Analysis of an Alternate Policy Gradient Estimator for Softmax Policies

by

Shivam Garg

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Policy gradient (PG) estimators are ineffective in dealing with softmax policies that are sub-optimally saturated, which refers to the situation when the policy concentrates its probability mass on sub-optimal actions. Sub-optimal policy saturation may arise from a bad policy initialization or a sudden change, i.e. a non-stationarity, in the environment that occurs after the policy has already converged. Unfortunately, current softmax PG estimators require a large number of updates to overcome policy saturation, which causes low sample efficiency and poor adaptability to new situations. To mitigate this problem, we propose a novel policy gradient estimator for softmax policies that utilizes the bias in the critic estimate and the noise present in the reward signal to escape the saturated regions of the policy parameter space. Our experiments, conducted on bandits and various reinforcement learning environments, and the resulting analysis show that this new estimator is more robust to policy saturation, and can be readily adapted to work with different PG algorithms and function approximation schemes.

Preface

This work was carried out in conjunction with Rupam Mahmood, Martha White, Samuele Tosatto, and Yangchen Pan. The project started when Rupam realized that writing the policy gradient (PG) in matrix form naturally lead to two different sample based estimators: the classic well known one and a new one which we call the *alternate estimator*. Initially, we tried using the alternate estimator to reduce the variance of the PG update. Although that approach mildly helped in the case of bandits, it did not seem fruitful for larger problems. Later, based on an empirical observation, we switched our focus on non-stationary problems, which got refined into the main focus of this work: the setting of *sub-optimally saturated softmax policies*. Significant portions of the thesis build upon the old submissions of this work (which focused on variance reduction and were rejected) to a conference and a workshop in 2020, and more recently an updated paper (focusing on sub-optimally saturated policies) that we have submitted to another conference in 2022.

For the sake of completeness and in the hope of increasing the readability of this thesis, I erred on the side of reproducing well known, possibly trivial, calculations throughout this work. Despite that, admittingly, the presentation still doesn't have enough background for an uninitiated reader and a lot of it will seem minor to the experienced ones; the correct balance remains elusive to me. Fortunately, the reader can substantially reduce the time spent reading this document by focusing on the following sections that contain the core ideas of this project: the alternate gradient estimator for bandits (§3.2), useful consequences of it being biased (§3.4), its extension to MDPs (§5.2), and the experiments on non-stationary problems (§7.2). Moreover, the reader can choose to skip the numerous footnotes in this thesis; they contain additional or clarifying remarks, and details that are often tangential to the main text. I should also mention that unfortunately, the figures are best viewed in **color** and on a zoomable screen.

The collaborative nature of this project necessitated the usage of "we" as the first person pronoun in writing of this thesis. However, I remain solely responsible for all the errors present, whether they be due to technical or presentation issues.

Shivam Garg

December, 2021

Acknowledgements

I am grateful to my amazing advisors Rupam and Martha for giving me the opportunity to work with them. I have learned a lot from them about research, teaching, and general life. Some of these are: having a scientific goal, how to conduct empirical analysis (creating and testing hypotheses, narrowing down the scope of the problem, running exploratory experiments), how to review well, how to write well, being brave, being disciplined, how to hold an argument, and seeking out help and helping others in whatever capacity you can. I am grateful to Csaba for giving me multiple opportunities from which I continue to learn tremendously, and also for serving on the examining committee. All three of them have been extremely kind and incredibly patient with me throughout.

I acknowledge that access to Compute Canada reduced the time taken to run the experiments for this thesis from months to days. And the university administration by allowing the students to return to the campus reduced the time for me to complete this project from years to months.

I am indebted to my friends, colleagues, and professors who have contributed in countless ways, some more than others, to making my time here memorable. I add a long but important list in α–β order: Abhishek, Adam, Aditya, Aidan, Alan, Alona, Alex L., Alex A., Amir K., Amir S., Ananya, Andy, Archit, Banafsheh, Bedir, Bingshan, Blanca, Cassidy, Chris, Connor, Daniel, Derek, Dhawal, Erfan, Eric, Farnaz, Farzane, Fatima, Fengdi, Gabor, Gautham, Greg, Hager, Hamza, Homayoon, Ivana, Janelle, Janey, Jiamin, Jincheng, Johannes, Junaid, Katja, Kenneth, Kenny, Khurram, Kiarash, Kirby, Kris, Liam, Mahtab, Maithrreye, Mark S., Martha S., Martin M., Matthew G., Matthew M., Matt S., Matt R., Mehran, Melanie, Michael P., Michael K., Mohamed E., Nelson, Nicolas, Niko, Nikoo, Osmar, Parash, Paritosh, Patrick, Per, Prabhat, Qingfeng, Raksha, Roshan, Rich, Robert, Russ, Sabina, Saeed, Samuele, Shahin, Shahrzad, Sham, Sharan, Shibhansh, Sheila, Shruthi, Sina, Soumyadeep, Steve, Sungsu, Tadashi, Tahsin, Tian, TK, U[1], Valentin, Valli, Vadim, Wesley, Xiaoqi, Xutong, Yan, Yangchen, Yi, Yixing, Yufeng, Zaheer, and Zahra.

Finally, I would like to thank my brother, my mother, and my father for always encouraging me in my endeavors. With their love, support, humor, and guidance, my life has been meaningful, stress free, and exceptionally fun!

[1] I am also grateful to **U**, you the reader, for taking out time to read this thesis.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Learning is a fundamental way humans improve the quality of the decisions they make, and by extension the quality of their lives. Individuals learn skills ranging from sensorimotor dexterity such as riding a bicycle to abstract mental abilities such as navigating an emotionally charged conversation. This learning happens continuously throughout their lives, and a good portion of it involves trying out various actions in different situations and identifying a good behavior by observing the resulting experience.

Deciding which actions to try is known as decision making. A salient feature of the decision making process is that it is sequential in nature: we need to perform a sequence of actions in order to accomplish any given task. For instance, in order to drink water, one may need to: find the water bottle, lift it up, open the cap, pour the water into your mouth, and so on. In order to accomplish most goals, we are similarly required to complete a series of actions. And a significant amount of human effort is spent in figuring out optimal decisions. From a more practical viewpoint, a variety of important business problems can also be modeled as sequential decision making tasks: autonomous driving, user targeted advertisement, inventory control, etc. Finding the optimal, or even acceptable, solutions to these problems is often difficult and time consuming, if not outright intractable. This creates the need for a general framework that can come up with good solutions for different tasks in a variety of different settings and with minimal human intervention.

Reinforcement learning (RL) is one such framework that aims to find good behavioral strategies in sequential decision making problems, by using the experience generated by the learner. This motivates the view that RL might be similar to the mechanisms employed by natural beings. By design, the performance of an RL system is not limited by that of a human expert. The only major input from a designer is to specify a goal, which is used to gauge the the goodness of a behavioral strategy. For instance, in the autonomous driving example, a possible goal could be to reach the destination in the shortest possible time while ensuring the safety of the passengers. In

such a problem. an RL system would begin with an initial driving behavior (either designed by the programmer or even a random one) and based on the resulting driving experience, continuously improve it. Ideally, it should also be able to improve its performance with time and adapt to changing circumstances.

To summarize, RL can be useful for achieving three separate objectives. First, for understanding natural intelligence by emulating it. If we build a self-improving agent which continually learns from its own experience using methods we fully understand, it would lend insights into the nature of the mechanisms employed by naturally intelligent beings (such as humans) themselves. Second, for pushing the boundaries of human knowledge by improving existing solutions[1]. And third, for building general purpose adaptive solutions to practical business problems[2].

In this work, instead of tackling the question of understanding natural intelligence or directly solving some practical business problem, we make small incremental progress in a particular class of RL algorithms known as policy gradient methods. Unfortunately, despite the promise of being able to adapt to changing circumstances (also known as non-stationarity), current RL methods, in particular policy gradient methods with a softmax policy, perform poorly with non-stationary problems. We explain this point further using the autonomous driving example: Consider an RL system trained to drive a car in a country with right-hand traffic. If this system were deployed in a country with left-hand traffic, ideally we would want it to quickly adapt to the new environment and learn the necessary skills. However, for this particular class of RL algorithms (policy gradient methods using a softmax policy) such an adaptation is not very efficient: the RL system is unable to unlearn its old behavior or acquire a new one in a reasonable amount of time. To help improve this situation, we propose and analyze a novel[3] policy gradient method, which improves the performance of RL based learners using softmax policies in particular kinds of non-stationary environments.

1.1 Problem Statement

Policy gradient (PG) algorithms aim to optimize a sequential decision making problem defined on a set of parameterized policies: the policy parameters are optimized using standard stochastic gradient-based update to maximize the net reward received by the agent. A salient feature of the PG methods is that they can be used with both discrete and continuous action spaces, and have

[1]For instance, the RL based algorithms for playing the games of Backgammon (Tesauro, 1995) and Go (Silver et al., 2017) discovered game playing strategies which were unbeknownst to the players before.

[2]People have also been using RL in conjunction with other approaches to come up with solutions to various industrial problems: such as controlling robots in warehouses (SORT robot, https://www.kindred.ai/blog/reinforcement-learning-beat-games-such-as-backgammon-and-go-and-is-paving-a-path-for-smarter-robots), or optimizing the performance of water treatment plants (Drayton Valley Water Treatment Project, https://edmontonjournal.com/news/local-news/pilot-project-tests-ai-to-improve-water-treatment).

[3]It was recently brought to our attention that the method proposed by us is not entirely novel and has been studied before in a different context; see Footnote 4 in §5.2.

been successfully deployed in a variety of real world tasks such as robotics (Haarnoja et al., 2018; Mahmood, et al., 2018) and large scale simulated problems (for instance, Berner et al., 2019).

When the action space is discrete, policies are typically represented using a categorical distribution parameterized by a softmax function. However, softmax policies have some inherent issues. Even with access to the true gradients, they can be slow in responding to non-stationarity (Hennes et al., 2019) and their performance heavily depends on the initialization of the policy parameters (Mei et al., 2020a; Mei et al., 2020b; Li et al., 2021). Both of these problems arise from an issue, which we call *sub-optimal policy saturation*.

