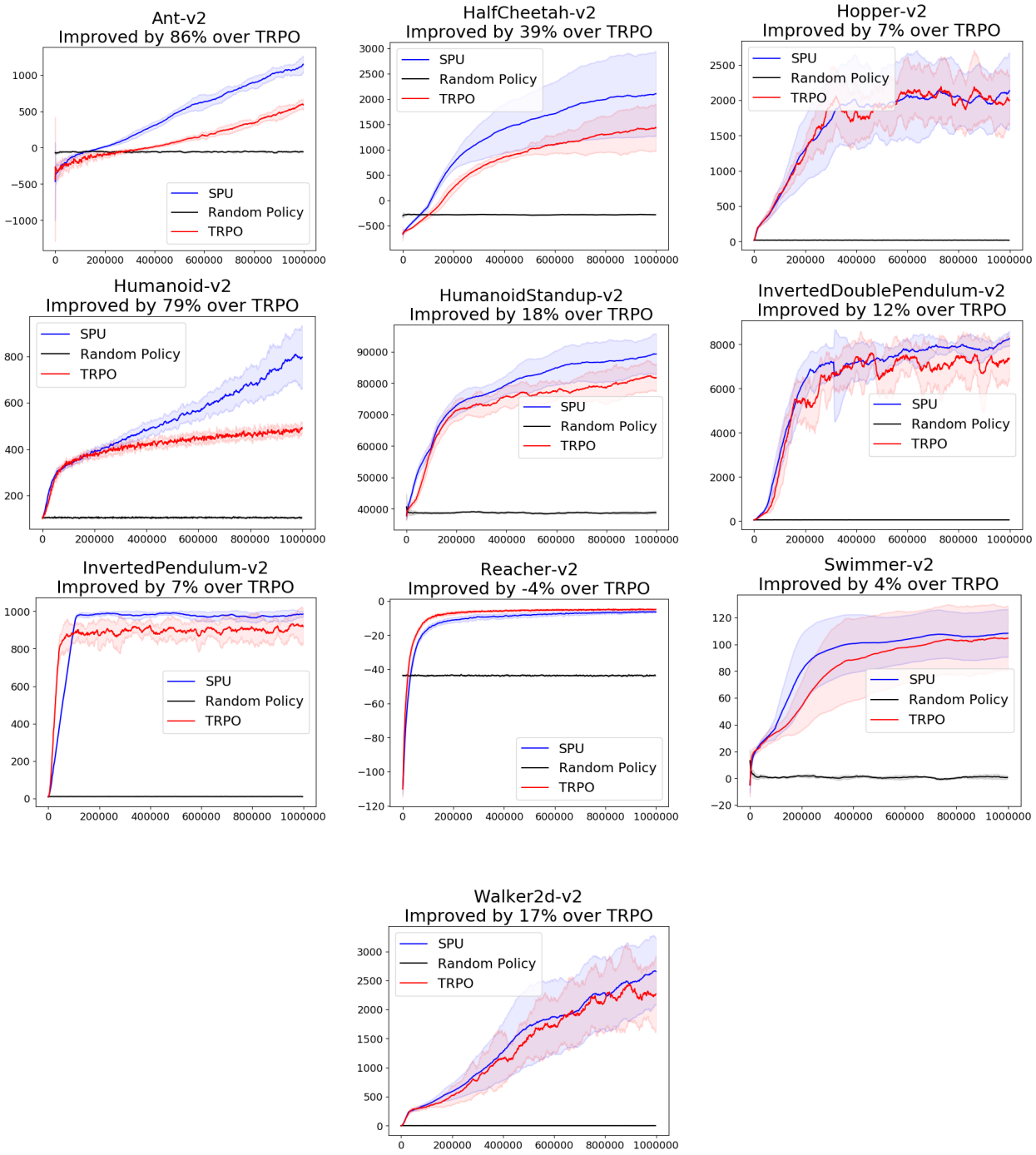


Figure 5.2: SPU versus TRPO, PPO on 10 Mujoco environments in 1 million timesteps. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

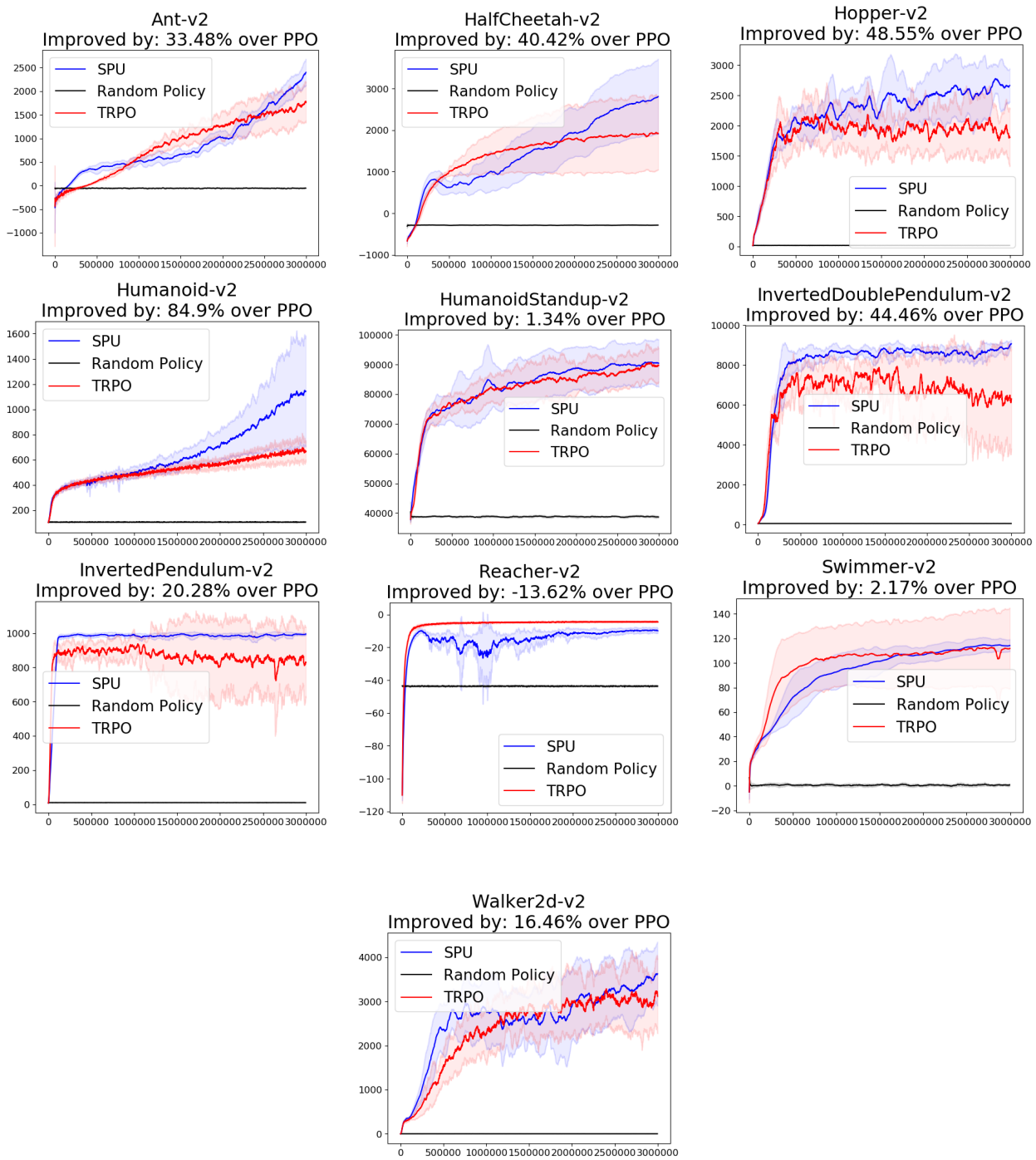


Figure 5.3: SPU versus TRPO, PPO on 10 Mujoco environments in 3 million timesteps. The x-axis indicates timesteps. The y-axis indicates the average episode reward of the last 100 episodes.

Table 5.1: Ablation study for SPU

Approach	Percentage better than TRPO	Performance vs. original algorithm
Original Algorithm	27%	0%
No grad KL	4%	- 85%
No dynamic stopping	24%	- 11%
No per-state acceptance	9%	- 67%

5.5.3 Sensitivity Analysis on Mujoco

To demonstrate the practicality of SPU, we show that its high performance is insensitive to hyper-parameter choice. One way to show this is as follows: for each SPU hyper-parameter, select a reasonably large interval, randomly sample the value of the hyper parameter from this interval, and then compare SPU (using the randomly chosen hyperparameter values) with TRPO. We sampled 100 SPU hyperparameter vectors (each vector including $\delta, \epsilon, \lambda$), and for each one determined the relative performance with respect to TRPO. Since values for SPU hyperparameter are randomly sampled, the percentage improvement of SPU over TRPO becomes a random variable. Figure 5.4 illustrates the CDF of this random variable.

First, we found that for all 100 random hyperparameter value samples, SPU performed better than TRPO. 75% and 50% of the samples outperformed TRPO by at least 18% and 21% respectively. The full CDF is given in Figure 5.4 in the Appendix. We can conclude that SPU's superior performance is largely insensitive to hyperparameter values.

5.5.4 Results on Atari

Mania et al. [2018] demonstrated that neural networks are not needed to obtain high performance in many MuJoCo environments. To conclusively evaluate SPU, we compare it against PPO on the Atari Arcade Learning Environments [Mnih et al. 2013]. Using the same network architecture and

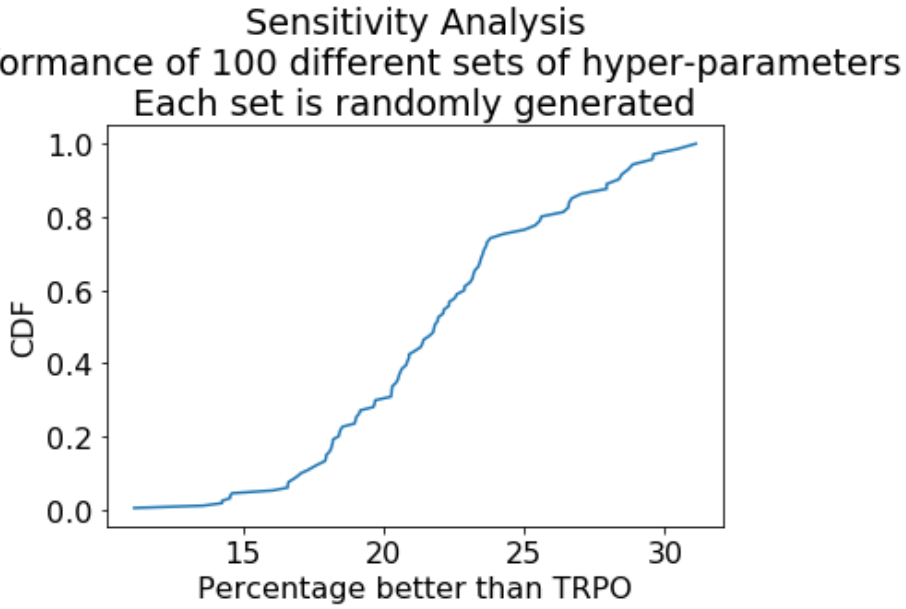


Figure 5.4: Sensitivity Analysis for SPU

hyperparameters, we learn to play 60 Atari games from raw pixels and rewards. This is highly challenging because of the diversity in the games and the high dimensionality of the observations.

Here, we compare SPU against PPO because PPO outperforms TRPO by 9% in MuJoCo. Averaged over 60 Atari environments and 20 seeds, SPU is 55% better than PPO in terms of averaged final performance. Figure 5.5 provides a high-level overview of the result. The dots in the shaded area represent environments where their performances are roughly similar. The dots to the right of the shaded area represent environment where SPU is more sample efficient than PPO. We can draw two conclusions: (i) In 36 environments, SPU and PPO perform roughly the same ; SPU clearly outperforms PPO in 15 environments while PPO clearly outperforms SPU in 9; (ii) In those 15+9 environments, the extent to which SPU outperforms PPO is much larger than the extent to which PPO outperforms SPU. SPU's high performance in both the MuJoCo and Atari domains demonstrates its high performance and generality.

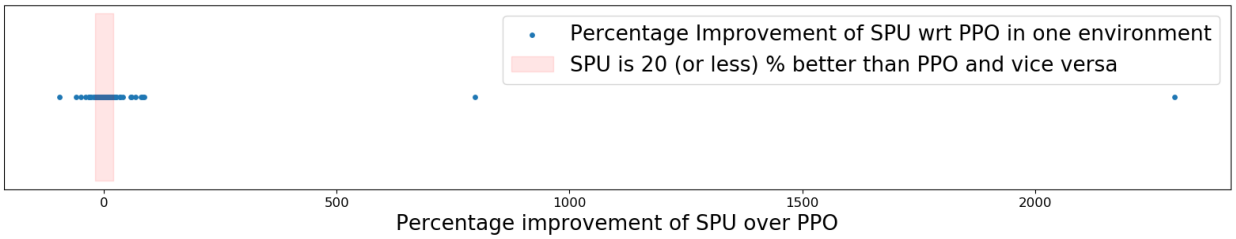


Figure 5.5: High-level overview of results on Atari

5.5.5 Implementation Details

MuJoCo

As was done in the previous Chapter and in [Schulman et al. 2017b], the policy is parameterized by a fully-connected feed-forward neural network with two hidden layers, each with 64 units and tanh nonlinearities. The policy outputs the mean of a Gaussian distribution with state-independent variable standard deviations. The action dimensions are assumed to be independent. The probability of an action is given by the multivariate Gaussian probability distribution function. The baseline used in the advantage value calculation is parameterized by a similarly sized neural network, trained to minimize the MSE between the sampled states TD(λ) returns and the their predicted values. For both the policy and baseline network, SPU and TRPO use the same architecture. To calculate the advantage values, we use Generalized Advantage Estimation [Schulman et al. 2016]. States are normalized by dividing the running mean and dividing by the running standard deviation before being fed to any neural networks. The advantage values are normalized by dividing the batch mean and dividing by the batch standard deviation before being used for policy update. The TRPO result is obtained by running the TRPO implementation provided by OpenAI [Dhariwal et al. 2017]. At every iteration, SPU collects 2048 samples before updating the policy and the baseline network. For both networks, gradient descent is performed using Adam [Kingma and Ba 2015] with step size 0.0003, minibatch size of 64. The step size is linearly annealed to 0 over the course of training. γ and λ for GAE [Schulman et al. 2016] are set

to 0.99 and 0.95 respectively. For SPU, $\delta, \epsilon, \lambda$ and the maximum number of epochs per iteration are set to 0.05/1.2, 0.05, 1.3 and 30 respectively. In the sensitivity analysis, the ranges of values for the hyperparameters $\delta, \epsilon, \lambda$ and maximum number of epochs are $[0.05, 0.07]$, $[0.01, 0.07]$, $[1.0, 1.2]$ and $[5, 30]$ respectively.

Atari

Unless otherwise mentioned, the hyperparameter values are the same as in the MuJoCo expreriments. The policy is parameterized by a convolutional neural network with the same architecture as described in [Mnih et al. 2015]. The output of the network is passed through a ReLu, linear and softmax layer in that order to give the action distribution. The output of the network is also passed through a different linear layer to give the baseline value. States are normalized by dividing by 255 before being fed into any network. The PPO result is obtained by running the PPO implementation provided by OpenAI [Dhariwal et al. 2017]. 8 different processes run in parallel to collect timesteps. At every iteration, each process collects 256 samples before updating the policy and the baseline network. Each process calculates its own update to the network's parameters and the updates are averaged over all processes before being used to update the network's parameters. Gradient descent is performed using Adam [Kingma and Ba 2015] with step size 0.0001. In each process, random number generators are initialized with a different seed according to the formula $process\_seed = experiment\_seed + 10000 * process\_rank$. Training is performed for 10 million timesteps for both SPU and PPO. For SPU, $\delta, \epsilon, \lambda$ and the maximum number of epochs per iteration are set to 0.02, δ/1.3, 1.1 and 9 respectively.

5.6 Related Work

Natural gradient [Amari 1998] was first introduced to policy gradient by Kakade [2001b] and then in Peters and Schaal [2008a,b]; Schulman et al. [2015], which we will collectively refer to as NPG/TRPO. As we have mentioned in the previous chapter, algorithmically, NPG/TRPO finds the

gradient update by solving the sample efficiency problem (5.1) with $D(\pi_\theta, \pi_{\theta_k}) = \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})$, i.e., use the aggregate KL-divergence for the policy proximity constraint. NPG/TRPO addresses this problem in the parameter space $\theta \in \Theta$. First, it approximates the objective in (5.1) with first-order Taylor approximation and $\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})$ using a similar second-order method. Second, it uses samples from π_{θ_k} to form estimates of these two approximations. Third, using these estimates (which are functions of θ), it solves for the optimal θ^*.

SPU takes a very different approach by first (i) posing and solving the optimization problem in the non-parameterized policy space, and then (ii) solving a supervised regression problem to find a parameterized policy that is near the optimal non-parameterized policy. A recent paper, Guided Actor Critic (GAC), independently proposed a similar decomposition [Tangkaratt et al. 2018]. However, GAC is much more restricted in that it considers only one specific constraint criterion (aggregated reverse-KL divergence) and applies only to continuous action spaces. Furthermore, GAC incurs significantly higher computational complexity, e.g. at every update, it minimizes the dual function to obtain the dual variables using Sequential Least-Squares Quadratic Programming (SLSQP) methods [Kraft et al. 1988]. Maximum Aposteriori Policy Optimisation (MPO) also independently propose a similar decomposition [Abdolmaleki et al. 2018]. MPO uses much more complex machinery, namely, Expectation Maximization to address the DRL problem. However, MPO has only demonstrates preliminary results on problems with discrete actions whereas our approach naturally applies to problems with either discrete or continuous actions. In both GAC and MPO, working in the non-parameterized space is a by-product of applying the main ideas in those papers to DRL. Our paper demonstrates that the decomposition alone is a general and useful technique for solving constrained policy optimization.

PPO-clipped [Schulman et al. 2017b] takes a very different approach to TRPO. At each iteration, PPO makes many gradient steps while only using the data from π_{θ_k}. Without the clipping, PPO is the approximation in (4.2). The clipping is analogous to the constraint in (5.1) in that it has the goal of keeping π_θ close to π_{θ_k}. Indeed, the clipping keeps $\pi_\theta(a_t|s_t)$ from becoming neither much

larger than $(1+\epsilon)\pi_{\theta_k}(a_t|s_t)$ nor much smaller than $(1-\epsilon)\pi_{\theta_k}(a_t|s_t)$. Thus, although the clipped PPO objective does not squarely fit into the optimization framework in (5.1), it is quite similar in spirit. We note that the PPO paper considers adding the KL penalty to the objective function, whose gradient is similar to ours. However, this form of gradient was demonstrated to be inferior to Clipped-PPO. To the best of our knowledge, it is only until our work that such form of gradient is demonstrated to outperform Clipped-PPO.

Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [Wu et al. 2017] proposed using Kronecker-factored approximation curvature (K-FAC) to update both the policy gradient and critic terms, giving a more computationally efficient method of calculating the natural gradients. ACER [Wang et al. 2017] exploits past episodes, linearizes the KL divergence constraint, and maintains an average policy network to enforce the KL divergence constraint. In future work, it would of interest to extend the SPU methodology to handle past episodes. In contrast to bounding the KL divergence on the action distribution as we have done in this work, Relative Entropy Policy Search [Peters et al. 2010] considers bounding the joint distribution of state and action and was only demonstrated to work for small problems.

The general idea of first finding a non-parametric solution then projecting back into the parameter space is also used in other sub-fields of machine learning. Notably in variational inference [Blei et al. 2017] where a non-parametric posterior distribution is projected into a space of parameterized distributions.

5.7 Conclusion

We developed a novel policy-space methodology, which can be used to compare and contrast various sample-efficient reinforcement learning algorithms, including PPO and different versions of TRPO. The methodology can also be used to study many other forms of constraints, such as constraining the aggregated and disaggregated reverse KL-divergence. We also proposed a new

sample-efficient class of algorithms called SPU, for which there is significant flexibility in how we set the targets.

As compared to PPO, our experimental results show that SPU with simple target functions can lead to improved sample-efficiency performance without increasing wall-clock time. In the future, it may be possible to achieve further gains with yet-to-be-explored classes of target functions, annealing the targets, and changing the number of passes through the data.

6 Constrained Reinforcement Learning

6.1 Introduction

In the prior chapters, we allowed the agent to freely explore the environment to obtain desirable behavior, provided that it leads to performance improvement. No regard is given to whether the agent's behavior may lead to negative or harmful consequences. Consider for instance the task of controlling a robot, certain maneuvers may damage the robot, or worse harm people around it. RL safety [Amodei et al. 2016] is a pressing topic in modern reinforcement learning research and imperative to applying reinforcement learning to safety-critical settings such as autonomous driving [Kendall et al. 2019; Scheel et al. 2021] and healthcare [Prasad et al. 2017; Komorowski et al. 2018; Gottesman et al. 2019].

Constrained Markov Decision Processes (CMDP) [Kallenberg 1983; Ross 1985; Beutler and Ross 1985; Ross and Varadarajan 1989; Altman 1999] provide a principled mathematical framework for dealing with such problems as it allows us to naturally incorporate safety criteria in the form of constraints. In low-dimensional finite settings, an optimal policy for CMDPs with known dynamics can be found by linear programming [Kallenberg 1983] or Lagrange relaxation [Ross 1985; Beutler and Ross 1985].

While we can solve problems with small state and action spaces via linear programming and

value iteration, function approximation is required in order to generalize over large state spaces. Based on recent advances in local policy search methods [Kakade and Langford 2002; Peters and Schaal 2008b; Schulman et al. 2015], Achiam et al. [2017] proposed the Constrained Policy Optimization (CPO) algorithm. However policy updates for the CPO algorithm involve solving an optimization problem through Taylor approximations and inverting a high-dimensional Fisher information matrix. These approximations often result in infeasible updates which would require additional recovery steps, this could sometimes cause updates to be backtracked leading to a waste of samples. Moreover, the theoretical guarantees for the CPO algorithm assume that the cost constraint takes the form of a discounted sum. Tessler et al. [2019] pointed out that trust-region based methods such as the Constrained Policy Optimization (CPO) algorithm [Achiam et al. 2017] cannot be used for non-discounted cost constraints.

In this chapter, we first show that the theoretical guarantees for the CPO algorithm can in fact be extended to non-discounted cost constraints. We then propose the First Order Constrained Optimization in Policy Space (FOCOPS) algorithm. FOCOPS attempts to answer the following question: given some current policy, what is the best constraint-satisfying policy update? Inspired by ideas introduced in Chapter 5, FOCOPS provides a solution to this question in the form of a two-step approach. First, we will show that the best policy update has a near-closed form solution when attempting to solve for the optimal policy in the nonparametric policy space rather than the parameter space. However in most cases, this optimal policy is impossible to evaluate. Hence we project this policy back into the parametric policy space. This can be achieved by drawing samples from the current policy and evaluating a loss function between the parameterized policy and the optimal policy we found in the nonparametric policy space. Theoretically, FOCOPS has an approximate upper bound for worst-case constraint violation throughout training. Practically, FOCOPS is extremely simple to implement since it only utilizes first order approximations. We further test our algorithm on a series of challenging high-dimensional continuous control tasks and found that FOCOPS achieves better performance while maintaining approximate constraint

satisfaction compared to current state of the art approaches, in particular second-order approaches such as CPO.

6.2 Background: Constrained Markov Decision Processes

A Constrained Markov Decision Processes (CMDP) [Kallenberg 1983; Ross 1985; Altman 1999] is an MDP equipped with a constraint set $\Pi_c = \{\pi \in \Pi : J_c(\pi) \leqslant b\}$ where $J_c(\pi)$ is the cost constraint which depends on the scalar cost function $c : \mathcal{S} \times \mathcal{A} \to [c_{\min}, c_{\max}]$, b is the constraint bound. The cost $c(s, a)$ is the immediate cost incurred by the agent in state s after taking action a. The goal of a CMDP problems finds a policy π *within the constraint set* that maximizes an agent's long-run reward, i.e.

$$\pi^* = \underset{\pi \in \Pi_c}{\operatorname{argmax}} J(\pi). \tag{6.1}$$

where again we use $J(\pi)$ to denote some policy performance measure.