Sub-optimal policy saturation refers to the situation when the policy places a high probability mass on sub-optimal actions. An agent with a saturated policy will not be able to explore other actions and may continue to remain in the sub-optimal region if an appropriate measure is not taken. Sub-optimal policy saturation is a significant issue that arises in multiple scenarios:

- **Non-stationarity:** when the agent is acting in a constantly changing environment, and what was once an optimal strategy may no longer work well.
- **Pre-training or transfer learning:** deep reinforcement learning systems are often pre-trained on different tasks before being used on the main task. The subtle differences in the structure of these tasks might lead to a policy which is good for the first task but is sub-optimally saturated for the main task.
- **Stochastic updates:** policy gradient agents are trained using sampled based updates which can have high stochasticity. Therefore, because of noisy updates, it is possible for a policy to become saturated on sub-optimal actions during the course of learning[4].

PG methods with softmax policies are particularly susceptible to the issue of saturation. Prior work has suggested different approaches to mitigate this issue: entropic regularization (Peters et al., 2010; Neu et al., 2017; Haarnoja et al., 2018; Ahmed et al., 2019), natural PG algorithms (Kakade, 2001; Peters and Schaal, 2008), and log-barrier regularization (Agarwal et al., 2021) can all make the policy more explorative. Explicit reward bonuses can also encourage the exploration of low-density state-action pairs (Auer et al. 2002). However, entropic regularization and explorative bonuses introduce additional terms in the objective, and therefore the resulting optimal policy can be different from the original one. Another solution is to replace the softmax with a better behaved function such as the escort transform (Mei et al., 2020b). We take a different approach to address this issue. Instead of augmenting the optimization objective with bonuses or regularizers, or replacing the softmax function entirely, we introduce a policy gradient estimator for softmax

[4]It is reasonable to assume that this noise should be useful for escaping such local optima. However, for softmax policies the noise signal is often too weak to overcome high policy saturation.

policies that inherently helps the agent escape the sub-optimally saturated regions; furthermore, the above augmentations can readily be applied on top of our estimator.

Our proposed estimator is a simple yet effective approach for dealing with sub-optimally saturated policies thereby making PG algorithms more robust. The classic likelihood ratio estimator for softmax policies, which we call as the *regular estimator*, takes a frustratingly large amount of samples to escape sub-optimally saturated policy regions. The regular estimator produces near zero gradients at saturation as both its expectation and variance, even with reward noise, are vanishingly small at those regions. In contrast, our proposed estimator, which we call the *alternate estimator*, has a non-zero variance in the same scenario that can be utilized to escape the sub-optimal regions. Further, the alternate estimator naturally utilizes the bias in the critic estimate to increase the policy's entropy, thereby encouraging exploration. As the critic estimate improves, this bias and consequently the effect of driving the policy towards a more uniform distribution reduces, allowing the agent to eventually achieve the optimal behavior.

1.2 Contributions

We present the alternate policy gradient estimator for softmax policies (for bandits in §3.2 and for MDPs in §5.2) and show that it works by utilizing reward noise (§3.3) as well as the bias in the critic (§3.4) to escape saturated regions in the policy space. We give theoretical justification for this behavior (previous two referenced sections) in the bandit setting complemented by simple and intuitive examples (Examples 1 and 2; §3.5). We provide an empirical verification of these properties on (1) problems with artificially saturated policies using both bandit (§4) and MDP (with tabular representation in §6 and with function approximation in §7) environments. And (2) on non-stationary tasks (for linear function approximation with tile-coding in §7.2 and for neural networks in §7.6) where sub-optimal policy saturation naturally arises. Our results show that this new estimator works well in both the bandit and the MDP settings with various PG algorithms such as REINFORCE (Williams, 1992) and online Actor-Critic (Konda and Tsitsiklis, 2000; Sutton et al., 2000), and is generally superior (or competitive) to the regular estimator under tabular, linear (with tile-coding), and neural representations.

Chapter 2

Background

In this chapter, we briefly describe the framework of Markov decision processes (MDPs) used by RL algorithms to model the decision making problem. We also describe a useful simplification of the MDP framework known as the bandit setting. We then discuss a particular class of RL solution methods, that we used in this work, known as the policy gradient methods, first for the simplified bandit setting and then for the full MDP setting. For the MDP framework, we focus on two popular policy gradient algorithms: REINFORCE and online Actor-Critic (AC). Finally, we give a brief description of the vector notation and some useful results on vector differentiation. To keep the presentation simple, we shun from using excessive mathematical rigor[1].

2.1 Problem Setting: Finite Markov Decision Processes

In RL, we typically model a sequential decision making problem using Markov decision processes (MDPs). The agent's surrounding is known as the environment and is formulated as an MDP $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mu, p, \gamma)$, where $\mathcal{S}$ is the discrete set of states, $\mathcal{A}$ is the discrete set of actions, $\mathcal{R} \subset \mathbb{R}$ is the discrete set of numerical rewards, $\mu \in \Delta(\mathcal{S})$ is the initial (start) state distribution, $p : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S} \times \mathcal{R})$ is the transition dynamics, and $\gamma \in [0, 1]$ is the discount factor. In these definitions, $\Delta(\mathcal{X})$ denotes the set of all possible probability distributions over the (discrete) set $\mathcal{X}$.

The agent interacts with the environment using a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ which governs its behavior. At the beginning of this interaction, the environment resets itself to a state $S_0 \sim \mu$. After this initial reset, at each timestep $t = 0, 1, 2, \ldots, T$, the agent picks an action $A_t \sim \pi(\cdot|S_t)$ which yields a reward R_t and the environment transitions to the next state S_{t+1}; both the next

[1]For instance, we will often use sums in places where integrals might have been more appropriate, freely use summations over infinite sets, inter-change infinite summations with expectations without any supporting argument, and define things in a rather intuitive manner. However, none of this compromises the veracity of any of the results.

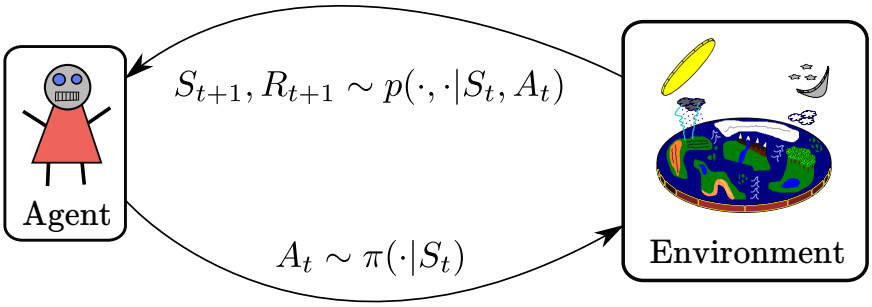

Figure 2.1: The interaction loop between an RL agent and an environment modeled using an MDP.

state and the next reward are sampled using the transition dynamics: $S_{t+1}, R_{t+1} \sim p(\cdot, \cdot|S_t, A_t)$. This interaction loop is illustrated in Figure 2.1. The cycle ends, when the environment reaches the terminal state $s_\times$, which may occur at different timesteps for different interactions. We denote the termination timestep by the random variable T which can be formally defined as $T := \min\{t \geq 0 : S_t = s_\times\}$. MDPs where the termination length T is guaranteed to be finite for any interaction are called episodic MDPs, and will be the focus of this work. In general, different episodes will terminate at different timesteps based on the problem dynamics. We use capital letters to denote random variables and small letters to denote the specific instances taken by those random variables. For instance, the expression $S_t = s$ implies that the random variable S_t, which represents the state variable that occurred at timestep t, took the specific state $s \in \mathcal{S}$. The sequence resulting from the interaction between the agent and the environment is known as an episode or a trajectory, and is represented by

$$S_0, A_0, R_1, S_1, A_1, \ldots, R_{T-1}, S_{T-1}, A_{T-1}, R_T, S_T.$$

The mechanics of this interaction gives rise to a probability distribution $\mathbb{P}_\pi$ over the set of all possible trajectories, which depends on agent's behavior and the environment's dynamics in the following way[2]: (1) $\mathbb{P}_\pi(S_0 = s_0) = \mu(s_0)$, (2) $\mathbb{P}_\pi(A_t = a_t|H_t = h_t) = \pi(a_t|s_t)$, and (3) $\mathbb{P}_\pi(S_{t+1} = s_{t+1}, R_{t+1} = r_{t+1}|H_t = h_t, A_t = a_t) = p(s_{t+1}, r_{t+1}|s_t, a_t)$ for all $0 \leq t \leq T-1$, where we denoted the sub-part of an episode using a sequence of random variables known as the history $H_t := (S_0, A_0, \ldots, R_t, S_t)$, and analogously $h_t := (s_0, a_0, \ldots, r_t, s_t)$. Further, we overload the notation $\mathbb{P}_\pi(H_t = h_t)$ to denote $\mathbb{P}_\pi(S_0 = s_0, A_0 = a_0, \ldots, R_t = r_1, S_t = s_t)$, and $\sum_{h_t} f(h_t)$ to denote $\sum_{s_0} \sum_{a_0} \cdots \sum_{r_t} \sum_{s_t} f(h_t)$, where f is some arbitrary function. When the context is clear, we will drop the term π from $\mathbb{P}_\pi$ and simply write $\mathbb{P}$ to reduce the clutter.