The cost constraint can take various forms, some common examples include:

- Per-state cost constraint [Dalal et al. 2018]

$$c(s) \leqslant b \text{ for all } s \in \mathcal{S}.$$

- Discounted cost constraint [Achiam et al. 2017]

$$\rho_{c,\gamma}(\pi) := \mathop{\mathbb{E}}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t)\right]$$

- Average cost constraint [Altman 1999]

$$\rho_c(\pi) := \lim_{N \to \infty} \frac{1}{N} \mathop{\mathbb{E}}_{\tau \sim \pi}\left[\sum_{t=0}^{N-1} c(s_t, a_t)\right]$$

- Sample path constraint [Ross 1985; Ross and Varadarajan 1991]

$$P_\pi\left(\lim_{N\to\infty}\frac{1}{N}\sum_{t=0}^{N-1}c(s_t,a_t)\leqslant b\right)=1$$

- Entropy constraint [Ziebart et al. 2008]

$$\mathop{\mathbb{E}}_{s}\left[-\int_a\pi(a|s)\log\pi(a|s)da\right]\geqslant b$$

- Value-at-Risk (VaR) constraint [Chow et al. 2017]

$$\text{VaR}_\alpha=\min_z\left\{P_\pi\left(\sum_{t=0}^{\infty}\gamma^t c(s_t,a_t)\leqslant z\right)\geqslant\alpha\right\}\leqslant b$$

In this thesis, We consider two forms of constraint sets: the average cost constraint set $\{\pi\in\Pi:\rho_c(\pi)\leqslant b\}$ and the discounted cost constraint set $\{\pi\in\Pi:\rho_{c,\gamma}(\pi)\leqslant b\}$. We will also restrict our attention to the case of a single cost constraints, however many of the results we introduce can be naturally extended to accommodate for multiple constraints.

Similar to the reward function, the form of the cost function $c(s,a)$ is usually specified by user depending specific application needs. For instance, in many applications involving robotic locomotive tasks, it is reasonable for safety reasons to set a speed limit for the robot. However just like the task of specifiying a reward function, defining a cost function and choosing a cost constraint often requires specific domain knowledge and can be very challenging in practice [Amodei et al. 2016]. More recently, safety constraints have also been expressed in other more natural forms such as human preferences [Christiano et al. 2017] or natural language [Luketina et al. 2019].

Analogous to V_γ^π, Q_γ^π, A_γ^π, $\bar{V}^\pi$, $\bar{Q}^\pi$, $\bar{A}^\pi$ for the rewards. We can define $V_{c,\gamma}^\pi$, $Q_{c,\gamma}^\pi$, $A_{c,\gamma}^\pi$, $\bar{V}_c^\pi$, $\bar{Q}_c^\pi$, $\bar{A}_c^\pi$ by replacing the reward function with the cost function.

6.3 Constrained RL as Local Policy Search

We first consider the case of discounted costs. Recall the local policy search problem introduced in Chapter 3. We can augment this problem to solve for CMDPs by adding the cost constraint, i.e.

$$\begin{aligned} \underset{\pi \in \Pi}{\text{maximize}} \quad & J(\pi) \\ \text{subject to} \quad & \rho_{c,\gamma}(\pi) \leqslant b \\ & D(\pi, \pi_k) \leqslant \delta. \end{aligned} \tag{6.2}$$

However solving CMDPs directly can be challenging and sample inefficient since after each policy update, additional samples need to be collected from the new policy in order to evaluate whether the constraints are satisfied.

Achiam et al. [2017] proposed replacing the cost constraint with a surrogate cost function which evaluates the constraint $\rho_{c,\gamma}(\pi_\theta)$ using samples collected from the current policy π_{θ_k}. This surrogate function is shown to be a good approximation to $\rho_{c,\gamma}(\pi_\theta)$ when π_θ and π_{θ_k} are close w.r.t. the KL divergence. Based on this idea, the CPO algorithm [Achiam et al. 2017] performs policy updates as follows: given some policy π_{θ_k}, the new policy $\pi_{\theta_{k+1}}$ is obtained by solving the optimization problem

$$\begin{aligned} \underset{\pi_\theta \in \Pi_\theta}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_{\theta_k}, \gamma} \\ a \sim \pi_\theta}} \left[A_\gamma^{\pi_{\theta_k}}(s, a) \right] \\ \text{subject to} \quad & \tilde{\rho}_{c,\gamma}(\pi_\theta) \leqslant b \\ & \mathop{\mathbb{E}}_{s \sim d_{\pi_{\theta_k}, \gamma}} \left[D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] \right] \leqslant \delta. \end{aligned} \tag{6.3}$$

Here,

$$\tilde{\rho}_{c,\gamma}(\pi_\theta) := \rho_{c,\gamma}(\pi_{\theta_k}) + \frac{1}{1-\gamma} \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_{\theta_k}, \gamma} \\ a \sim \pi_\theta}} \left[A_{c,\gamma}^{\pi_{\theta_k}}(s, a) \right] \tag{6.4}$$

is a surrogate cost function used to approximate the cost constraint. Note that (6.4) can be evaluated using samples from π_{θ_k}. By Corollary 2 of Achiam et al. [2017] and (3.20):

$$\left|\rho_{c,\gamma}(\pi_\theta) - \tilde{\rho}_{c,\gamma}(\pi_\theta)\right| \leqslant \frac{\gamma\epsilon_{c,\gamma}}{(1-\gamma)^2}\sqrt{2\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})} \tag{6.5}$$

where $\epsilon_{c,\gamma} = \max_s \left|\mathbb{E}_{a\sim\pi'}[A^{\pi}_{c,\gamma}(s,a)]\right|$. This shows that the surrogate cost is a good approximation to $\rho_{c,\gamma}(\pi_\theta)$ when π_θ and π_{θ_k} are close w.r.t. the KL divergence. When $\pi_{\theta_{k+1}}$ is the solution to (6.3), it satisfies the following bound for worst-case guarantee for cost constraint satisfaction (Proposition 2 of Achiam et al. [2017]):

$$\rho_{c,\gamma}(\pi_{\theta_{k+1}}) \leqslant b + \frac{\sqrt{2\delta}\gamma\epsilon_{c,\gamma}}{(1-\gamma)^2} \tag{6.6}$$

However, this framework is problematic when the cost constraint is undiscounted. Consider the corresponding average cost problem to (6.3):

$$\begin{aligned} \underset{\pi_\theta\in\Pi_\theta}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}}\\ a\sim\pi_\theta}}[\bar{A}^{\pi_{\theta_k}}(s,a)] \\ \text{subject to} \quad & \tilde{\rho}_c(\pi_\theta) \leqslant b \\ & \bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k}) \leqslant \delta. \end{aligned} \tag{6.7}$$

where

$$\tilde{\rho}_c(\pi_\theta) := \rho_c(\pi_{\theta_k}) + \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}}\\ a\sim\pi_{\theta_k}}}[A_c^{\pi_\theta}(s,a)] \tag{6.8}$$

is the average cost surrogate function. We can easily show that

$$\lim_{\gamma\to 1}(1-\gamma)(\rho_{c,\gamma}(\pi_\theta) - \tilde{\rho}_{c,\gamma}(\pi_\theta)) = \rho_c(\pi_\theta) - \tilde{\rho}_c(\pi_\theta) \quad \text{and} \quad \lim_{\gamma\to 1}\frac{\gamma\epsilon_{c,\gamma}}{1-\gamma}\sqrt{2\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})} = \infty$$

However, by Theorem 3.5[1] and (3.20):

$$|\rho_c(\pi_\theta) - \tilde{\rho}_c(\pi_\theta)| \leqslant \xi_c^{\pi_\theta} \sqrt{2\bar{D}_{\mathrm{KL}}(\pi_\theta \parallel \pi_{\theta_k})} \tag{6.9}$$

where $\xi_c^{\pi_\theta} = (\kappa^* - 1) \max_s \mathbb{E}_{a\sim\pi_\theta} |A_c^{\pi_{\theta_k}}(s,a)|$. We then have the following result:

Proposition 6.1. *Suppose π_θ and π_{θ_k} satisfy the constraints $\tilde{\rho}_c(\pi_\theta) < b$ and $\bar{D}_{KL}(\pi_\theta \parallel \pi_{\theta_k}) \leqslant \delta$, then*

$$\rho_c(\pi_\theta) \leqslant b + \xi_c^{\pi_\theta} \sqrt{2\delta} \tag{6.10}$$

The upper-bound in Proposition 6.1 provides a worst-case constraint violation guarantee when π_θ is the solution to the average-cost variant of (6.3). It is an undiscounted parallel to Proposition 2 in Achiam et al. [2017] which provides a similar guarantee for the discounted case. It shows that contrary to what was previously believed [Tessler et al. 2019], Problem (6.3) can easily be modified to accommodate for average cost constraints and still satisfy an upper bound for worst-case constraint violation. In the next section, we will show how the CPO algorithm [Achiam et al. 2017] can be modified for average cost constraints.

6.4 Average Cost CPO

Consider the optimization problem (6.7). Similar to TRPO/ATRPO, we can apply first and second order Taylor approximations to (6.7) which then gives us

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & g^T(\theta - \theta_k) \\ \text{subject to} \quad & \tilde{c} + \tilde{g}^T(\theta - \theta_k) \leqslant 0 \\ & \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \leqslant \delta \end{aligned} \tag{6.11}$$

[1] It is straightforward to show that the theorem still holds when we replace the reward with the cost.

where g, H were defined Section 4.1, $\tilde{c} = \rho(\pi_{\theta_k}) - b$, and

$$\tilde{g} := \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}} \\ a\sim\pi_{\theta_k}}} \left[\nabla_\theta \log \pi_\theta(a|s)|_{\theta=\theta_k} \bar{A}_c^{\pi_{\theta_k}}(s,a) \right] \tag{6.12}$$

is the gradient of the constraint. Similar to the case of ATRPO, $g, \tilde{g}, H$, and $\tilde{c}$ can all be approximated using samples collected from π_{θ_k}. The term $\bar{A}_c^{\pi_{\theta_k}}(s,a)$ also involves the cost bias-function which can be approximated via a separate cost critic network. The optimization problem (6.11) is a convex optimization problem where strong duality holds, hence it can be solved using a simple Lagrangian argument. The update rule takes the form

$$\theta = \theta_k + \frac{1}{\lambda} H^{-1}(g - \nu\tilde{g}) \tag{6.13}$$

where λ and ν are Lagrange multipliers satisfying [Achiam et al. 2017]:

$$\max_{\lambda,\nu\geqslant 0} -\frac{1}{2\lambda}\left(g^T H^{-1} g + 2\nu g^T H^{-1}\tilde{g} + \nu^2 \tilde{g} H^{-1}\tilde{g}^T\right) + \nu\tilde{c} - \frac{1}{2}\lambda\delta \tag{6.14}$$

The dual problem (6.14) can be solved explicitly [Achiam et al. 2017]. Similar to ATRPO, we use the conjugate gradient method to estimate H and perform a backtracking line search procedure to guarantee approximate constraint satisfaction.

6.5 First Order Constrained Optimization in Policy Space

The CPO/ACPO algorithm attempts to approximately solve (6.3) and (6.7) introduced in the previous section introduces several sources of errors in the process, namely

(i) Sampling error resulting from taking sample trajectories from the current policy.

(ii) Approximation errors resulting from Taylor approximations.

(iii) Using the conjugate gradient method in order to calculate the inverse of a Fisher information matrix.

In practice the presence of these errors require the CPO/ACPO algorithm to take additional steps during each update in the training process in order to recover from constraint violations, this is often difficult to achieve and may not always work well in practice. In this section, we introduce a simple first-order method which is able to eliminate the last two sources of error and outperform CPO/ACPO.

Instead of solving (6.3) and (6.7) directly, we use a two-step approach inspired by the high-level ideas introduced in Chapter 5:

1. Given policy π_{θ_k}, find an *optimal update policy* π^* by solving the optimization problem from (6.3) and (6.7) in the nonparameterized policy space.

2. Project the policy found in the previous step back into the parameterized policy space Π_θ by solving for the closest policy $\pi_\theta \in \Pi_\theta$ to π^* in order to obtain $\pi_{\theta_{k+1}}$.

Subsequently, we will consider the discounted cost problem in (6.3), however our results can be naturally extended to the average cost case.