The goal of an RL agent is to find a policy that accumulates as much reward as possible. Specifically, the agent is interested in maximizing the sum of rewards known as the return, which

[2]The second condition specifies the policy π to be stationary (i.e. it is does not vary with the timestep t) and memoryless (i.e. it depends only on the last observed state s_t and not on the complete history h_t). This choice can be justified by the Markov property which is stated as the third condition. For a detailed treatment, see Chapter 38 of Lattimore and Szepesvári (2018) or https://rltheory.github.io/lecture-notes/planning-in-mdps/lec1/.

is defined as

$$G_t := \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1}. \tag{2.1}$$

In particular we are interested in maximizing G_0, the return over the complete episode. However, note that the return, apart from the trajectory dynamics, also depends on the uncertainty in the world—the exact same sequence of actions applied multiple times might still result in different returns—and therefore is not the most robust statistic to gauge the quality of the agent's behavior. An elegant way of dealing with this issue is to instead use a statistic that marginalizes out this uncertainty: the expected return accumulated by the agent when it follows a particular behavior policy π. Therefore, the goal of an RL agent is defined as a maximization problem of the quantity

$$\mathcal{J}_\pi := \mathbb{E}_\pi[G_0] = \mathbb{E}_\pi\left[R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T\right], \tag{2.2}$$

where the term $\mathbb{E}_\pi$ emphasizes that the expectation is taken over the episode sampled using the agent's policy π. Further, it is also useful to define the state value function $v_\pi : \mathcal{S} \to \mathbb{R}$ and the action-value function $q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ which measure the expected return obtained by the agent when it starts in a given state s or a state-action pair (s, a) respectively. These value functions are defined as

$$v_\pi(s) := \mathbb{E}_\pi\left[G_t | S_t = s\right] \qquad \text{and} \qquad q_\pi(s, a) := \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right]. \tag{2.3}$$

Specifying an Episodic Task using the Continuing Notation

In the upcoming sections, we will heavily focus on the term $\mathbb{E}_\pi[R_1 + \gamma R_2 + \cdots + \gamma^{T-1} R_T]$. However, this term can become messy to deal with since T itself is a random variable. Thus, given the simple structure of the variable G_0, we use a simple trick[3]: we reformulate (for alternate formulations, also see White, 2017) the episodic task as a continuing problem. Instead of considering the episode to have terminated at timestep T, we will assume that it continues forever. To maintain the equivalence with an episodic task, we add the constraint that upon entering the terminal state $s_\times$, the agent never leaves it and all the future rewards it receives are zero, i.e. $p(S_{t+1} = s_\times, R_{t+1} = 0 | S_t = s_\times, A_t = a) = 1$ for all timesteps $t \geq 0$ and any action $a \in \mathcal{A}$. With this formulation, the return G_t can equivalently be expressed as

$$G_t := \sum_{k=0}^{T-t-1} \gamma^k R_{t+k+1} = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2.4}$$

[3]There exist techniques for dealing with problems where the length of an episode is itself a random variable, which don't use such a reformulation. For instance, see *stopping time* in the stochastic processes literature; also see Wald's identity (`https://inst.eecs.berkeley.edu/~ee126/fa17/wald.pdf`).

since the rewards $R_{T+1}, R_{T+2}, \ldots$ are all zero. Further, with this change, the objective $\mathcal{J}_\pi$ can be written as

$$\mathcal{J}_\pi := \mathbb{E}_\pi[G_0] = \sum_{h_\infty} \mathbb{P}_\pi(H_\infty = h_\infty) \sum_{k=0}^{\infty} \gamma^k r_{k+1}, \tag{2.5}$$

where H_∞ is defined to be equal to the infinite length sequence $(S_0, A_0, R_1, S_1, A_1, \ldots)$ and analogously $h_\infty := (s_0, a_0, r_1, s_1, a_1, \ldots)$. Similarly, the value functions become:

$$v_\pi(s) := \mathbb{E}_\pi\left[G_t | S_t = s\right] = \sum_{h_\infty} \mathbb{P}_\pi(H_\infty = h_\infty | S_t = s) \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad \text{and} \tag{2.6}$$

$$q_\pi(s, a) := \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \sum_{h_\infty} \mathbb{P}_\pi(H_\infty = h_\infty | S_t = s, A_t = a) \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.7}$$

where $t \geq 0$ is an arbitrary timestep. Further, because of the reward structure,

$$v_\pi(s_\times) = 0, \qquad \text{and} \qquad q_\pi(s_\times, a) = 0, \tag{2.8}$$

for any policy π and action a. It is important to remember that the continuing formulation is just a mathematical convenience, and even though it introduces H_∞, we are still exclusively focusing on episodic tasks. In practice, whenever the agent will encounter the state $s_\times$, it will reset the episode by sampling $S_0 \sim \mu$. Also, we assume that the agent will not waste its resources on learning a policy for the terminal state $s_\times$; without loss of generality, we will simply define it to be a uniform distribution over the action space, i.e. $\pi(\cdot|s_\times) := \mathcal{U}(\mathcal{A})$.

2.2 The Bandit Problem Setting

The bandit setting is a highly useful and practical simplification of the MDP framework: it is used to describe non-sequential decision making problems and has no concept of states. A bandit problem can be completely specified using $(\mathcal{A}, \mathcal{R}, p)$, where $\mathcal{A}$ is the discrete action set, $\mathcal{R} \in \mathbb{R}$ is the discrete set of rewards, and $p : \mathcal{A} \to \Delta(\mathcal{R})$ is the probability of getting a certain reward given the action. It will also be useful to define the reward function $r : \mathcal{A} \to \mathbb{R}$ which gives the expected reward[4] given an action:

$$r(a) := \mathbb{E}[R|A = a] = \sum_{x \in \mathcal{R}} \mathbb{P}(R = x|A = a)x = \sum_{x \in \mathcal{R}} p(x|a)x. \tag{2.9}$$

To control its behavior, the agent maintains a policy $\pi \in \Delta(\mathcal{A})$. In a bandit task, at each

[4]Note that the symbol r refers to the reward function $r(\cdot)$ when describing bandits, and is different from the variable r_t used in the previous MDP section. It should be clear from context, whether we are referring to the reward function $r(\cdot)$ for bandits, or the reward at timestep t, i.e. r_t for the MDP setting.

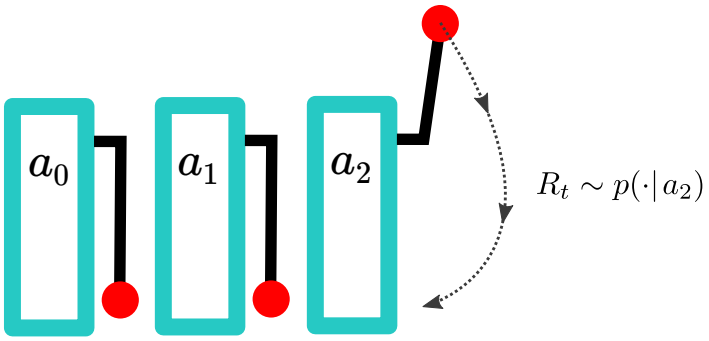

Figure 2.2: Multi-armed bandit setting with three actions: $\mathcal{A} = \{a_0, a_1, a_2\}$.

timestep t, the agent samples an action $A_t \in \mathcal{A}$ (an action is often called an arm in the bandit literature) from its policy, which results in a reward $R_t \sim p(\cdot|A_t)$. This is illustrated in Figure 2.2. The goal of the agent is to learn a policy π which maximizes the expected instantaneous reward

$$\mathcal{J}_\pi := \mathbb{E}_\pi[R] = \sum_a \pi(a) \sum_x p(x|a)x = \sum_a \pi(a) r(a), \tag{2.10}$$

where we overloaded the term $\mathcal{J}_\pi$ to represent both the expected instantaneous reward in bandits (Eq. 2.10) and the expected return in MDPs (Eq. 2.2); the exact meaning should again be clear from the context. Further, the value function equivalent in bandits is known as the average reward function $r_\pi := \mathbb{E}_\pi[R]$, which is incidentally equal to $\mathcal{J}_\pi$. The bandit problem focuses exclusively on the problem of *exploration-exploitation*, which is essentially equivalent to searching, in as little time as possible, for an arm that yields the highest expected reward.

2.3 Gradient Bandit Algorithm for Softmax Policies

Gradient bandit algorithms (Sutton and Barto, 2018) maintain a policy $\pi_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$, and the agent tunes these parameters to find a policy that maximizes the objective $\mathcal{J}_\pi$. The agent uses gradient based optimization techniques to tune these parameters, i.e. it first computes $\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi$ and then uses an update rule like gradient ascent to update the policy parameters: $\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \alpha \nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi$, where $\alpha \in \mathbb{R}^+$ is the stepsize parameter. Let us now calculate the gradient of this objective for the softmax policy parameterization, which is the focus of this thesis. A softmax policy is given by:

$$\pi_{\boldsymbol{\theta}}(a) = \frac{e^{\theta_a}}{\sum_{c \in \mathcal{A}} e^{\theta_c}}, \tag{2.11}$$

where θ_a is a learnable parameter and corresponds to the action preference for action a. When the context is clear, we will drop the $\boldsymbol{\theta}$ subscript from the policy and simply write π. Before we give the expression for the gradient of the objective, let us calculate the derivative of $\pi(a)$ with respect

to θ_b where a and b are any two, possibly identical, actions:

$$\frac{\partial}{\partial\theta_a}\pi(b) = \frac{\partial}{\partial\theta_a}\frac{e^{\theta_b}}{\sum_c e^{\theta_c}} = \frac{\mathbb{I}(a=b)e^{\theta_b}\sum_c e^{\theta_c} - e^{\theta_a}e^{\theta_b}}{\left(\sum_c e^{\theta_c}\right)^2} = \frac{e^{\theta_b}}{\sum_c e^{\theta_c}}\left[\mathbb{I}(a=b) - \frac{e^{\theta_a}}{\sum_c e^{\theta_c}}\right]$$
$$= \pi(b)\left[\mathbb{I}(a=b) - \pi(a)\right], \tag{2.12}$$

where $\mathbb{I}(a=b)$ is the indicator function that returns 1 when a and b are equal, and 0 otherwise. Now we can calculate the gradient of the objective $\mathcal{J}_\pi$ for the softmax policy with respect to the action preference θ_a for an arbitrary action a:

$$\frac{\partial}{\partial\theta_a}\mathcal{J}_\pi = \frac{\partial}{\partial\theta_a}\sum_{b\in\mathcal{A}}\pi(b)r(b) = \sum_{b\in\mathcal{A}} r(b)\frac{\partial}{\partial\theta_a}\pi(b) = \sum_{b\in\mathcal{A}} r(b)\pi(b)\left[\mathbb{I}(a=b) - \pi(a)\right] \tag{2.13}$$
$$= r(a)\pi(a)\left[1-\pi(a)\right] + \sum_{b\in\mathcal{A}\setminus\{a\}} r(b)\pi(b)\left[0-\pi(a)\right]$$
$$= r(a)\pi(a) - r(a)\pi(a)\pi(a) - \sum_{b\in\mathcal{A}\setminus\{a\}} r(b)\pi(b)\pi(a)$$
$$= \pi(a)r(a) - \pi(a)\sum_{b\in\mathcal{A}} r(b)\pi(b)$$
$$= \pi(a)\big[r(a) - r_\pi\big], \tag{2.14}$$

where in the first line we use the fact that $r(\cdot)$ is independent of the policy (and thereby the parameter θ_a) and the result from Eq. 2.12. With the above gradient expression, the parameters θ_as can be updated using gradient ascent $\theta_a \leftarrow \theta_a + \alpha\frac{\partial}{\partial\theta_a}\mathcal{J}_\pi$, leading to the update:

$$\theta_a \leftarrow \theta_a + \alpha\pi_{\boldsymbol{\theta}}(a)\big[r(a) - r_\pi\big] \qquad \text{for all } a\in\mathcal{A}, \tag{2.15}$$

where α is the stepsize parameter. The above update requires the knowledge of the expected reward function $r(\cdot)$ for all the possible actions. Unfortunately, such knowledge is seldom available to the agent, and therefore, the agent must resort to employing a sample based gradient estimator $\mathbf{g}(R,A)$ that satisfies[5]

$$\nabla_{\boldsymbol{\theta}}\mathcal{J}_\pi = \mathbb{E}_{A\sim\pi_{\boldsymbol{\theta}},R\sim p(\cdot|A)}[\mathbf{g}(R,A)]. \tag{2.16}$$

With access to such an estimator, the agent can simply sample an action A from its policy and obtain the corresponding reward R from the environment, which can then be used to compute $\mathbf{g}(R,A)$. The agent can then use stochastic gradient ascent to update its parameters. Let us

[5]We define the expectation of a vector random variable (or equivalently that of a vector function of random variables, as demonstrated in Eq. 2.16) in the standard way: for a d-dimensional random variable $\mathbf{X}$, $\mathbb{E}[\mathbf{X}] := \sum_{\mathbf{X}} p(\mathbf{X})\,\mathbf{X} = \sum_{x_1,x_2,\ldots,x_d} p(X_1=x_1, X_2=x_2,\ldots,X_d=x_d)\,\mathbf{X}$. Note that this definition is equivalent to taking elementwise expectation, i.e. $\mathbb{E}[\mathbf{X}] \equiv \begin{bmatrix}\mathbb{E}[X_1] & \mathbb{E}[X_2] & \cdots & \mathbb{E}[X_d]\end{bmatrix}^\top$. Using this definition, the expectation of a vector function (see §2.8) $\mathbf{f}(\cdot)$ operating on a (scalar or vector) random variable $\mathbf{Y}$, is given by $\mathbb{E}[\mathbf{f}(\mathbf{Y})] = \sum_{\mathbf{Y}} p(\mathbf{Y})\,\mathbf{f}(\mathbf{Y})$.

derive one such estimator, which matches the one given in Chapter 2 of Sutton and Barto (2018). Continuing from Eq. 2.14,

$$\begin{aligned}
\frac{\partial}{\partial \theta_a}\mathcal{J}_\pi &= \pi(a)\big[r(a) - r_\pi\big] = \pi(a)\left[r(a) - \sum_{b\in\mathcal{A}} r(b)\pi(b)\right] = \pi(a)r(a) - \sum_{b\in\mathcal{A}} \pi(a)r(b)\pi(b) \\
&= \sum_{b\in\mathcal{A}} \pi(b)r(b)\mathbb{I}(a=b) - \sum_{b\in\mathcal{A}} \pi(b)r(b)\pi(a) = \sum_{b\in\mathcal{A}} \pi(b)r(b)\Big(\mathbb{I}(a=b) - \pi(a)\Big) \\
&= \sum_{b\in\mathcal{A}} \pi(b) \sum_{x\in\mathcal{R}} p(x|b)x\Big(\mathbb{I}(a=b) - \pi(a)\Big) \\
&= \mathbb{E}_{A\sim\pi, R\sim p(\cdot|A)} \underbrace{\Big[R\Big(\mathbb{I}(A=a) - \pi(a)\Big)\Big]}_{=:g_a^{\text{No-b}}(R,A)}. \qquad (2.17)
\end{aligned}$$

It is possible to reduce the variance of the above estimator by subtracting an action independent baseline b from the reward while still maintaining its unbiasedness (Sutton et al., 2000). A popular choice of baseline is $b = r_\pi$; with this baseline, we get what we call as the regular gradient bandit estimator $\mathbf{g}^{\text{REG}}(R, A)$:

$$\frac{\partial}{\partial \theta_a}\mathcal{J}_\pi = \mathbb{E}_{A\sim\pi, R\sim p(\cdot|A)} \underbrace{\Big[\big(R - r_\pi\big)\Big(\mathbb{I}(A=a) - \pi(a)\Big)\Big]}_{=:g_a^{\text{REG}}(R,A)}. \qquad (2.18)$$

However, note that this estimator still requires access to r_π and thus the knowledge of the reward function r. The way to circumvent this issue is by estimating the average reward, say, by a running exponential average $\hat{r}_\pi$. Further, let our agent estimate the above expectation using a single sample. The resulting gradient bandits algorithm using the regular estimator is presented as Algorithm 1.

Algorithm 1 Gradient Bandit with Regular Estimator

Input a softmax policy $\pi_{\boldsymbol{\theta}}(a) := \frac{e^{\theta_a}}{\sum_{b\in\mathcal{A}} e^{\theta_b}}$ and an average reward function estimate $\hat{r}_\pi$
Input the policy stepsize α and the average reward stepsize β
Initialize $\boldsymbol{\theta}$ and $\hat{r}_\pi$
for total number of timesteps **do**
  Sample an action $A \sim \pi$ and a reward $R \sim p(\cdot|A)$
  for each action $a \in \mathcal{A}$ **do**
    `# Calculate the gradient estimator`
    $\hat{g}_a^{\text{REG}} = \big(R - \hat{r}_\pi\big)\Big(\mathbb{I}(A=a) - \pi(a)\Big)$
    `# Update the policy parameter`
    $\theta_a \leftarrow \theta_a + \alpha \cdot \hat{g}_a^{\text{REG}}$
  `# Update the average reward estimate`
  $\hat{r}_\pi \leftarrow \beta R + (1-\beta)\hat{r}_\pi$

2.4 Policy Gradient Methods

Policy gradient methods are the MDP analog of gradient bandits. The agent using them starts by considering a parameterized policy $\pi_{\mathbf{w}}$ with parameters $\mathbf{w}$. Then it estimates the gradient of the objective $\nabla_{\mathbf{w}} \mathcal{J}_\pi$ using data collected by interacting with the environment, and uses stochastic gradient ascent to update the parameters $\mathbf{w}$. Often, in order to obtain a good estimate of the gradient $\nabla_{\mathbf{w}} \mathcal{J}_\pi$, the agent also needs access to the value function v_π. Therefore, the agent might also need to maintain a value function estimate $\hat{v}_{\boldsymbol{\omega}}$ with parameters $\boldsymbol{\omega}$. In this work, we focus on algorithms that estimate the gradient, by using a particular form of policy gradient estimator known as the log-likelihood estimator (for an explanation of this terminology, see the last paragraph of §2.7). Other forms of estimators derived from, say, the re-parameterization policy gradient (Silver et al., 2014; Lan and Mahmood, 2021), and the related algorithms are out of the scope of this work.

The policy gradient can be estimated by a large number of policy gradient estimators (for instance, see Table 1, Huang and Jiang, 2020). These estimators have different behaviors and give rise to a wide range of algorithms that are suitable for different scenarios. Elegant, succinct, and widely varied derivations of various policy gradient theorems and the related estimators already exist in the literature (Williams 1992; Sutton et al., 2000; Konda and Tsitsiklis, 2000; Kakade, 2001; Degris et al., 2012a, Silver et al., 2014; Schulman et al., 2015; Ghosh et al., 2020; Huang and Jiang, 2020; Tang et al., 2020; Lan and Mahmood, 2021; Vaswani et al., 2021). Therefore, in this chapter we will consider some of the simplest estimators and focus on details that are often not mentioned explicitly. We will demonstrate how the policy gradient, based on small changes in the calculation, naturally can be expressed as multiple sample based estimators each of which has different properties and has to be estimated using a unique sampling process.

We start by stating a probability trick, that we will heavily employ in our calculations. The proof uses the marginalization of the joint probability distribution over discrete random variables.

Proposition 1. ***(Marginalization Trick):*** *Let $\mathcal{U} = \{X_0, X_1, \ldots, \}$ be a countably infinite set of random variables, where X_i takes values from the set $\mathcal{X}_i$ for all $i \geq 0$. Let $\mathbb{P}$ be a joint probability distribution over these random variables $X_1, X_2, \ldots$, and define the shorthand $\sum_{\mathcal{U}} \mathbb{P}(\mathcal{U}) := \sum_{x_1 \in \mathcal{X}_1} \sum_{x_2 \in \mathcal{X}_2} \cdots \mathbb{P}(X_1 = x_1, X_2 = x_2, \ldots)$. Also let $\mathcal{V} = \{Y_1, Y_2, \ldots, Y_k\} \subset \mathcal{U}$ for some finite k, and let $f(\cdot)$ be a function defined on the random variables in the set $\mathcal{V}$. Then*

$$\sum_{\mathcal{U}} \mathbb{P}(\mathcal{U}) f(\mathcal{V}) = \sum_{\mathcal{V}} \left(\sum_{\mathcal{U} \setminus \mathcal{V}} \mathbb{P}(\mathcal{U}) \right) f(\mathcal{V}) = \sum_{\mathcal{V}} \mathbb{P}(\mathcal{V}) f(\mathcal{V}). \tag{2.19}$$

In order to take the gradient of the objective of policy gradient algorithms, let us expand $\mathcal{J}_\pi$ to highlight its dependence on the probability of the trajectory $\mathbb{P}_\pi$ using which, immediately after,

we will show its dependence on the policy π:

$$\mathcal{J}_\pi = \sum_{h_\infty} \mathbb{P}(H_\infty = h_\infty) \sum_{t=0}^{\infty} \gamma^t r_{t+1} \tag{2.20}$$

$$= \sum_{t=0}^{\infty} \gamma^t \sum_{h_\infty} \mathbb{P}(H_\infty = h_\infty) \cdot r_{t+1} = \sum_{t=0}^{\infty} \gamma^t \sum_{r_{t+1}} \mathbb{P}(R_{t+1} = r_{t+1}) \cdot r_{t+1}, \tag{2.21}$$

where in the second line we used the marginalization trick with $\mathcal{U} = \{H_\infty : S_i \in \mathcal{S}, A_i \in \mathcal{A}, \text{ and } R_i \in \mathcal{R} \text{ for all } i \geq 0\}$ and $\mathcal{V} = \{R_{t+1}\}$. We will now separately take the gradients of Eqs. 2.20 and 2.21 to obtain different PG estimators. Before we do that, it is helpful to state some intermediate results:

$$\begin{aligned}
\nabla \mathbb{P}(H_\infty = h_\infty) &= \mathbb{P}(H_\infty = h_\infty) \frac{\nabla \mathbb{P}(H_\infty = h_\infty)}{\mathbb{P}(H_\infty = h_\infty)} = \mathbb{P}(H_\infty = h_\infty) \nabla \log \mathbb{P}(H_\infty = h_\infty) \\
&= \mathbb{P}(H_\infty = h_\infty) \left[\nabla \log \mu(s_0) + \nabla \sum_{k=0}^{\infty} \log \pi(a_k|s_k) + \nabla \sum_{k=0}^{\infty} \log p(s_{k+1}, r_{k+1}|s_k, a_k) \right] \\
&= \mathbb{P}(H_\infty = h_\infty) \sum_{k=0}^{\infty} \nabla \log \pi(a_k|s_k),
\end{aligned} \tag{2.22}$$

where the first and the last gradient terms in second line are zero since both the distributions μ and p are fixed independently of π. Analogously, we also obtain

$$\begin{aligned}
&\nabla \mathbb{P}(R_t = r_t) \\
&= \nabla \sum_{h_{t-1}, a_{t-1}} \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, R_t = r_t) \\
&= \sum_{h_{t-1}, a_{t-1}} \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, R_t = r_t) \nabla \log \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, R_t = r_t) \\
&= \sum_{h_{t-1}, a_{t-1}} \mathbb{P}(H_{t-1} = h_{t-1}, A_{t-1} = a_{t-1}, R_t = r_t) \sum_{k=0}^{t-1} \nabla \log \pi(a_k|s_k).
\end{aligned} \tag{2.23}$$