6.5.1 Finding the Optimal Update Policy

In the first step, we consider the optimization problem

$$\begin{aligned} \underset{\pi\in\Pi}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k},\gamma} \\ a\sim\pi}} [A_\gamma^{\pi_{\theta_k}}(s,a)] \\ \text{subject to} \quad & \tilde{\rho}_{c,\gamma}(\pi) \leqslant b \\ & \bar{D}_{\mathrm{KL}}(\pi \parallel \pi_{\theta_k}) \leqslant \delta. \end{aligned} \tag{6.15}$$

Note that this problem is almost identical to Problem (6.3) except the parameter of interest is

now the nonparameterized policy π and not the policy parameter θ. We can show that Problem (6.15) admits the following solution:

Theorem 6.2. *Let $\tilde{b} = (1-\gamma)(b - \rho_{c,\gamma}(\pi_{\theta_k}))$. If π_{θ_k} is a feasible solution, the optimal policy for (6.15) takes the form*

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_{\mu,\nu}(s)} \exp\left(\frac{1}{\mu}\left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right)\right) \tag{6.16}$$

where $\mathcal{Z}_{\mu,\nu}(s)$ is the partition function which ensures (6.16) is a valid probability distribution, μ and ν are solutions to the optimization problem:

$$\min_{\mu,\nu \geqslant 0} \; \mu\delta + \nu\tilde{b} + \mu \mathop{\mathbb{E}}_{s\sim d_{\pi_{\theta_k},\gamma}} \left[\log \mathcal{Z}_{\mu,\nu}(s)\right] \tag{6.17}$$

Proof. We will begin by showing that Problem (6.15) is convex w.r.t. $\pi = \{\pi(a|s) : s \in \mathcal{S}, a \in \mathcal{A}\}$. First note that the objective function is linear w.r.t. π. Since $\rho_c(\pi_{\theta_k})$ is a constant w.r.t. π, the first constraint is linear. The trust region constraint can be rewritten as $\sum_s d_{\pi_{\theta_k},\gamma}(s) D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \leqslant \delta$, the KL divergence is convex w.r.t. its first argument, therefore the trust region constraint which is a linear combination of convex functions is also convex. Since π_{θ_k} satisfies the cost constraint and is also an interior point within the set given by the trust region constraint ($D_{\mathrm{KL}}\left(\pi_{\theta_k} \| \pi_{\theta_k}\right) = 0$, and $\delta > 0$), therefore Slater's constraint qualification holds, strong duality holds.

We can therefore solve for the optimal value of Problem (6.15) p^* by solving the corresponding dual problem. Let

$$L(\pi,\mu,\nu) = \mu\delta + \nu\tilde{b} + \mathop{\mathbb{E}}_{s\sim d_{\pi_{\theta_k},\gamma}} \left[\mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)} \left[A_\gamma^{\pi_{\theta_k}}(s,a)\right] - \nu \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)} \left[A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right] - \mu D_{\mathrm{KL}}\left(\pi \| \pi_{\theta_k}\right)[s] \right] \tag{6.18}$$

Therefore,

$$p^* = \max_{\pi\in\Pi} \min_{\mu,\nu\geqslant 0} L(\pi,\mu,\nu) = \min_{\mu,\nu\geqslant 0} \max_{\pi\in\Pi} L(\pi,\mu,\nu) \tag{6.19}$$

where we invoked strong duality in the second equality. We note that if π^*, μ^*, ν^* are optimal for

(6.19), π^* is also optimal for Problem (6.15) [Boyd et al. 2004].

Consider the inner maximization problem in (6.19), we can decompose this problem into separate problems, one for each s. This gives us an optimization problem of the form,

$$\begin{aligned} \underset{\pi}{\text{maximize}} \quad & \mathop{\mathbb{E}}_{a\sim\pi(\cdot|s)}\left[A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) - \mu(\log\pi(a|s) - \log\pi_{\theta_k}(a|s))\right] \\ \text{subject to} \quad & \sum_a \pi(a|s) = 1 \\ & \pi(a|s) \geqslant 0 \quad \text{for all } a \in \mathcal{A} \end{aligned} \tag{6.20}$$

which is equivalent to the inner maximization problem in (6.19). This is clearly a convex optimization problem which we can solve using a simple Lagrangian argument. We find that the optimal solution π^* to (6.20) takes the form

$$\pi^*(a|s) = \frac{\pi_{\theta_k}(a|s)}{\mathcal{Z}_{\mu,\nu}(s)} \exp\left(\frac{1}{\mu}\left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right)\right)$$

Plugging π^* back into Equation 6.19 gives us

$$\begin{aligned} p^* &= \min_{\mu,\nu\geqslant 0} \mu\delta + \nu\tilde{b} + \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}} \\ a\sim\pi^*}}\left[A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) - \mu(\log\pi^*(a|s) - \log\pi_{\theta_k}(a|s))\right] \\ &= \min_{\mu,\nu\geqslant 0} \mu\delta + \nu\tilde{b} + \mathop{\mathbb{E}}_{\substack{s\sim d_{\pi_{\theta_k}} \\ a\sim\pi^*}}\left[A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) - \mu(\log\pi_{\theta_k}(a|s) - \log\mathcal{Z}_{\mu,\nu}(s)\right. \\ &\left. + \frac{1}{\mu}(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)) - \log\pi_{\theta_k}(a|s))\right] \\ &= \min_{\mu,\nu\geqslant 0} \mu\delta + \nu\tilde{b} + \mu \mathop{\mathbb{E}}_{s\sim d_{\pi_{\theta_k}}}\left[\log\mathcal{Z}_{\mu,\nu}(s)\right] \end{aligned}$$

□

The form of the optimal policy is intuitive, it gives high probability mass to areas of the state-action space with high return which is offset by a penalty coefficient times the cost advantage.

We will refer to the optimal solution to (6.16) as the *optimal update policy*. We also note that it is possible to extend our results to accommodate for multiple constraints by introducing Lagrange multipliers $\nu_1, \ldots, \nu_m \geqslant 0$, one for each cost constraint and applying a similar duality argument.

It is also straightforward to show that (6.16) satisfies the worst-case constraint violation guarantee in (6.6).

6.5.2 Approximating the Optimal Update Policy

When solving Problem (6.15), we allow π to be in the set of all stationary policies Π thus the resulting π^* is not necessarily in the parameterized policy space Π_θ and we may no longer be able to evaluate or sample from π^*. Therefore in the second step we project the optimal update policy back into the parameterized policy space by minimizing the loss function:

$$\mathcal{L}(\theta) = \mathop{\mathbb{E}}_{s\sim d^{\pi_{\theta_k}}} \left[D_{\mathrm{KL}} \left(\pi_\theta \| \pi^* \right) [s] \right] \tag{6.21}$$

Here $\pi_\theta \in \Pi_\theta$ is some projected policy which we will use to approximate the optimal update policy. We can use first-order methods to minimize this loss function where we make use of the following result:

Corollary 6.3. *The gradient of $\mathcal{L}(\theta)$ takes the form*

$$\nabla_\theta \mathcal{L}(\theta) = \mathop{\mathbb{E}}_{s\sim d^{\pi_{\theta_k}}} \left[\nabla_\theta D_{KL} \left(\pi_\theta \| \pi^* \right) [s] \right] \tag{6.22}$$

where

$$\nabla_\theta D_{KL} \left(\pi_\theta \| \pi^* \right) [s] = \nabla_\theta D_{KL} \left(\pi_\theta \| \pi_{\theta_k} \right) [s] - \frac{1}{\mu} \mathop{\mathbb{E}}_{a\sim\pi_{\theta_k}} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \right] \tag{6.23}$$

Proof. We only need to calculate the gradient of the loss function for a single sampled s. We first

note that,

$$\begin{aligned} D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s] =& -\sum_a \pi_\theta(a|s) \log \pi^*(a|s) + \sum_a \pi_\theta(a|s) \log \pi_\theta(a|s) \\ =& H(\pi_\theta, \pi^*)[s] - H(\pi_\theta)[s] \end{aligned}$$

where $H(\pi_\theta)[s]$ is the entropy and $H(\pi_\theta, \pi^*)[s]$ is the cross-entropy under state s. We expand the cross entropy term which gives us

$$\begin{aligned} H(\pi_\theta, \pi^*)[s] &= -\sum_a \pi_\theta(a|s) \log \pi^*(a|s) \\ &= -\sum_a \pi_\theta(a|s) \log \left(\frac{\pi_{\theta_k}(a|s)}{Z_{\mu,\nu}(s)} \exp\left[\frac{1}{\mu} \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \right] \right) \\ &= -\sum_a \pi_\theta(a|s) \log \pi_{\theta_k}(a|s) + \log \mathcal{Z}_{\mu,\nu}(s) - \frac{1}{\mu} \sum_a \pi_\theta(a|s) \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \end{aligned}$$

We then subtract the entropy term to recover the KL divergence:

$$\begin{aligned} D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s] =& D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] + \log \mathcal{Z}_{\mu,\nu}(s) - \frac{1}{\mu} \sum_a \pi_\theta(a|s) \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \\ =& D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] + \log \mathcal{Z}_{\mu,\nu}(s) - \frac{1}{\mu} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \right] \end{aligned}$$

where in the last equality we applied importance sampling to rewrite the expectation w.r.t. π_{θ_k}. Finally, taking the gradient on both sides gives us:

$$\nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi^*\right)[s] = \nabla_\theta D_{\mathrm{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s] - \frac{1}{\mu} \mathop{\mathbb{E}}_{a \sim \pi_{\theta_k}(\cdot|s)} \left[\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a) \right) \right].$$

□

Note that (6.22) can be estimated by sampling from the trajectories generated by policy π_{θ_k} which allows us to train our policy using stochastic gradients.

Corollary 6.3 provides an outline for our algorithm. At every iteration we begin with a policy π_{θ_k}, which we use to run trajectories and gather data. We use that data and (6.17) to first estimate μ and ν. We then draw a minibatch from the data to estimate $\nabla_\theta \mathcal{L}(\theta)$ given in Corollary 6.3. After taking a gradient step using Equation (6.22), we draw another minibatch and repeat the process.

6.5.3 Practical Implementation

Solving the dual problem (6.17) is computationally impractical for large state/action spaces as it requires calculating the partition function $\mathcal{Z}_{\mu,\nu}(s)$ which often involves evaluating a high-dimensional integral or sum. Furthermore, μ and ν depends on k and should be adapted at every iteration.

We note that as $\mu \to 0$, π^* approaches a greedy policy; as μ increases, the policy becomes more exploratory. We also note that μ is similar to the temperature term used in maximum entropy reinforcement learning [Ziebart et al. 2008], which has been shown to produce reasonable results when kept fixed during training [Schulman et al. 2017a; Haarnoja et al. 2018]. In practice, we found that a fixed μ found through hyperparameter sweeps provides good results. However ν needs to be continuously adapted during training so as to ensure cost constraint satisfaction. Here we appeal to an intuitive heuristic for determining ν based on primal-dual gradient methods [Bertsekas 2014]. Recall that by strong duality, the optimal μ^* and ν^* minimizes the dual function (6.17) which we will denote by $L(\pi^*, \mu, \nu)$. We can therefore apply gradient descent w.r.t. ν to minimize $L(\pi^*, \mu, \nu)$. We can show that

Corollary 6.4. *The derivative of $L(\pi^*, \mu, \nu)$ w.r.t. ν is*

$$\frac{\partial L(\pi^*, \mu, \nu)}{\partial \nu} = \tilde{b} - \mathop{\mathbb{E}}_{\substack{s \sim d_{\pi_{\theta_k}} \\ a \sim \pi^*}} [A_{c,\gamma}^{\pi_{\theta_k}}(s, a)] \tag{6.24}$$

Proof. From Theorem 6.2, we have

$$L(\pi^*, \mu, \nu) = \mu\delta + \nu\tilde{b} + \mu \mathop{\mathbb{E}}_{s\sim d_{\pi_{\theta_k}}} [\log \mathcal{Z}_{\mu,\nu}(s)]. \tag{6.25}$$

The first two terms is an affine function w.r.t. ν, therefore its derivative is $\tilde{b}$. We will then focus on the expectation in the last term.