2.5 REINFORCE

REINFORCE (Williams, 1992) is a policy gradient algorithm that uses Monte-Carlo rollouts to estimate the policy gradient. We first calculate the gradient estimator used by REINFORCE and then present the complete algorithm. If we take the gradient of Eq. 2.20 with respect to the policy

parameters and use Eq. 2.22, we obtain

$$\begin{aligned}\nabla \mathcal{J}_\pi &= \sum_{h_\infty} \mathbb{P}(H_\infty = h_\infty) \left[\sum_{k=0}^{\infty} \nabla \log \pi(a_k|s_k)\right] \sum_{t=0}^{\infty} \gamma^t r_{t+1} \\ &= \mathbb{E}_{H_\infty \sim \mathbb{P}_\pi} \left[G_0 \sum_{k=0}^{\infty} \nabla \log \pi(A_k|S_k)\right], \end{aligned} \tag{2.24}$$

where the second line follows from the definition of the expectation under $\mathbb{P}_\pi$ and the definition of the return $G_0 := \sum_{t=0}^{\infty} \gamma^t R_{t+1}$. Interestingly, if we instead take the gradient of Eq. 2.21 and use Eq. 2.23, we can get a better PG estimator:

$$\nabla \mathcal{J}_\pi = \sum_{t=0}^{\infty} \gamma^t \sum_{h_t, a_t, r_{t+1}} \mathbb{P}(H_t = h_t, A_t = a_t, R_{t+1} = r_{t+1}) \left[\sum_{k=0}^{t} \nabla \log \pi(a_k|s_k)\right] r_{t+1} \tag{2.25}$$

$$\begin{aligned} &= \sum_{t=0}^{\infty} \gamma^t \sum_{h_\infty} \mathbb{P}(h_\infty = h_\infty) \left[\sum_{k=0}^{t} \nabla \log \pi(a_k|s_k)\right] r_{t+1} \\ &= \sum_{h_\infty} \mathbb{P}(H_\infty = h_\infty) \sum_{k=0}^{\infty} \gamma^k \nabla \log \pi(a_k|s_k) \left[\sum_{t=k}^{\infty} \gamma^{t-k} r_{t+1}\right] \\ &= \mathbb{E}_{H_\infty \sim \mathbb{P}_\pi} \left[\sum_{k=0}^{\infty} \gamma^k \nabla \log \pi(A_k|S_k) \left(\sum_{t=k}^{\infty} \gamma^{t-k} R_{t+1}\right)\right] \\ &= \mathbb{E}_{H_\infty \sim \mathbb{P}_\pi} \left[\sum_{k=0}^{\infty} \gamma^k \nabla \log \pi(A_k|S_k) G_k\right], \end{aligned} \tag{2.26}$$

where in the second line we used the marginalization trick, in the third line we rearranged the order of the summations, in the fourth line we used the definition of expectation under $\mathbb{P}_\pi$, and in the last line we used the definition of the return $G_k := \sum_{t=k}^{\infty} \gamma^{t-k} R_{t+1}$. Although both the above PG estimators (given in Eq. 2.24 and Eq. 2.26) are unbiased, the latter has a lower variance (see Table 1, Huang and Jiang, 2020) and is thus more desirable. We can further reduce the variance of the latter estimator by subtracting an action independent baseline from the return without introducing any bias (Williams, 1992; Sutton et al., 2000). One popular choice is the value function v_π (Williams, 1992; Sutton et al., 2000); other possible choices are baselines which optimally reduce the variance of the PG estimator (Greensmith et al., 2004; Weaver and Tao, 2013). This leads to the REINFORCE with baseline estimator $\mathbf{g}^{REINF}(H_\infty)$:

$$\nabla \mathcal{J}_\pi = \mathbb{E}_{H_\infty \sim \mathbb{P}_\pi} \underbrace{\left[\sum_{t=0}^{\infty} \gamma^t \nabla \log \pi(A_t|S_t) \Big(G_t - v_\pi(S_t)\Big)\right]}_{=: \mathbf{g}^{REINF}(H_\infty)}. \tag{2.27}$$

The REINFORCE algorithm uses the above PG estimator to update the policy weights. However, computing this estimator requires access to v_π and involves an infinite sum. To solve the first issue, the agent, instead of using the true value function, can use an estimate $\hat{v}_{\boldsymbol{\omega}}$, where $\boldsymbol{\omega}$ represents the learnable parameters. For the second issue, recall that we fixed $\pi(\cdot|s_\times) := \mathcal{U}(\mathcal{A})$, and consequently $\nabla\pi(A_t|S_t) = 0$ for all $t \geq T$. Therefore, the REINFORCE with baseline PG estimator can be approximated as: $\mathbb{E}_{H_\infty \sim \mathbb{P}_\pi}\left[\sum_{t=0}^{T(H_\infty)-1} \gamma^t \nabla \log \pi(A_t|S_t)\Big(G_t - \hat{v}_{\boldsymbol{\omega}}(s_t)\Big)\right]$, where $T(H_\infty)$ emphasizes the fact that given a trajectory H_∞, the agent can figure out the termination step. We now present the REINFORCE algorithm (Algorithm 2).

Algorithm 2 REINFORCE with Baseline

Input a policy $\pi_{\mathbf{w}}(a|s)$ and a value function estimate $\hat{v}_{\boldsymbol{\omega}}(s)$
Input the policy stepsize α and the value function stepsize β
Initialize $\mathbf{w}$ and $\boldsymbol{\omega}$
for total number of episodes **do**
  Sample an episode $(S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T) \sim \mathbb{P}_\pi$
  `# Calculate the PG estimate`
  $\hat{\mathbf{g}}^{\text{REINF}} = \sum_{t=0}^{T-1} \gamma^t (G_t - \hat{v}_{\boldsymbol{\omega}}(S_t)) \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(A_t|S_t)$
  `# Update the policy`
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha \cdot \hat{\mathbf{g}}^{\text{REINF}}$
  for each step of the episode $t = 0, 1, \ldots, T-1$ **do**
    Calculate the return $G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T$
    `# Update the value function estimate`
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \beta \cdot \Big(G_t - \hat{v}_{\boldsymbol{\omega}}(S_t)\Big) \nabla_{\boldsymbol{\omega}} \hat{v}_{\boldsymbol{\omega}}(S_t)$

REINFORCE is a true gradient based algorithm, and therefore should converge atleast to a local optimum under certain conditions on stepsizes (Bertsekas and Tsitsiklis, 2000). In the version given above, we made some additional design choices. First, the agent uses a single trajectory to compute the gradient estimator; using more trajectories to estimate the gradient should result in a higher quality estimate; but the agent will make fewer updates. Second, it uses data from the same trajectory with a squared Monte-Carlo loss (Sutton and Barto, 2018) to estimate the value function. Using the same trajectory for updating both the actor and the baseline should introduce some bias but allows the agent to reuse data; for a brief discussion and further references on this topic, see Wu et al. (2020). The choice of the baseline loss function also affects the learning dynamics. Our choice was motivated by convenience—value functions are easy to estimate and work well in practice; a principled way of deriving baseline loss functions is an open research question. And third design choice was to update the baseline after the policy[6].

Algorithms, like REINFORCE, which store the entire episode, and wait until the termination to

[6]This reduces the bias from correlation. For a small counterexample where updating the baseline before the policy can lead to zero gradients, see slide 25 by Martha White, Policy gradient tutorial, NeurIPS 2020 (https://drive.google.com/file/d/12w3EHzgE6CNaBhjPrqJO7IDY7WCnWBPW/view).

make the updates are called batch algorithms. In contrast, algorithms which don't store the episode and make an incremental update per-timestep are known as online algorithms. Online algorithms have the benefit of distributing the computation equally across the agent's timesteps and don't require extensive memory for storing episodes. We now present one such online algorithm.

2.6 Online Actor-Critic (AC)

The online AC algorithm (Chapter 13, Sutton and Barto, 2018), unlike REINFORCE, uses an estimate of the action-value function in place of the return, which enables it to update its policy at each timestep of the episode. Before calculating the online AC estimator, we discuss two relevant concepts: the expected termination length of an episode and the on-policy distribution of a policy.

The expected termination length under $\mathbb{P}_\pi$, denoted by $\mathbb{E}[T]$, is the average number of timesteps an episode will last when the agent follows the policy π in a given environment. It can be computed in terms of the probability distribution $\mathbb{P}$ as follows:

$$\mathbb{E}[T] = \sum_{k=0}^{\infty} \mathbb{P}(T > k) = \sum_{k=0}^{\infty} \mathbb{P}(S_k \neq s_\times) = \sum_{k=0}^{\infty} \sum_{s \in \mathcal{S} \setminus \{s_\times\}} \mathbb{P}(S_k = s), \tag{2.28}$$

where the first equality follows from the tail-sum formula for probability. In the second equality we used the fact that the event $\{T > k\} = \{S_k \neq s_\times\}$, because if the termination time is greater than k then the episode could not have terminated on the kth timestep, i.e. the kth state cannot be equal to the terminal state $s_\times$. And in the last equality we used the reasoning $\mathbb{P}(S_k \neq s_\times) = 1 - \mathbb{P}(S_k = s_\times) = \sum_{s \in S} \mathbb{P}(S_k = s) - \mathbb{P}(S_k = s_\times) = \sum_{s \in \mathcal{S} \setminus \{s_\times\}} \mathbb{P}(S_k = s)$. In general, the agent can't compute $\mathbb{E}[T]$, since it depends (in addition to the policy π) on the MDP dynamics μ and p.

The on-policy distribution of policy π in episodic tasks $d_\pi = \Delta(\mathcal{S} \setminus \{s_\times\} \times \mathbb{N}_0)$, is a joint distribution over the set of non-terminal states and the timesteps. We define it as follows:

$$d_\pi(s, k) := \frac{\mathbb{P}(S_k = s)}{\sum_{i=0}^{\infty} \sum_{x \in \mathcal{S} \setminus \{s_\times\}} \mathbb{P}(S_i = x)} \qquad \text{for all } s \in \mathcal{S} \setminus \{s_\times\} \text{ and } k \in \mathbb{N}_0. \tag{2.29}$$

To understand this distribution, let us marginalize it over the states and the timesteps separately:

$$d_\pi^{\text{states}}(s) := \sum_{k \in \mathbb{N}_0} d_\pi(s, k) = \frac{\sum_{k=0}^{\infty} \mathbb{P}(S_k = s)}{\sum_{x \in \mathcal{S} \setminus \{s_\times\}} \left[\sum_{i=0}^{\infty} \mathbb{P}(S_i = x)\right]} \qquad \text{for all } s \in \mathcal{S} \setminus \{s_\times\},$$

$$d_\pi^{\text{timesteps}}(k) := \sum_{s \in \mathcal{S} \setminus \{s_\times\}} d_\pi(s, k) = \frac{\mathbb{P}(k < T)}{\sum_{i=0}^{\infty} \mathbb{P}(i < T)} \qquad \text{for all } k \in \mathbb{N}_0.$$

From the expression for d_π^{states}, we see that the states that have a higher likelihood of occurring at

any step in the episode are proportionately more likely to be sampled. And from the expression for $d_\pi^{\text{timesteps}}$, we observe that the timesteps which are more likely to occur during an episode, i.e. they are less than the termination length of a given episode, have a proportionately higher probability of being sampled. Both these marginal distributions hint that d_π is related to the distribution of state-timestep pairs that the agent observes during its interaction with the environment.

The distribution $d_\pi(s, k)$ specifies how often the agent will observe a certain non-terminal state s at a given timestep k in a finite length episode when it follows policy π. Sampling a state and timestep from d_π is equivalent to picking a random timestep and the state on that timestep from the agent's experience. One way to see this is to imagine the set of all possible (finite-length) episodes. Then if we pick a state-timestep pair uniformly randomly from this list of episodes, the probability of picking a particular pair (s, k) is exactly equal to $d_\pi(s, k)$.

We are now ready to derive the PG estimator used by online AC[7]. We continue with the policy gradient expression given in Eq. 2.25 and move around the last summation, which gives us

$$
\begin{aligned}
&\nabla \mathcal{J}_\pi \\
&= \sum_{k=0}^{\infty} \sum_{t=k}^{\infty} \gamma^t \sum_{h_t, a_t, r_{t+1}} \mathbb{P}(H_t = h_t, A_t = a_t, R_{t+1} = r_{t+1}) \nabla \log \pi(a_k | s_k) r_{t+1} \\
&= \sum_{k=0}^{\infty} \sum_{t=k}^{\infty} \gamma^t \sum_{s_k, a_k, r_{t+1}} \mathbb{P}(S_k = s_k, A_k = a_k) \cdot \mathbb{P}(R_{t+1} = r_{t+1} | S_k = s_k, A_k = a_k) \nabla \log \pi(a_k | s_k) r_{t+1} \\
&= \sum_{k=0}^{\infty} \sum_{s_k, a_k} \mathbb{P}(S_k = s_k, A_k = a_k) \gamma^k \nabla \log \pi(a_k | s_k) \left[\sum_{t=k}^{\infty} \gamma^{t-k} \sum_{r_{t+1}} \mathbb{P}(R_{t+1} = r_{t+1} | S_k = s_k, A_k = a_k) r_{t+1} \right] \\
&= \sum_{k=0}^{\infty} \sum_{s} \mathbb{P}(S_k = s) \sum_{a} \pi(a|s) \gamma^k \nabla \log \pi(a|s) q_\pi(s, a) \qquad (2.30) \\
&= \sum_{k=0}^{\infty} \sum_{s \in \mathcal{S} \setminus \{s_\times\}} \mathbb{P}(S_k = s) \sum_{a} \pi(a|s) \gamma^k \nabla \log \pi(a|s) q_\pi(s, a) \\
&= \mathbb{E}[T] \cdot \sum_{k=0}^{\infty} \sum_{s \in \mathcal{S} \setminus \{s_\times\}} \underbrace{\left(\frac{\mathbb{P}(S_k = s)}{\sum_{i=0}^{\infty} \sum_{x \in \mathcal{S} \setminus \{s_\times\}} \mathbb{P}(S_i = x)} \right)}_{=: \, d_\pi(s,k)} \sum_{a} \pi(a|s) \gamma^k \nabla \log \pi(a|s) q_\pi(s, a) \\
&= \mathbb{E}[T] \cdot \mathbb{E}_{S, K \sim d_\pi; A \sim \pi(\cdot|S)} \left[\gamma^K \nabla \log \pi(A|S) q_\pi(S, A) \right], \qquad (2.31)
\end{aligned}
$$

where in the second line we used the marginalization trick followed by the definition of conditional probability; in the third line we moved around summations; in the fourth line we used the definition of $q_\pi(s_k, a_k)$, the (standard) MDP assumption that domain of the states and action at each timestep is the same, and the definition of conditional probability; in the fifth line we used the fact that

[7] The derivation presented here is nearly identical to that given in §13.2 of Sutton and Barto (2018).

$q_\pi(s_\times, a) = 0$ for all $a \in \mathcal{A}$, and in the second to last line we multiplied and divided by $\mathbb{E}[T]$[8]. Now, using similar techniques (marginalization trick along with rearrangement of sums[9]) as used to derive Eq. 2.30 from Eq. 2.25, we obtain the online actor-critic gradient estimator $\mathbf{g}^{\text{OAC}}(S, A, R, S', K)$:

$$\nabla \mathcal{J}_\pi \propto \mathbb{E}_{S,K\sim d_\pi; A\sim\pi(\cdot|S); S', R\sim p(\cdot,\cdot|S,A)} \underbrace{\Big[\gamma^K \nabla \log \pi(A|S)\Big(R + \gamma v_\pi(S') - v_\pi(S)\Big)\Big]}_{=:\mathbf{g}^{\text{OAC}}(S,A,R,S',K)}, \tag{2.32}$$

where we subtracted a value function baseline, and conveniently dropped $\mathbb{E}[T]$, which the agent cannot compute anyways, by using the fact that for a given policy π and the environment dynamics (which we assume to be fixed), $\mathbb{E}[T]$ is effectively a constant. This estimator depends on random variables that the agent has access to at each timestep, i.e. the agent doesn't need to wait until the end of the episode to update its policy. Also, note that this estimator uses $R + \gamma v_\pi(S') - v_\pi(S) =: \delta(S, R, S')$, a quantity known as the temporal difference (TD) error. Therefore, a natural choice for the agent to learn the value function is by using the TD-style updates (Sutton, 1988), again in an online fashion. We now present the complete online AC algorithm (Algorithm 3).

Algorithm 3 Online Actor-Critic

Input a policy $\pi_{\mathbf{w}}(a|s)$ and a value function estimate $\hat{v}_{\boldsymbol{\omega}}(s)$
Input the policy stepsize α and the value function stepsize β
Initialize $\mathbf{w}$ and $\boldsymbol{\omega}$
for each episode (in total number of episodes) **do**
  Sample $S \sim \mu$
  Initialize the geometric discounting term $I_\gamma \leftarrow 1$
  while the state S is not terminal, i.e. $S \neq s_\times$ **do**
    Sample $A \sim \pi_{\mathbf{w}}(\cdot|S)$ and $S', R \sim p(\cdot, \cdot|S, A)$
    $\delta \leftarrow R + \gamma \hat{v}_{\boldsymbol{\omega}}(S') - \hat{v}_{\boldsymbol{\omega}}(S)$ (if S' is terminal, then $\hat{v}_{\boldsymbol{\omega}}(S') = \hat{v}_{\boldsymbol{\omega}}(s_\times) := 0$)
    # Calculate the gradient estimator
    $\hat{\mathbf{g}}^{\text{OAC}} = I_\gamma \delta \nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(A|S)$
    # Update the policy
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha \cdot \hat{\mathbf{g}}^{\text{OAC}}$
    $I_\gamma \leftarrow \gamma I_\gamma$
    # Update the value function estimate
    $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \beta \cdot \delta \nabla_{\boldsymbol{\omega}} \hat{v}_{\boldsymbol{\omega}}(S)$
    $S \leftarrow S'$

[8] Note that we implicitly assumed that $\mathbb{E}_\pi[T] < \infty$ for all possible policies π. In fact, this assumption is necessary (see Point 4, Assumption 1, https://rltheory.github.io/documents/assignments/midterm.pdf) if we are considering the episodic undiscounted setting. However, it is not too difficult to think of situations where this could be violated in practice even for supposedly episodic MDPs (for instance, see Footnote 2 in §6). Also, this assumption is critical for this calculation: we do not know how to derive/motivate the online Actor-Critic estimator for settings where this assumption does not hold true; for instance continuing problems.

[9] It is also possible to equivalently use the law of iterated expectation (see https://en.wikipedia.org/wiki/Law_of_total_expectation) to do most of the calculations presented in this chapter. We instead chose to explicitly write out the summation and use the marginalization trick, since we find it to be more instructive.

The (online) actor-critic algorithms get their name from the interplay between two components: the actor, which refers to the policy, and the critic, which refers only to the baselines that are learned using a TD-style update[10]. However, because the policy weights are updated at each timestep, sampling from the agent's episode is not exactly equivalent to sampling from the distribution d_π (Eq. 2.32). As a result, the above actor-critic algorithm does not seem like a true gradient algorithm: it feels more like a two-timescale stochastic approximation algorithm which is inspired from gradient based calculations (cf. the TDC algorithm, Sutton et al., 2009). Prior works (Bhatnagar et al., 2009; Wu et al., 2020) use a two timescale viewpoint to prove that versions of the online AC algorithm (using linear function approximation with the average reward formulation for continuing tasks) converge. This algorithm, similar to REINFORCE, again used a single sample for estimating the gradient, and retained the same update order as before: the policy gets updated before the critic. Further, since we used the same stream of data to update both the actor and the critic, there will be some bias due to correlations (again see Wu et al., 2020); there will also be bias due to the Markovian nature of data collection (see Bresler et al., 2020). Also note that because we ignored the term $\mathbb{E}_\pi[T]$ while estimating the gradient, we are effectively scaling the stepsize with a different constant (equal to the inverse of the expected termination length for that policy) at each update: $\alpha/\mathbb{E}_\pi[T]$.