The partial derivative of $\mathcal{Z}$ w.r.t. ν is,

$$\begin{aligned}
\frac{\partial \mathcal{Z}_{\mu,\nu}(s)}{\partial \nu} &= \frac{\partial}{\partial \nu} \sum_a \pi_{\theta_k}(a|s) \exp\left(\frac{1}{\mu}\left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right)\right) \\
&= \sum_a -\pi_{\theta_k}(a|s) \frac{A_C^{\pi_{\theta_k}}(s,a)}{\mu} \exp\left(\frac{1}{\mu}\left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right)\right) \\
&= \sum_a -\frac{A_C^{\pi_{\theta_k}}(s,a)}{\mu} \frac{\pi_{\theta_k}(a|s)}{Z_{\mu,\nu}(s)} \exp\left(\frac{1}{\mu}\left(A_\gamma^{\pi_{\theta_k}}(s,a) - \nu A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right)\right) \mathcal{Z}_{\mu,\nu}(s) \\
&= -\frac{\mathcal{Z}_{\mu,\nu}(s)}{\mu} \mathop{\mathbb{E}}_{a\sim\pi^*(\cdot|s)} \left[A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right].
\end{aligned} \tag{6.26}$$

Therefore,

$$\frac{\partial \log \mathcal{Z}_{\mu,\nu}(s)}{\partial \nu} = \frac{\partial \mathcal{Z}_{\mu,\nu}(s)}{\partial \nu} \frac{1}{\mathcal{Z}_{\mu,\nu}(s)} = -\frac{1}{\mu} \mathop{\mathbb{E}}_{a\sim\pi^*(\cdot|s)} \left[A_{c,\gamma}^{\pi_{\theta_k}}(s,a)\right]. \tag{6.27}$$

Plugging the above derivative into the expectation and combining (**??**) with the derivatives of the affine term gives us the final desired result. □

The last term in the gradient expression in Equation (6.24) cannot be evaluated since we do not have access to π^*. However since π_{θ_k} and π^* are 'close', it is reasonable to assume that $E_{s\sim d_{\pi_{\theta_k}}, a\sim\pi^*}[A^{\pi_{\theta_k}}(s,a)] \approx E_{s\sim d_{\pi_{\theta_k}}, a\sim\pi_{\theta_k}}[A^{\pi_{\theta_k}}(s,a)] = 0$. In practice we find that this term can be set to zero which gives the update term:

$$\nu \leftarrow \mathop{\text{proj}}_{\nu} \left[\nu - \alpha(b - \rho_{c,\gamma}(\pi_{\theta_k}))\right] \tag{6.28}$$

where α is the step size, here we have incorporated the discount term $(1-\gamma)$ in $\tilde{b}$ into the step size. The projection operator proj_ν projects ν back into the interval $[0, \nu_{\max}]$ where $\nu_{\max}$ is chosen so that ν does not become too large. However we will show in later sections that FOCOPS is generally insensitive to the choice of $\nu_{\max}$ and setting $\nu_{\max} = +\infty$ does not appear to greatly reduce performance. Practically, $\rho_{c,\gamma}(\pi_{\theta_k})$ can be estimated via Monte Carlo methods using trajectories collected from π_{θ_k}. We note that the update rule in Equation (6.28) is similar in to the update rule introduced in Chow et al. [2017]. We recall that in (6.16), ν acts as a cost penalty term where increasing ν makes it less likely for state-action pairs with higher costs to be sampled by π^*. Hence in this regard, the update rule in (6.28) is intuitive in that it increases ν if $\rho_{c,\gamma}(\pi_{\theta_k}) > b$ (i.e. the cost constraint is violated for π_{θ_k}) and decreases ν otherwise. Using the update rule (6.28), we can then perform one update step on ν before updating the policy parameters θ.

Our method is a first-order method, so the approximations that we make is only accurate near the initial condition (i.e. $\pi_\theta = \pi_{\theta_k}$). In order to better enforce this we also add to (6.22) a per-state acceptance indicator function $I(s_j) := \mathbf{1}_{D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s_j] \leqslant \delta}$. This way sampled states whose $D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s]$ is too large are rejected from the gradient update. The resulting sample gradient update term is

$$\hat{\nabla}_\theta \mathcal{L}(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \left[\nabla_\theta D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s_j] - \frac{1}{\mu} \frac{\nabla_\theta \pi_\theta(a_j|s_j)}{\pi_{\theta_k}(a_j|s_j)} \left(\hat{A}_\gamma(s_j, a_j) - \nu \hat{A}_{c,\gamma}(s_j, a_j) \right) \right] I(s_j). \quad (6.29)$$

Here N is the number of samples we collected using policy π_{θ_k}, $\hat{A}$ and $\hat{A}_C$ are estimates of the advantage functions (for the return and cost) obtained from critic networks. We estimate the advantage functions using the Generalized Advantage Estimator (GAE) [Schulman et al. 2016]. We can then apply stochastic gradient descent using Equation (6.29). During training, we use the early stopping criteria $\frac{1}{N} \sum_{i=1}^{N} D_{\text{KL}}(\pi_\theta \| \pi_{\theta_k})[s_i] > \delta$ which helps prevent trust region constraint violation for the new updated policy. We update the parameters for the value net by minimizing the Mean Square Error (MSE) of the value net output and some target value (which can be estimated

via Monte Carlo or bootstrap estimates of the return). We emphasize again that FOCOPS only requires first order methods (gradient descent) and is thus extremely simple to implement.

Algorithm 4 presents a summary of the FOCOPS algorithm.

Algorithm 4 FOCOPS Outline

Initialize: Policy network π_{θ_0}, Value networks V_{ϕ_0}, $V^C_{\psi_0}$.

while Stopping criteria not met **do**
  Generate trajectories $\tau \sim \pi_{\theta_k}$.
  Estimate C-returns and advantage functions.
  Update ν using Equation (6.28).
  for K epochs **do**
    for each minibatch **do**
      Update value networks by minimizing MSE of V_{ϕ_k}, $V^{\text{target}}_{\phi_k}$ and $V^C_{\psi_k}$, $V^{C,\text{target}}_{\psi_k}$.
      Update policy network using Equation (6.29)
    if $\frac{1}{N}\sum_{j=1}^{N} D_{\text{KL}}\left(\pi_\theta \| \pi_{\theta_k}\right)[s_j] > \delta$ **then**
      Break out of inner loop

6.6 Experiments

We designed two different sets of experiments to test the efficacy of the FOCOPS algorithm. In the first set of experiments, we train different robotic agents to move along a straight line or a two dimensional plane, but the speed of the robot is constrained for safety purposes. The second set of experiments is inspired by the Circle experiments from Achiam et al. [2017]. Both sets of experiments are implemented using the OpenAI Gym API [Brockman et al. 2016] for the MuJoCo physical simulator [Todorov et al. 2012].

In addition to the CPO algorithm, we are also including for comparison two algorithms based on Lagrangian methods [Bertsekas 1997], which uses adaptive penalty coefficients to enforce constraints. For an objective function $f(\theta)$ and constraint $g(\theta) \leqslant 0$, the Lagrangian method solves max-min optimization problem $\max_\theta \min_{\nu \geqslant 0}(f(\theta) - \nu g(\theta))$. These methods first perform gradient ascent on θ, and then gradient descent on ν. Chow et al. [2019] and Ray et al. [2019] combined

Lagrangian method with PPO [Schulman et al. 2017b] and TRPO [Schulman et al. 2015] to form the PPO Lagrangian and TRPO Lagrangian algorithms, which we will subsequently abbreviate as PPO-L and TRPO-L respectively.

6.6.1 Robots with Speed Limit

We consider six MuJoCo environments where we attempt to train a robotic agent to walk. However we impose a speed limit on our environments. The cost thresholds are calculated using 50% of the speed attained by an unconstrained PPO agent after training for a million samples. For agents manuvering on a two-dimensional plane, the cost is calculated as

$$c(s, a) = \sqrt{v_x^2 + v_y^2}$$

For agents moving along a straight line, the cost is calculated as

$$c(s, a) = |v_x|$$

where v_x, v_y are the velocities of the agent in the x and y directions respectively.

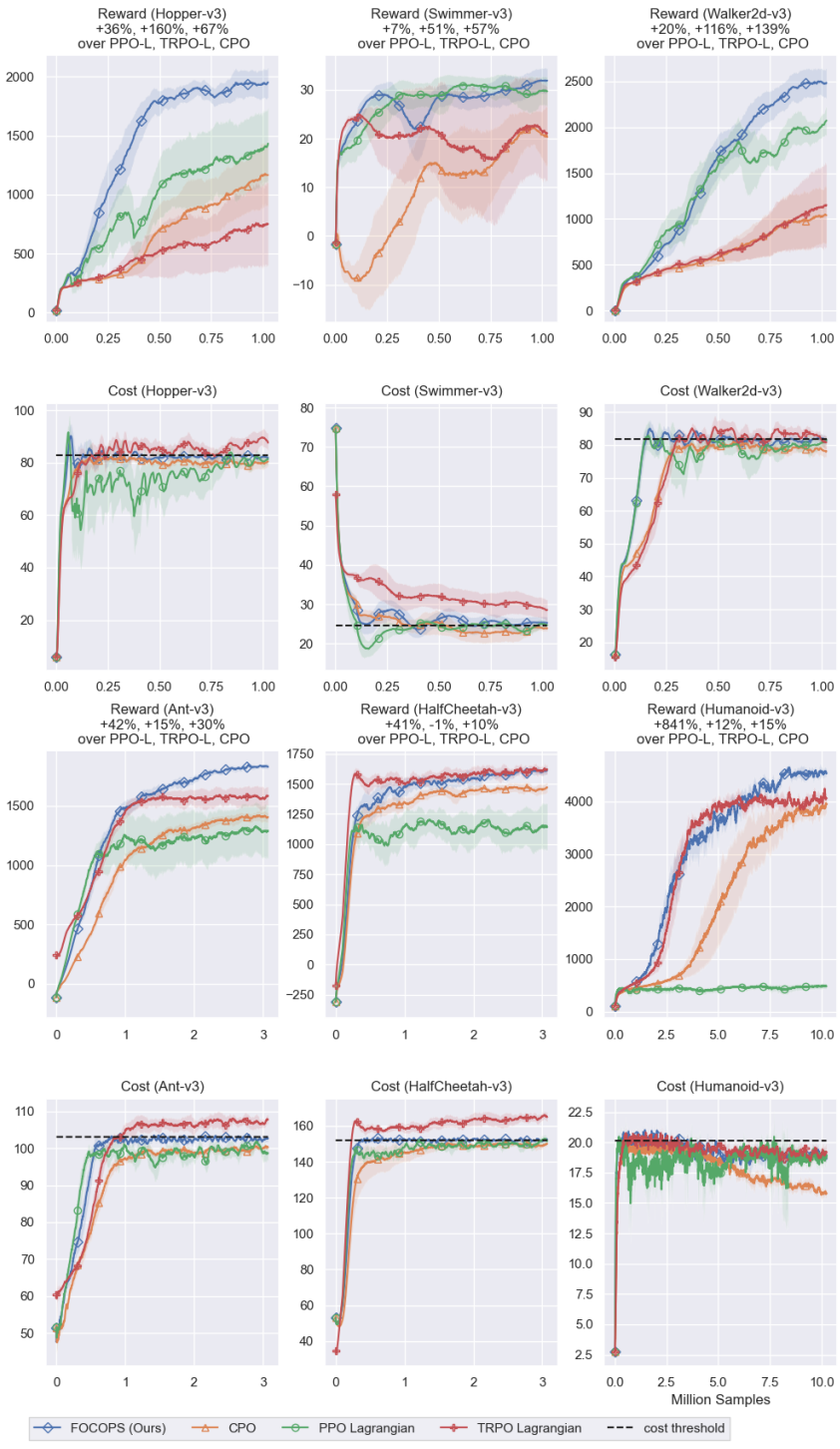


Figure 6.1: Learning curves for robots with speed limit tasks. The x-axis represent the number of samples used and the y-axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The shaded regions represent the bootstrap normal 95% confidence interval. FOCOPS consistently enforce approximate constraint satisfaction while having a higher performance on five out of the six tasks.

Figure 6.1 shows that FOCOPS outperforms other baselines in terms of reward on most tasks while enforcing the cost constraint. In theory, FOCOPS assumes that the initial policy is feasible. This assumption is violated in the Swimmer-v3 environment. However in practice, the gradient update term increases the dual variable associated with the cost when the cost constraint is violated, this would result in a feasible policy after a certain number of iterations. We observed that this is indeed the case with the swimmer environment (and similarly the AntCircle environment in the next section). Note also that Lagrangian methods outperform CPO on several environments in terms of reward, this is consistent with the observation made by Ray et al. [2019] and Stooke et al. [2020]. However on most tasks TRPO-L does not appear to consistently maintain constraint satisfaction during training. For example on HalfCheetah-v3, even though TRPO-L outperforms FOCOPS in terms of total return, it violates the cost constraint by nearly 9%. PPO-L is shown to do well on simpler tasks but performance deteriorates drastically on the more challenging environments (Ant-v3, HalfCheetah-v3, and Humanoid-v3), this is in contrast to FOCOPS which perform particularly well on these set of tasks. In Table 6.1 we summarized the performance of all four algorithms.

6.6.2 Circle Tasks

For these tasks, we use the same exact geometric setting, reward, and cost constraint function as Achiam et al. [2017].The goal of the agents is to move along the circumference of a circle while remaining within a safe region smaller than the radius of the circle. The exact geometry of the task is shown in Figure 6.2. The reward and cost functions are defined as:

$$r(s) = \frac{-yv_x + xv_y}{1 + |\sqrt{x^2 + y^2} - r|}$$
$$c(s) = \mathbf{1}(|x| > x_{\text{lim}}).$$

Table 6.1: Bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples over 10 random seeds of reward/cost return after training on robot with speed limit environments. Cost thresholds are in brackets under the environment names.