Degris et al. (2012a, 2012b) extend the online AC algorithm to use eligibility traces. Both these papers drop the γ^k term from their updates; Degris et al. (2012a) does so by making an approximation in the policy gradient derivation. Recently, Nota and Thomas (2020) showed that the estimators which drop this term do no take the gradient of the policy objective $\mathcal{J}_\pi$ and such updates can be sub-optimal in certain MDPs. Despite that, there is a good reason drop this term: γ^k geometrically discounts the contribution from state-action pairs that occur at later timesteps, making the algorithm sample inefficient. This dropping of γ^k now seems like a general practice and is adopted by most (if not all) deep policy gradient algorithms. However, in all the experiments in this thesis, we retain this γ^k term in our implementations of different policy gradient algorithms.

2.7 The Canonical PG Estimator

Let us now consider an estimator (first introduced in Sutton et al., 2000) that has a particularly succinct form. In fact, this is the expression that people often think of when they are referring to the *policy gradient theorem*. Moreover, it was also useful in implementing the expected PG updates

[10]Interestingly, the baseline in REINFORCE is not considered as a critic, because it is learned using Monte-Carlo based returns. Maybe one reason (admittingly, we are unsure) for this is that TD-style updates allow online learning which increases the effect actor and critic have on each other, whereas in REINFORCE, the baseline and the policy are not as tightly coupled (also see Chapter 13, Sutton and Barto, 2018).

for the tabular experiments given in §6. We rearrange the summations in Eq. 2.30 to get

$$\begin{aligned}\nabla \mathcal{J}_\pi &= \sum_s \left[\sum_{k=0}^{\infty} \gamma^k \mathbb{P}(S_k = s)\right] \sum_a \pi(a|s) \nabla \log \pi(a|s) q_\pi(s,a) \\ &= \sum_s \nu_\pi(s) \sum_a \pi(a|s) \nabla \log \pi(a|s) q_\pi(s,a) \\ &= \mathbb{E}_{S\sim\nu_\pi; A\sim\pi(\cdot|S)} \underbrace{\Big[\nabla \log \pi(A|S) q_\pi(S,A)\Big]}_{=:\mathbf{g}^{PG-Other}(S,A)}, \end{aligned} \tag{2.33}$$

where in the second line we used the γ-discounted state visitation distribution[11] (also known as the occupancy measure) under policy π, defined as

$$\nu_\pi(s) := \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(S_k = s). \tag{2.34}$$

Note that each of the above PG estimators (given by Eqs. 2.24, 2.26, 2.31, and 2.33) contain the term $\nabla \log \pi(A|S)$; it is because of this reason, each these estimators is often known as a log-likelihood type of policy gradient estimator.

2.8 Differentiation of Vector Variables

In this section, we first explain the vector notation we use and then state some useful vector differentiation rules. We assume that $\boldsymbol{\theta} \in \mathbb{R}^m$ is a vector and θ_i denotes the ith element of $\boldsymbol{\theta}$, $\mathbf{c} \in \mathbb{R}^m$ is a vector independent of $\boldsymbol{\theta}$, $\text{diag}(\boldsymbol{\theta})$ denotes the $m \times m$ diagonal matrix with θ_j as the jth element on the main diagonal, the function $\alpha : \mathbb{R}^m \to \mathbb{R}$ defined as $\alpha(\boldsymbol{\theta}) := \alpha(\theta_1, \theta_2, \ldots, \theta_m) \in \mathbb{R}$ is a scalar function dependent on the vector $\boldsymbol{\theta}$, the function $\mathbf{x} : \mathbb{R}^m \to \mathbb{R}^n$ defined as $\mathbf{x}(\boldsymbol{\theta}) := [x_1(\boldsymbol{\theta})\ x_2(\boldsymbol{\theta})\ \ldots\ x_n(\boldsymbol{\theta})]^\top \in \mathbb{R}^n$ is a vector function of $\boldsymbol{\theta}$, and $\mathbf{y}(\mathbf{x}(\boldsymbol{\theta})) \in \mathbb{R}^p$ is a vector function which depends on $\mathbf{x}$ and consequently on $\boldsymbol{\theta}$ as well. Finally, for a function $f : \mathbb{R} \to \mathbb{R}$, we define $f(\boldsymbol{\theta}) := [f(\theta_1)\ f(\theta_2)\ \cdots\ f(\theta_m)]^\top$ as the vector obtained by applying the function f to the vector $\boldsymbol{\theta}$ elementwise. For example, $e^{\boldsymbol{\theta}} = [e^{\theta_1}\ e^{\theta_2}\ \cdots\ e^{\theta_m}]^\top$.

The gradient of a scalar $\alpha(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is the vector defined as

$$\nabla_{\boldsymbol{\theta}} \alpha(\boldsymbol{\theta}) := \left[\frac{\partial \alpha(\boldsymbol{\theta})}{\partial \theta_1} \quad \frac{\partial \alpha(\boldsymbol{\theta})}{\partial \theta_2} \quad \cdots \quad \frac{\partial \alpha(\boldsymbol{\theta})}{\partial \theta_m}\right]^\top_{m\times 1}.$$

[11] Technically, ν_π is not a probability distribution, since it does not sum to one: $\sum_s \nu_\pi(s) = \frac{1}{1-\gamma}$.

We overload the gradient operator to take gradient of a vector $\mathbf{x}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta}) &:= \Big[\nabla_{\boldsymbol{\theta}} x_1(\boldsymbol{\theta}) \quad \nabla_{\boldsymbol{\theta}} x_2(\boldsymbol{\theta}) \quad \dots \quad \nabla_{\boldsymbol{\theta}} x_n(\boldsymbol{\theta})\Big]_{m\times n} \\ &\equiv \begin{bmatrix} \frac{\partial x_1(\boldsymbol{\theta})}{\partial \theta_1} & \frac{\partial x_2(\boldsymbol{\theta})}{\partial \theta_1} & \dots & \frac{\partial x_n(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial x_1(\boldsymbol{\theta})}{\partial \theta_2} & \frac{\partial x_2(\boldsymbol{\theta})}{\partial \theta_2} & \dots & \frac{\partial x_n(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_1(\boldsymbol{\theta})}{\partial \theta_m} & \frac{\partial x_2(\boldsymbol{\theta})}{\partial \theta_m} & \dots & \frac{\partial x_n(\boldsymbol{\theta})}{\partial \theta_m} \end{bmatrix}_{m\times n}.\end{aligned}$$

According to the above definition, $\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta})$ is the transpose of the usual Jacobian matrix, and readily reduces to the gradient vector if $\mathbf{x}(\boldsymbol{\theta})$ is a scalar (i.e. $n = 1$). Further, using this convention, if $\boldsymbol{\theta}$ is a scalar (i.e. $m = 1$), then

$$\nabla_{\theta}\,\mathbf{x}(\theta) = \Big[\frac{\partial x_1(\theta)}{\partial \theta} \quad \frac{\partial x_2(\theta)}{\partial \theta} \quad \dots \quad \frac{\partial x_n(\theta)}{\partial \theta}\Big]_{1\times n}.$$

Finally, if both $\mathbf{x}(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ are scalars (i.e. $m = n = 1$), then

$$\nabla_{\theta} x(\theta) = \frac{\partial x(\theta)}{\partial \theta}.$$

We are now ready to state some useful propositions, along with their proofs, regarding the differentiation of vector variables.

Proposition 2. $\nabla_{\boldsymbol{\theta}}\big(\,\boldsymbol{\theta}^{\top}\,\mathbf{c}\,\big) = \nabla_{\boldsymbol{\theta}}\big(\,\mathbf{c}^{\top}\,\boldsymbol{\theta}\,\big) = \mathbf{c}\,.$

Proof.

$$\frac{\partial}{\partial \theta_k}\,\boldsymbol{\theta}^{\top}\,\mathbf{c} = \frac{\partial}{\partial \theta_k}\sum_{i=1}^{m}\theta_i c_i = c_k \qquad \Rightarrow \qquad \nabla_{\boldsymbol{\theta}}\,\boldsymbol{\theta}^{\top}\,\mathbf{c} = \mathbf{c}\,.$$

□

Proposition 3. $\nabla_{\boldsymbol{\theta}}\Big[\alpha(\boldsymbol{\theta})\,\mathbf{x}(\boldsymbol{\theta})\Big] = \mathbf{x}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\alpha(\boldsymbol{\theta})^{\top} + \alpha(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta}).$

Proof.

$$\begin{aligned}&\Big(\nabla_{\boldsymbol{\theta}}\Big[\alpha(\boldsymbol{\theta})\,\mathbf{x}(\boldsymbol{\theta})\Big]\Big)_{ij} = \frac{\partial}{\partial \theta_i}\Big[\alpha(\boldsymbol{\theta})x_j(\boldsymbol{\theta})\Big] = \frac{\partial \alpha(\boldsymbol{\theta})}{\partial \theta_i}x_j(\boldsymbol{\theta}) + \alpha(\boldsymbol{\theta})\frac{\partial x(\boldsymbol{\theta})_j}{\partial \theta_i} \\ \Rightarrow \qquad &\nabla_{\boldsymbol{\theta}}\Big[\alpha(\boldsymbol{\theta})\,\mathbf{x}(\boldsymbol{\theta})\Big] = \nabla_{\boldsymbol{\theta}}\alpha(\boldsymbol{\theta})\,\mathbf{x}(\boldsymbol{\theta})^{\top} + \alpha(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta}).\end{aligned}$$

□

Proposition 4. $\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta})) = \Big[\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta})\Big]\Big[\nabla_{\mathbf{x}(\boldsymbol{\theta})}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))\Big].$

Proof. We show this result taking the total derivative[12] of $\mathbf{y}$ with respect to $\boldsymbol{\theta}$.

$$\begin{aligned}
&\Big(\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))\Big)_{ij} = \frac{\partial y_j(\mathbf{x}(\boldsymbol{\theta}))}{\partial \theta_i} = \sum_{k=1}^{m} \frac{\partial y_j(\mathbf{x}(\boldsymbol{\theta}))}{\partial x_k(\boldsymbol{\theta})} \frac{\partial x_k(\boldsymbol{\theta})}{\partial \theta_i} \\
\Rightarrow \quad &\Big(\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))\Big)_{\cdot j} = \sum_{k=1}^{m} \frac{\partial y_j(\mathbf{x}(\boldsymbol{\theta}))}{\partial x_k(\boldsymbol{\theta})} \nabla_{\boldsymbol{\theta}} x_k(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta}) \nabla_{\mathbf{x}(\boldsymbol{\theta})} y_j(\mathbf{x}(\boldsymbol{\theta})) \\
\Rightarrow \quad &\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta})) = \Big[\nabla_{\boldsymbol{\theta}}\,\mathbf{x}(\boldsymbol{\theta})\Big]\Big[\nabla_{\mathbf{x}(\boldsymbol{\theta})}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))\Big],
\end{aligned}$$

where $\Big(\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))\Big)_{\cdot j}$ denotes the jth column of the matrix $\nabla_{\boldsymbol{\theta}}\,\mathbf{y}(\mathbf{x}(\boldsymbol{\theta}))$. □

Proposition 5. *Let f be a function, applied elementwise to a vector $\boldsymbol{\theta}$, i.e.*

$$f(\boldsymbol{\theta}) := [f(\theta_1)\ f(\theta_2)\ \cdots\ f(\theta_m)]^\top.$$

Then

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = diag\left(\begin{bmatrix} \frac{\partial f(\theta_1)}{\partial \theta_1} & \frac{\partial f(\theta_2)}{\partial \theta_2} & \cdots & \frac{\partial f(\theta_m)}{\partial \theta_m} \end{bmatrix}^\top\right).$$

[12] See https://mathworld.wolfram.com/TotalDerivative.html.

Chapter 3

Alternate Gradient Bandits

This chapter introduces the alternate gradient bandit estimator and discusses its properties.

3.1 Gradient Bandits using Vector Notation

We begin by describing the vector notation which makes our calculations succinct and easier to follow. For a k-armed bandit, let $\boldsymbol{\pi}, \mathbf{r} \in \mathbb{R}^k$ denote the policy and the reward vectors with $[\boldsymbol{\pi}]_k = \pi(a_k)$ and $[\mathbf{r}]_k = r(a_k)$. Using these vectors, the expectations of scalar quantities become vector inner products: $\mathcal{J}_\pi \equiv r_\pi = \mathbb{E}_\pi[R] = \sum_a \pi(a) r(a) = \boldsymbol{\pi}^\top \mathbf{r}$. The softmax policy (defined by Eq. 2.11) can be written as

$$\boldsymbol{\pi} = e^{\boldsymbol{\theta}} / (\mathbf{1}^\top e^{\boldsymbol{\theta}}), \tag{3.1}$$

where the vector $e^{\boldsymbol{\theta}} \in \mathbb{R}^k$ is defined as $[e^{\boldsymbol{\theta}}]_a := e^{\theta_a}$. The gradient of $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ with respect to $\boldsymbol{\theta}$ can also be computed using just differentiation of vector variables:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi} &= \nabla_{\boldsymbol{\theta}} \left[\frac{e^{\boldsymbol{\theta}}}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} \right] = e^{\boldsymbol{\theta}} \left(\nabla_{\boldsymbol{\theta}} \left[\frac{1}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} \right] \right)^\top + \frac{\nabla_{\boldsymbol{\theta}} e^{\boldsymbol{\theta}}}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} && \text{(using Proposition 3)} \\
&= e^{\boldsymbol{\theta}} \left(\nabla_{\boldsymbol{\theta}} e^{\boldsymbol{\theta}} \nabla_{e^{\boldsymbol{\theta}}} \left[\frac{1}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} \right] \right)^\top + \frac{\nabla_{\boldsymbol{\theta}} e^{\boldsymbol{\theta}}}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} && \text{(using Proposition 4)} \\
&= e^{\boldsymbol{\theta}} \left(\nabla_{\boldsymbol{\theta}} e^{\boldsymbol{\theta}} \nabla_{e^{\boldsymbol{\theta}}} (\mathbf{1}^\top e^{\boldsymbol{\theta}}) \nabla_{(\mathbf{1}^\top e^{\boldsymbol{\theta}})} \left[\frac{1}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} \right] \right)^\top + \frac{\nabla_{\boldsymbol{\theta}} e^{\boldsymbol{\theta}}}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} && \text{(using Proposition 4)} \\
&= -e^{\boldsymbol{\theta}} \left(\operatorname{diag}(e^{\boldsymbol{\theta}}) \mathbf{1} \frac{1}{(\mathbf{1}^\top e^{\boldsymbol{\theta}})^2} \right)^\top + \frac{\operatorname{diag}(e^{\boldsymbol{\theta}})}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} && \text{(using Proposition 5)} \\
&= -\frac{e^{\boldsymbol{\theta}}}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} \mathbf{1}^\top \frac{\operatorname{diag}(e^{\boldsymbol{\theta}})}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} + \frac{\operatorname{diag}(e^{\boldsymbol{\theta}})}{\mathbf{1}^\top e^{\boldsymbol{\theta}}} = -\boldsymbol{\pi}\, \mathbf{1}^\top \operatorname{diag}(\boldsymbol{\pi}) + \operatorname{diag}(\boldsymbol{\pi}) \\
&= (\mathbf{I} - \boldsymbol{\pi}\, \mathbf{1}^\top) \operatorname{diag}(\boldsymbol{\pi}), && (3.2)
\end{aligned}$$

where $\mathbf{1}$ represents a $k \times 1$ vector of ones. As a sanity check, note that the gradient in Eq. 3.2 corresponds to the derivative in Eq. 2.12.

Given the above expression, the gradient of the objective for the softmax policy for gradient bandits (Eq. 2.14) can be recovered as

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi &= \nabla_{\boldsymbol{\theta}} \big(\boldsymbol{\pi}^\top \mathbf{r} \big) = \big[\nabla_{\boldsymbol{\theta}} \, \boldsymbol{\pi} \big] \big[\nabla_{\boldsymbol{\pi}} (\boldsymbol{\pi}^\top \mathbf{r}) \big] = (\mathbf{I} - \boldsymbol{\pi} \, \mathbf{1}^\top) \operatorname{diag}(\boldsymbol{\pi}) \, \mathbf{r} \\ &= \operatorname{diag}(\boldsymbol{\pi}) \, \mathbf{r} - \boldsymbol{\pi} \, \mathbf{1}^\top \operatorname{diag}(\boldsymbol{\pi}) \, \mathbf{r} = \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi} \, \boldsymbol{\pi}^\top \mathbf{r} \qquad (3.3) \\ &= \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi} \, r_\pi = \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \mathbf{1}), \qquad (3.4)\end{aligned}$$

where $\odot$ represents the elementwise vector product, which for any two vectors $\mathbf{a}$ and $\mathbf{b}$ is defined as $(\mathbf{a} \odot \mathbf{b})_i = a_i b_i$. Now continuing from Eq. 3.3, the regular PG estimator can be derived as follows

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi &= \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi} \, \boldsymbol{\pi}^\top \mathbf{r} = \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi} \, \mathbf{r}^\top \boldsymbol{\pi} = [\operatorname{diag}(\mathbf{r}) - \boldsymbol{\pi} \, \mathbf{r}^\top] \, \boldsymbol{\pi} \\ &= \sum_a \pi(a) \big[r(a) \, \mathbf{e}_a - r(a) \, \boldsymbol{\pi} \big] = \sum_a \pi(a) r(a) \big[\mathbf{e}_a - \boldsymbol{\pi} \big] = \sum_a \pi(a) \sum_x p(x|a) x \big[\mathbf{e}_a - \boldsymbol{\pi} \big] \\ &= \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)} \underbrace{[R(\mathbf{e}_A - \boldsymbol{\pi})]}_{=: \mathbf{g}^{\text{No-b}}(R,A)}, \qquad (3.5)\end{aligned}$$

where $\mathbf{e}_A$ is the Ath column of the identity matrix, i.e. a basis vector (or a one-hot vector) with the element corresponding to action A equal to one and all other elements equal to zero. We can also subtract any action independent baseline from the reward in the above gradient estimator without making it biased:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi = \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)} \underbrace{[(R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi})]}_{=: \mathbf{g}^{\text{REG}}(R,A)} = \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)} \underbrace{[(R - b)(\mathbf{e}_A - \boldsymbol{\pi})]}_{=: \hat{\mathbf{g}}^{\text{REG}}(R,A)}, \qquad (3.6)$$

where b is any action independent reward baseline.

3.2 The Alternate Gradient Bandit Estimator

We now derive the alternate gradient bandit estimator. Again continuing from Eq. 3.3, gives us

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi &= \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi} \, \boldsymbol{\pi}^\top \mathbf{r} = \boldsymbol{\pi} \odot \mathbf{r} - r_\pi \, \boldsymbol{\pi} = [\operatorname{diag}(\mathbf{r}) - r_\pi \, \mathbf{I}] \, \boldsymbol{\pi} \\ &= \sum_a \pi(a) \big[r(a) \, \mathbf{e}_a - r_\pi \, \mathbf{e}_a \big] = \sum_a \pi(a) \, \mathbf{e}_a \big[r(a) - r_\pi \big] = \sum_a \pi(a) \, \mathbf{e}_a \sum_x p(x|a) \big[x - r_\pi \big] \\ &= \mathbb{E}_{A \sim \pi, R \sim p(\cdot|A)} \underbrace{[(R - r_\pi) \, \mathbf{e}_A]}_{=: \mathbf{g}^{\text{ALT}}(R,A)}. \qquad (3.7)\end{aligned}$$

The alternate estimator $\mathbf{g}^{\text{ALT}}(R, A)$ is similar to the regular estimator except that it is missing the $-\boldsymbol{\pi}$ term that $\mathbf{g}^{\text{REG}}(R, A)$ has. This fact allows the alternate estimator to utilize the noise in the reward signal. We discuss this point further in §3.3. Further, unlike the regular estimator, where we additionally introduced the baseline r_π, the term r_π automatically came up during derivation of the alternate estimator. With the alternate estimator, we cannot arbitrarily add any action independent baseline or drop r_π: the alternate estimator with a baseline $b \neq r_