Environment		PPO-L	TRPO-L	CPO	**FOCOPS**
Ant-v3	Reward	1291.4 ± 216.4	1585.7 ± 77.5	1406.0 ± 46.6	$\mathbf{1830.0 \pm 22.6}$
(103.12)	Cost	98.78 ± 1.77	107.82 ± 1.16	100.25 ± 0.67	102.75 ± 1.08
HalfCheetah-v3	Reward	1141.3 ± 192.4	$\mathbf{1621.59 \pm 39.4}$	1470.8 ± 40.0	1612.2 ± 25.9
(151.99)	Cost	151.53 ± 1.88	164.93 ± 2.43	150.05 ± 1.40	152.36 ± 1.55
Hopper-v3	Reward	1433.8 ± 313.3	750.3 ± 355.3	1167.1 ± 257.6	$\mathbf{1953.4 \pm 127.3}$
(82.75)	Cost	81.29 ± 2.34	87.57 ± 3.48	80.39 ± 1.39	81.84 ± 0.92
Humanoid-v3	Reward	471.3 ± 49.0	4062.4 ± 113.3	3952.7 ± 174.4	$\mathbf{4529.7 \pm 86.2}$
(20.14)	Cost	18.89 ± 0.77	19.23 ± 0.76	15.83 ± 0.41	18.63 ± 0.37
Swimmer-v3	Reward	29.73 ± 3.13	21.15 ± 9.56	20.31 ± 6.01	$\mathbf{31.94 \pm 2.60}$
(24.52)	Cost	24.72 ± 0.85	28.57 ± 2.68	23.88 ± 0.64	25.29 ± 1.49
Walker2d-v3	Reward	2074.4 ± 155.7	1153.1 ± 473.3	1040.0 ± 303.3	$\mathbf{2485.9 \pm 158.3}$
(81.89)	Cost	81.7 ± 1.14	80.79 ± 2.13	78.12 ± 1.78	81.27 ± 1.33

where x, y are the positions of the agent on the plane, v_x, v_y are the velocities of the agent along the x and y directions, r is the radius of the circle, and x_{lim} specifies the range of the safety region. The radius is set to $r = 10$ for both Ant and Humanoid while x_{lim} is set to 3 and 2.5 for Ant and Humanoid respectively.

Similar to the previous tasks, we provide learning curves (Figure 6.3) and numerical summaries (Table 6.2) of the experiments. We also plotted an unconstrained PPO agent for comparison. On these tasks, all four approaches are able to approximately enforce cost constraint satisfaction (set at 50), but FOCOPS does so while having a higher performance. Note for both tasks, the 95% confidence interval for FOCOPS lies above the confidence intervals for all other algorithms, this is strong indication that FOCOPS outperforms the other three algorithms on these particular tasks.

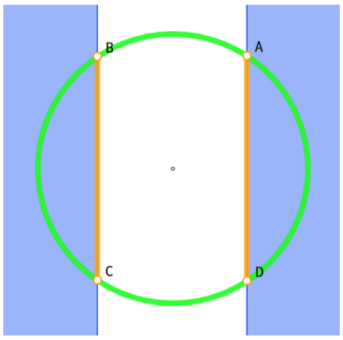


Figure 6.2: In the Circle task, reward is maximized by moving along the green circle. The agent is not allowed to enter the blue regions, so its optimal constrained path follows the line segments AD and BC (figure and caption taken from [Achiam et al. 2017]).

Table 6.2: Bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples over 10 random seeds of reward/cost return after training on circle environments for 10 million samples. Cost thresholds are in brackets under the environment names.

Environment		PPO-L	TRPO-L	CPO	**FOCOPS**
Ant-Circle	Reward	637.4 ± 88.2	416.7 ± 42.1	390.9 ± 43.9	$\mathbf{965.9 \pm 46.2}$
(50.0)	Cost	50.4 ± 4.4	50.4 ± 3.9	50.0 ± 3.5	49.9 ± 2.2
Humanoid-Circle	Reward	1024.5 ± 23.4	697.5 ± 14.0	671.0 ± 12.5	$\mathbf{1106.1 \pm 32.2}$
(50.0)	Cost	50.3 ± 0.59	49.6 ± 0.96	47.9 ± 1.5	49.9 ± 0.8

6.6.3 FOCOPS for Different Cost Thresholds

In this section, we verify that FOCOPS works effectively for different threshold levels. We experiment on the robots with speed limits environments. For each environment, we calculated the cost required for an unconstrained PPO agent after training for 1 million samples. We then used 25%, 50%, and 75% of this cost as our cost thresholds and trained FOCOPS on each of thresholds respectively. The learning curves are reported in Figure 6.4. We note from these plots that FOCOPS can effectively learn constraint-satisfying policies for different cost thresholds.

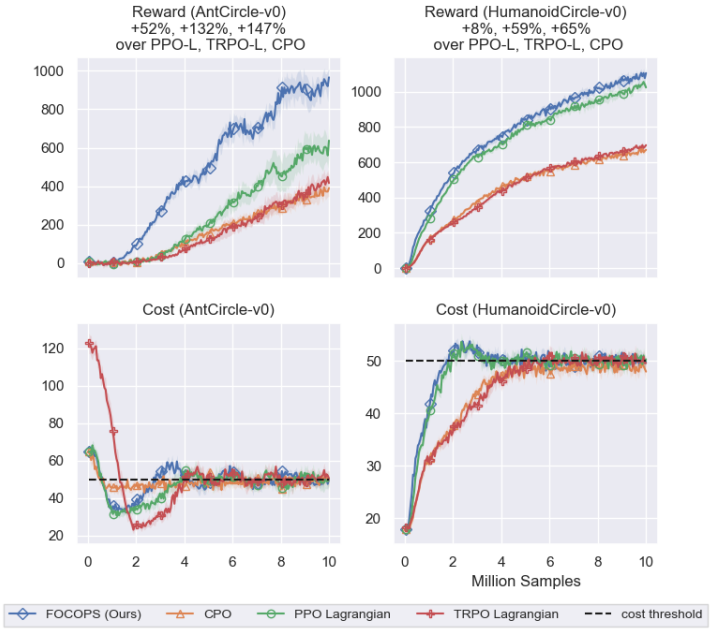


Figure 6.3: Comparing reward and cost returns on circle Tasks. The x-axis represent the number of samples used and the y-axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The shaded regions represent the bootstrap normal 95% confidence interval. An unconstrained PPO agent is also plotted for comparison.

6.6.4 Generalization Analysis

We used trained agents using all four algorithms (PPO Lagrangian, TRPO Lagrangian, CPO, and FOCOPS) on robots with speed limit tasks shown in Figure 6.1. For each algorithm, we picked the seed with the highest maximum return of the last 100 episodes which does not violate the cost constraint at the end of training. The reasoning here is that for a fair comparison, we wish to pick the best performing seed for each algorithm. We then ran 10 episodes using the trained agents on 10 unseen random seeds (identical seeds are used for all four algorithms) to test how well the algorithms generalize over unseen data. The final results of running the trained agents on the speed limit and circle tasks are reported in Tables 6.3. We note that on unseen seeds FOCOPS outperforms the other three algorithms on five out of six tasks.

Table 6.3: Average return of 10 episodes for trained agents on the robots with speed limit tasks on 10 unseen random seeds. Results shown are the bootstrap mean and normal 95% confidence interval with 1000 bootstrap samples.

Environment		PPO-L	TRPO-L	CPO	FOCOPS
Ant-v3	Reward	920.4 $\pm$ 75.9	1721.4 $\pm$ 191.2	1335.57 $\pm$ 43.17	**1934.9 $\pm$ 99.5**
(103.12)	Cost	68.25 $\pm$ 11.05	99.20 $\pm$ 2.55	80.72 $\pm$ 3.82	105.21 $\pm$ 5.91
HalfCheetah-v3	Reward	1698.0 $\pm$ 22.5	1922.4 $\pm$ 12.9	1805.5 $\pm$ 60.0	**2184.3 $\pm$ 32.6**
(151.99)	Cost	150.21 $\pm$ 4.47	179.82 $\pm$ 1.73	164.67 $\pm$ 9.43	158.39 $\pm$ 6.56
Hopper-v3	Reward	2084.9 $\pm$ 39.69	2108.8 $\pm$ 24.8	**2749.9 $\pm$ 47.0**	2446.2 $\pm$ 9.0
(82.75)	Cost	83.43 $\pm$ 0.41	82.17 $\pm$ 1.53	52.34 $\pm$ 1.95	81.26 $\pm$ 0.88
Humanoid-v3	Reward	582.2 $\pm$ 28.9	3819.3 $\pm$ 489.2	1814.8 $\pm$ 221.0	**4867.3 $\pm$ 350.8**
(20.14)	Cost	18.93 $\pm$ 0.93	18.60 $\pm$ 1.27	20.30 $\pm$ 1.81	21.58 $\pm$ 0.74
Swimmer-v3	Reward	37.90 $\pm$ 1.05	33.48 $\pm$ 0.44	33.45 $\pm$ 2.30	**39.37 $\pm$ 2.04**
(24.52)	Cost	25.49 $\pm$ 0.57	32.81 $\pm$ 2.61	22.61 $\pm$ 0.33	17.23 $\pm$ 1.64
Walker2d-v3	Reward	1668.7 $\pm$ 337.1	2638.9 $\pm$ 163.3	2141.7 $\pm$ 331.9	**3148.6 $\pm$ 60.5**
(81.89)	Cost	79.23 $\pm$ 1.24	90.96 $\pm$ 0.97	40.67 $\pm$ 6.86	73.35 $\pm$ 2.67

6.6.5 Sensitivity Analysis

We tested FOCOPS across ten different values of λ, and five difference values of $\nu_{\max}$ while keeping all other parameters fixed by running FOCOPS for 1 millon samples on each of the robots with speed limit experiment. For ease of comparison, we normalized the values by the return and cost of an unconstrained PPO agent trained for 1 million samples (i.e. if FOCOPS achieves a return of x and an unconstrained PPO agent achieves a result of y, the normalized result reported is x/y) The results on the robots with speed limit tasks are reported in Tables 6.4 and 6.5. We note that the more challenging environments such as Humanoid are more sensitive to parameter choices but overall FOCOPS is largely insensitive to hyperparameter choices (especially the choice of $\nu_{\max}$). We also presented the performance of PPO-L and TRPO-L for different values of $\nu_{\max}$.

Table 6.4: Performance of FOCOPS for Different λ

	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
λ	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
0.1	0.66	0.55	0.38	0.46	0.77	0.50	0.63	0.52	0.34	0.51	0.43	0.48	0.53	0.50
0.5	0.77	0.54	0.38	0.45	0.97	0.50	0.71	0.54	0.36	0.50	0.66	0.50	0.64	0.50
1.0	0.83	0.55	0.47	0.47	1.04	0.50	0.80	0.52	0.34	0.49	0.76	0.49	0.70	0.50
1.3	0.83	0.55	0.42	0.47	1.00	0.50	0.85	0.53	0.36	0.51	0.87	0.49	0.72	0.51
1.5	0.83	0.55	0.42	0.47	1.01	0.50	0.87	0.52	0.37	0.51	0.87	0.50	0.73	0.51
2.0	0.83	0.55	0.42	0.47	1.06	0.50	0.89	0.52	0.37	0.52	0.82	0.45	0.73	0.51
2.5	0.79	0.54	0.43	0.47	1.03	0.50	0.94	0.53	0.35	0.50	0.73	0.49	0.71	0.51
3.0	0.76	0.54	0.42	0.47	1.01	0.49	0.92	0.52	0.41	0.50	0.77	0.49	0.72	0.50
4.0	0.70	0.54	0.40	0.46	1.00	0.49	0.87	0.53	0.43	0.49	0.64	0.49	0.67	0.50
5.0	0.64	0.55	0.40	0.47	1.01	0.50	0.81	0.54	0.38	0.49	0.57	0.50	0.63	0.51

Table 6.5: Performance of FOCOPS for Different $\nu_{\max}$

	Ant-v3		HalfCheetah-v3		Hopper-v3		Humanoid-v3		Swimmer-v3		Walker2d-v3		All Environments	
$\nu_{\max}$	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost	Reward	Cost
1	0.83	0.55	0.45	0.61	1.00	0.51	0.87	0.52	0.40	0.62	0.88	0.50	0.74	0.55
2	0.83	0.55	0.42	0.47	1.01	0.50	0.87	0.52	0.35	0.51	0.87	0.50	0.73	0.51
3	0.81	0.54	0.41	0.47	1.01	0.49	0.83	0.53	0.34	0.49	0.87	0.50	0.71	0.50
5	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.31	0.49	0.87	0.50	0.71	0.51
10	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.34	0.47	0.87	0.50	0.71	0.50
$+\infty$	0.82	0.55	0.41	0.47	1.01	0.50	0.83	0.53	0.35	0.47	0.88	0.50	0.72	0.50

6.6.6 Implementation Details

All experiments were implemented in Pytorch 1.3.1 and Python 3.7.4 on Intel Xeon Gold 6230 processors. We used our own Pytorch implementation of CPO based on https://github.com/jachiam/cpo. For PPO, PPO Lagrangian, TRPO Lagrangian, we used an optimized PPO and TRPO implementation based on https://github.com/Khrylx/PyTorch-RL, https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, and https://github.com/ikostrikov/pytorch-trpo.

We used a two-layer feedforward neural network with a tanh activation for both our policy and value networks. We assume the policy is Gaussian with independent action dimensions. The policy networks outputs a mean vector and a vector containing the state-independent log standard deviations. States are normalized by the running mean the running standard deviation

before being fed to any network. The advantage values are normalized by the batch mean and batch standard deviation before being used for policy updates. Except for the learning rate for ν which is kept fixed, all other learning rates are linearly annealed to 0 over the course of training. Our hyperparameter choices are based on the default choices in the implementations cited at the beginning of the section. For FOCOPS, PPO Lagrangian, and TRPO Lagrangian, we tuned the value of $\nu_{\max}$ across $\{1, 2, 3, 5, 10, +\infty\}$ and used the best value for each algorithm. However we found all three algorithms are not especially sensitive to the choice of $\nu_{\max}$. Table 6.6 summarizes the hyperparameters used in our experiments.

6.7 Related Work

In the tabular case, CMDPs have been extensively studied for different constraint criteria [Kallenberg 1983; Beutler and Ross 1985, 1986; Ross 1989; Ross and Varadarajan 1989, 1991; Altman 1999].

In high-dimensional settings, Chow et al. [2017] proposed a primal-dual method which is shown to converge to policies satisfying cost constraints. Tessler et al. [2019] introduced a penalized reward formulation and used a multi-timescaled approach for training an actor-critic style algorithm which guarantees convergence to a fixed point. However multi-timescaled approaches impose stringent requirements on the learning rates which can be difficult to tune in practice. We note that neither of these methods are able to guarantee cost constraint satisfaction during training.

Several recent work leveraged advances in control theory to solve the CMDP problem. Chow et al. [2018, 2019] presented a method for constructing Lyapunov function which guarantees constraint-satisfaction during training. Stooke et al. [2020] combined PID control with Lagrangian methods which dampens cost oscillations resulting in reduced constraint violations.

Recently Yang et al. [2020] independently proposed the Projection-Based Constrained Policy

Optimization (PCPO) algorithm which utilized a different two-step approach. PCPO first finds the policy with the maximum return by doing one TRPO [Schulman et al. 2015] update. It then projects this policy back into the feasible cost constraint set in terms of the minimum KL divergence. While PCPO also satisfies theoretical guarantees for cost constraint satisfaction, it uses second-order approximations in both steps. FOCOPS is first-order which results in a much simpler algorithm in practice. Furthermore, empirical results from PCPO does not consistently outperform CPO.

The idea of first solving within the nonparametric space and then projecting back into the parameter space has a long history in machine learning and has recently been adopted by the RL community. Abdolmaleki et al. [2018] took the "inference view" of policy search and attempts to find the desired policy via the EM algorithm, whereas FOCOPS is motivated by the "optimization view" by directly solving the cost-constrained trust region problem using a primal-dual approach then projecting the solution back into the parametric policy space. Peters et al. [2010] and Montgomery and Levine [2016] similarly took an optimization view but are motivated by different optimization problems. However to the best of our knowledge, FOCOPS is the first algorithm to apply these ideas to cost-constrained RL.

6.8 Conclusion

In this chapter, we extended the theoretical guarantees for CPO from Achiam et al. [2017] to the average cost case. We then introduced FOCOPS—a simple first-order approach for training RL agents with safety constraints. FOCOPS is theoretically motivated and is shown to empirically outperform more complex second-order methods. FOCOPS is also easy to implement. We believe in the value of simplicity as it makes RL more accessible to researchers in other fields who wish to apply such methods in their own work. Our results indicate that constrained RL is an effective approach for addressing RL safety and can be efficiently solved using our two step approach.

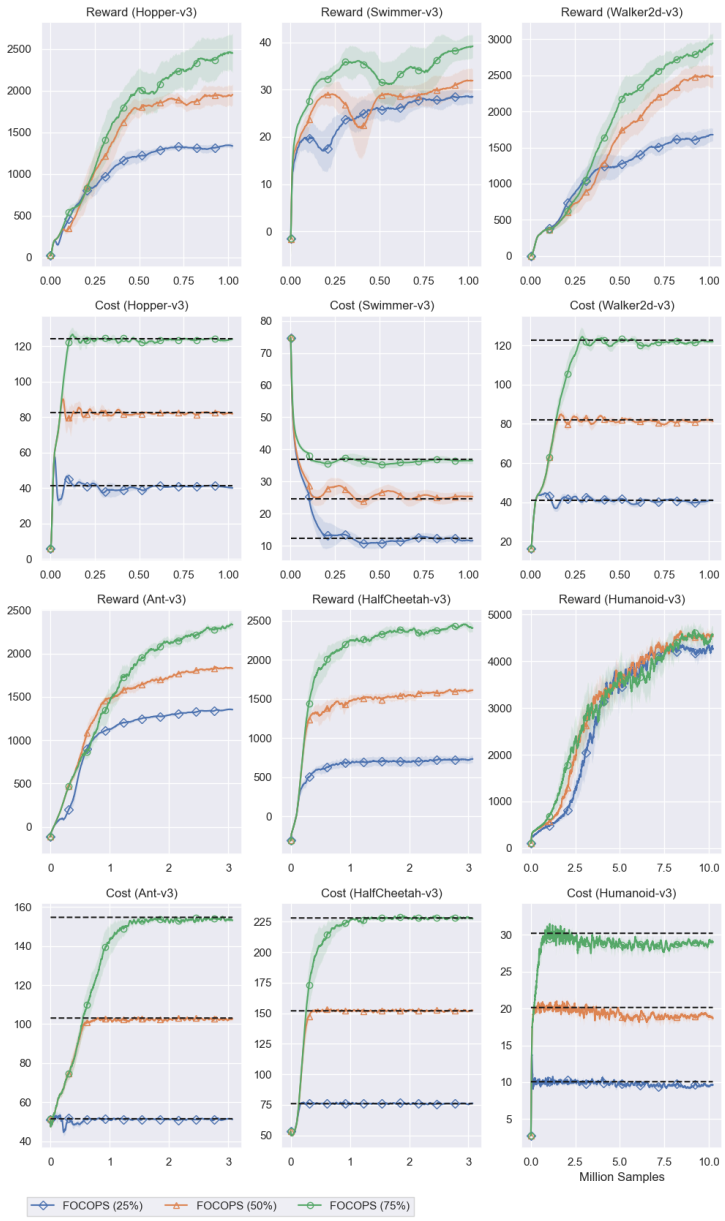


Figure 6.4: Performance of FOCOPS on robots with speed limit tasks with different cost thresholds. The x-axis represent the number of samples used and the y-axis represent the average total reward/cost return of the last 100 episodes. The solid line represent the mean of 1000 bootstrap samples over 10 random seeds. The horizontal lines in the cost plots represent the cost thresholds corresponding to 25%, 50%, and 75% of the cost required by an unconstrained PPO agent trained with 1 million samples. Each solid line represents FOCOPS trained with the corresponding thresholds. The shaded regions represent the bootstrap normal 95% confidence interval.

Table 6.6: Hyperparameters for robots with speed limit experiments

Hyperparameter	PPO	PPO-L	TRPO-L	CPO	FOCOPS
No. of hidden layers	2	2	2	2	2
No. of hidden nodes	64	64	64	64	64
Activation	tanh	tanh	tanh	tanh	tanh
Initial log std	-0.5	-0.5	-1	-0.5	-0.5
Discount for reward γ	0.99	0.99	0.99	0.99	0.99
Discount for cost γ_C	0.99	0.99	0.99	0.99	0.99
Batch size	2048	2048	2048	2048	2048
Minibatch size	64	64	N/A	N/A	64
No. of optimization epochs	10	10	N/A	N/A	10
Maximum episode length	1000	1000	1000	1000	1000
GAE parameter (reward)	0.95	0.95	0.95	0.95	0.95
GAE parameter (cost)	N/A	0.95	0.95	0.95	0.95
Learning rate for policy	3×10^{-4}	3×10^{-4}	N/A	N/A	3×10^{-4}
Learning rate for reward value net	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning rate for cost value net	N/A	3×10^{-4}	3×10^{-4}	3×10^{-4}	3×10^{-4}
Learning rate for ν	N/A	0.01	0.01	N/A	0.01
$L2$-regularization coeff. for value net	3×10^{-3}	3×10^{-3}	3×10^{-3}	3×10^{-3}	3×10^{-3}
Clipping coefficient	0.2	0.2	N/A	N/A	N/A
Damping coeff.	N/A	N/A	0.01	0.01	N/A
Backtracking coeff.	N/A	N/A	0.8	0.8	N/A
Max backtracking iterations	N/A	N/A	10	10	N/A
Max conjugate gradient iterations	N/A	N/A	10	10	N/A
Iterations for training value net[2]	1	1	80	80	1
Temperature λ	N/A	N/A	N/A	N/A	1.5
Trust region bound δ	N/A	N/A	0.01	0.01	0.02
Initial ν, $\nu_{\max}$	N/A	0, 1	0, 2	N/A	0, 2

7 | Conclusion and Future Directions

This thesis explored several algorithmic and theoretical issues around on-policy deep reinforcement learning. Sample efficiency is of central concern to our work and we explored this issue through the perspective of local policy search, average reward RL, and constrained RL.

In Chapters 3 and 4, we focused on the problem of average reward deep reinforcement learning. We introduced a new policy improvement bound for the average reward setting and demonstrated that an algorithm based on this bound can significantly outperform its discounted counterparts. Our work successfully addressed the mismatch between training and evaluation objectives which is common in modern DRL problems.

In Chapter 5 we revisited the original local policy search [Kakade and Langford 2002; Peters and Schaal 2008b] problem. We developed a unifying framework for policy updates involving a two-step process. Our approach is both easy to implement and could flexibly accommodate for a wide range of proximity measures.

Finally in Chapter 6, we considered the problem of RL problems with safety constraints. We examined both discounted and non-discounted cost constraints where we were able to extend the worst-case constraint violation guarantee from Achiam et al. [2017] to the non-discounted case. We then developed a simple first-order algorithm which balances both performance and safety.

One extension to our current work is to the *off-policy* setting. We recall that for on-policy algorithms, data is discarded after each update. Therefore an obvious approach to improve sample efficiency would be to attempt to re-use data generated by all or several previous policies. However,

due to the use of data points coming from several different distributions during the update phase, off-policy methods can be very unstable when combined with deep neural network function approximators [Sutton and Barto 2018].

For the average reward setting, We recall from Chapter 4 that critic estimation in the average reward value functions require knowledge of the gain term ρ. Estimating ρ in the on-policy setting is trivial and can be accomplished by a simple Monte Carlo estimate with trajectories collected using the current policy. However, this task becomes much more challenging in the off-policy case where data points come from different distributions. Recently Wan et al. [2020] introduced a convergent Q-learning-like algorithm for learning the optimal state-action bias function in the tabular setting which updates ρ in a temporal-differencing manner, and Zhang et al. [2021] proposed an algorithm for average reward off-policy evaluation with linear function approximators. However an effective off-policy DRL algorithm targeted at the average reward remains an open question.

Another potential research question is how to effectively use off-policy data for constrained RL problems. While in theory Lagrangian methods and the Lyapunov framework does allow extensions to the off-policy setting, constraint satisfaction is extremely challenging due to covariate shifts introduced by off-policy data [Chow et al. 2019]. Cost evaluation is an essential part of solving a constrained RL problem since an agent's decision process requires evaluating whether a policy satisfies the cost constraints. In the on-policy case, this can be done using either a simple Monte Carlo estimate or the surrogate function introduced in Chapter 6. However in the off-policy case, this requires evaluating a policy using data collected from other policies. The problem of Off-policy Policy Evaluation (OPE) [Voloshin et al. 2019] is itself a challenging problem where more research is needed.

Bibliography

Abbasi-Yadkori, Y., Bartlett, P., Bhatia, K., Lazic, N., Szepesvari, C., and Weisz, G. (2019). Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, pages 3692–3702.

Abdolmaleki, A., Springenberg, J. T., Tassa, Y., Munos, R., Heess, N., and Riedmiller, M. (2018). Maximum a posteriori policy optimisation. *International Conference on Learning Representation (ICLR)*.

Abounadi, J., Bertsekas, D., and Borkar, V. S. (2001). Learning algorithms for markov decision processes with average cost. *SIAM Journal on Control and Optimization*, 40(3):681–698.

Achiam, J. (2017). UC Berkeley CS 285 (Fall 2017), Advanced Policy Gradients. URL: http://rail.eecs.berkeley.edu/deeprlcourse-fa17/f17docs/lecture_13_advanced_pg.pdf. Last visited on 2020/05/24.

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2019). On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *arXiv preprint arXiv:1908.00261*.

Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2020). Optimality and approximation with

policy gradient methods in markov decision processes. In *Conference on Learning Theory*, pages 64–66. PMLR.

Altman, E. (1999). *Constrained Markov decision processes*, volume 7. CRC Press.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Amit, R., Meir, R., and Ciosek, K. (2020). Discount factor as a regularizer in reinforcement learning. In *International conference on machine learning*.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38.

Baxter, J. and Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350.

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334.

Bertsekas, D. P. (2014). *Constrained optimization and Lagrange multiplier methods*. Academic press.

Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic programming and optimal control*, volume 1,2. Athena scientific Belmont, MA.

Beutler, F. J. and Ross, K. W. (1985). Optimal policies for controlled markov chains with a constraint. *Journal of mathematical analysis and applications*, 112(1):236–252.

Beutler, F. J. and Ross, K. W. (1986). Time-average optimal constrained semi-markov decision processes. *Advances in Applied Probability*, 18(2):341–359.

Blackwell, D. (1962). Discrete dynamic programming. *The Annals of Mathematical Statistics*, pages 719–726.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877.

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Brémaud, P. (2020). *Markov Chains Gibbs Fields, Monte Carlo Simulation and Queues*. Springer, 2 edition.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Chen, X., Wang, C., Zhou, Z., and Ross, K. (2021). Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations (ICLR)*.

Cho, G. E. and Meyer, C. D. (2001). Comparison of perturbation bounds for the stationary distribution of a markov chain. *Linear Algebra and its Applications*, 335(1-3):137–150.

Chow, Y., Ghavamzadeh, M., Janson, L., and Pavone, M. (2017). Risk-constrained reinforcement learning with percentile risk criteria. *The Journal of Machine Learning Research*, 18(1):6070–6120.

Chow, Y., Nachum, O., Duenez-Guzman, E., and Ghavamzadeh, M. (2018). A lyapunov-based approach to safe reinforcement learning. In *Advances in neural information processing systems*, pages 8092–8101.

Chow, Y., Nachum, O., Faust, A., Ghavamzadeh, M., and Duenez-Guzman, E. (2019). Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031*.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. (2017). Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, pages 4299–4307.

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*.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. (2017). Openai baselines. https://github.com/openai/baselines.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2017). Rl^2: Fast reinforcement learning via slow reinforcement learning. In *International Conference on Learning Representations (ICLR)*.

Even-Dar, E., Kakade, S. M., and Mansour, Y. (2009). Online markov decision processes. *Mathematics of Operations Research*, 34(3):726–736.

Frobenius, G. (1912). über matrizen aus nicht negativen elementen. *Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften*, pages 456–477.

Furmston, T., Lever, G., and Barber, D. (2016). Approximate newton methods for policy search in markov decision processes. *Journal of Machine Learning Research*, 17.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

Gottesman, O., Johansson, F., Komorowski, M., Faisal, A., Sontag, D., Doshi-Velez, F., and Celi, L. A. (2019). Guidelines for reinforcement learning in healthcare. *Nature medicine*, 25(1):16–18.

Grinstead, C. M. and Snell, J. L. (2012). *Introduction to probability*. American Mathematical Soc.

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

Horn, R. A. and Johnson, C. R. (2012). *Matrix analysis*. Cambridge university press.

Howard, R. A. (1960). *Dynamic programming and markov processes*. John Wiley.

Hunter, J. J. (2005). Stationary distributions and mean first passage times of perturbed markov chains. *Linear Algebra and its Applications*, 410:217–243.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. (2015). The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189. Citeseer.

Jiang, N., Singh, S. P., and Tewari, A. (2016). On structural properties of mdps that bound loss due to shallow planning. In *IJCAI*, pages 1640–1647.

Kakade, S. (2001a). Optimizing average reward using discounted rewards. In *International Conference on Computational Learning Theory*, pages 605–615. Springer.

Kakade, S. and Langford, J. (2002). Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, volume 2, pages 267–274.

Kakade, S. M. (2001b). A natural policy gradient. *Advances in neural information processing systems*, 14.

Kakade, S. M. (2003). *On the sample complexity of reinforcement learning*. PhD thesis, University College London.

Kallenberg, L. (1983). *Linear Programming and Finite Markovian Control Problems*. Centrum Voor Wiskunde en Informatica.

Kemeny, J. and Snell, I. (1960). *Finite Markov Chains*. Van Nostrand, New Jersey.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., Lam, V.-D., Bewley, A., and Shah, A. (2019). Learning to drive in a day. In *International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*.

Komorowski, M., Celi, L. A., Badawi, O., Gordon, A. C., and Faisal, A. A. (2018). The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11):1716–1720.

Kraft, D. et al. (1988). A software package for sequential quadratic programming.

Lehmann, E. L. and Casella, G. (2006). *Theory of point estimation*. Springer Science & Business Media.

Lehnert, L., Laroche, R., and van Seijen, H. (2018). On value function representation of long horizon problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Levin, D. A. and Peres, Y. (2017). *Markov chains and mixing times*, volume 107. American Mathematical Soc.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

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. *International Conference on Learning Representations (ICLR)*.

Luketina, J., Nardelli, N., Farquhar, G., Foerster, J., Andreas, J., Grefenstette, E., Whiteson, S., and Rocktäschel, T. (2019). A survey of reinforcement learning informed by natural language. *arXiv preprint arXiv:1906.03926*.

Mahadevan, S. (1996). Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1-3):159–195.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. (2021). A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *NIPS Deep Learning Workshop*.

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.

Montgomery, W. H. and Levine, S. (2016). Guided policy search via approximate mirror descent. *Advances in Neural Information Processing Systems*, 29:4008–4016.

Naik, A., Shariff, R., Yasui, N., and Sutton, R. S. (2019). Discounted reinforcement learning is not an optimization problem. *NeurIPS Optimization Foundations for Reinforcement Learning Workshop*.

Neu, G., Antos, A., György, A., and Szepesvári, C. (2010). Online markov decision processes under bandit feedback. In *Advances in Neural Information Processing Systems*, pages 1804–1812.

Norvig, P. and Russell, S. (2002). *A modern approach*. Prentice Hall Upper Saddle River, NJ, USA:.

Nota, C. and Thomas, P. S. (2020). Is the policy gradient a gradient? In *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Perron, O. (1907). Zur theorie der matrices. *Mathematische Annalen*, 64(2):248–263.

Peters, J., Mulling, K., and Altun, Y. (2010). Relative entropy policy search. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*.

Peters, J. and Schaal, S. (2008a). Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190.

Peters, J. and Schaal, S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697.

Petrik, M. and Scherrer, B. (2008). Biasing approximate dynamic programming with a lower discount factor. In *Twenty-Second Annual Conference on Neural Information Processing Systems-NIPS 2008*.

Pirotta, M., Restelli, M., Pecorino, A., and Calandriello, D. (2013). Safe policy iteration. In *International Conference on Machine Learning*, pages 307–315.

Prasad, N., Cheng, L.-F., Chivers, C., Draugelis, M., and Engelhardt, B. E. (2017). A reinforcement learning approach to weaning of mechanical ventilation in intensive care units. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

Ray, A., Achiam, J., and Amodei, D. (2019). Benchmarking Safe Exploration in Deep Reinforcement Learning. *arXiv preprint arXiv:1910.01708*.

Ross, K. W. (1985). Constrained markov decision processes with queueing applications. *Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B- SCI. & ENG.]*,, 46(4).

Ross, K. W. (1989). Randomized and past-dependent policies for markov decision processes with multiple constraints. *Operations Research*, 37(3):474–477.

Ross, K. W. and Varadarajan, R. (1989). Markov decision processes with sample path constraints: the communicating case. *Operations Research*, 37(5):780–790.

Ross, K. W. and Varadarajan, R. (1991). Multichain markov decision processes with a sample path constraint: A decomposition approach. *Mathematics of Operations Research*, 16(1):195–207.

Rummery, G. A. and Niranjan, M. (1994). *On-line Q-learning using connectionist systems*, volume 37. Citeseer.

Scheel, O., Bergamini, L., Wołczyk, M., Osiński, B., and Ondruska, P. (2021). Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *International Conference on Robotics and Automation (ICRA)*. IEEE.

Schulman, J., Chen, X., and Abbeel, P. (2017a). Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the tenth international conference on machine learning*, volume 298, pages 298–305.

Seneta, E. (2006). *Non-negative matrices and Markov chains*. Springer Science & Business Media.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Song, H. F., Abdolmaleki, A., Springenberg, J. T., Clark, A., Soyer, H., Rae, J. W., Noury, S., Ahuja, A., Liu, S., Tirumala, D., et al. (2020). V-mpo: on-policy maximum a posteriori policy optimization for discrete and continuous control. *International Conference on Learning Representations*.

Stooke, A., Achiam, J., and Abbeel, P. (2020). Responsive safety in reinforcement learning by pid lagrangian methods. In *International Conference on Machine Learning*.

Strang, G. (2007). *Computational science and engineering*. Wellesley-Cambridge Press.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063.

Tadepalli, P. and Ok, D. (1994). H-learning: A reinforcement learning method to optimize undiscounted average reward. Technical Report 94-30-01, Oregon State University.

Tangkaratt, V., Abdolmaleki, A., and Sugiyama, M. (2018). Guide actor-critic for continuous control. In *International Conference on Learning Representations (ICLR)*.

Tessler, C., Mankowitz, D. J., and Mannor, S. (2019). Reward constrained policy optimization. *International Conference on Learning Representation (ICLR)*.

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*, pages 5026–5033. IEEE.

Tomar, M., Shani, L., Efroni, Y., and Ghavamzadeh, M. (2020). Mirror descent policy optimization. *arXiv preprint arXiv:2005.09814*.

Tsybakov, A. B. (2008). *Introduction to nonparametric estimation*. Springer Science & Business Media.

Veinott, A. F. (1966). On finding optimal policies in discrete dynamic programming with no discounting. *The Annals of Mathematical Statistics*, 37(5):1284–1294.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Voloshin, C., Le, H. M., Jiang, N., and Yue, Y. (2019). Empirical study of off-policy policy evaluation for reinforcement learning. *arXiv preprint arXiv:1911.06854*.

Vuong, Q., Zhang, Y., and Ross, K. W. (2019). Supervised policy update for deep reinforcement learning. In *International Conference on Learning Representation (ICLR)*.

Wan, Y., Naik, A., and Sutton, R. S. (2020). Learning and planning in average-reward markov decision processes. *arXiv preprint arXiv:2006.16318*.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., and de Freitas, N. (2017). Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations (ICLR)*.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, King's College, Cambridge University.

Wei, C.-Y., Jafarnia-Jahromi, M., Luo, H., Sharma, H., and Jain, R. (2020). Model-free reinforcement learning in infinite-horizon average-reward markov decision processes. In *International conference on machine learning*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in neural information processing systems (NIPS)*, pages 5285–5294.

Yang, S., Gao, Y., An, B., Wang, H., and Chen, X. (2016). Efficient average reward reinforcement learning using constant shifting values. In *AAAI*, pages 2258–2264.

Yang, T.-Y., Rosca, J., Narasimhan, K., and Ramadge, P. J. (2020). Projection-based constrained policy optimization. In *International Conference on Learning Representation (ICLR)*.

Zhang, S., Wan, Y., Sutton, R. S., and Whiteson, S. (2021). Average-reward off-policy policy evaluation with function approximation. *arXiv preprint arXiv:2101.02808*.

Zhang, Y. and Ross, K. (2021). On-policy deep reinforcement learning for the average reward criterion. In *International Conference on Machine Learning*.

Zhang, Y., Vuong, Q., and Ross, K. (2020). First order constrained optimization in policy space. In *Advances in Neural Information Processing Systems*, volume 33.

Zhao, T., Hachiya, H., Niu, G., and Sugiyama, M. (2011). Analysis and improvement of policy gradient estimation. In *Advances in Neural Information Processing Systems*, pages 262–270.

Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA.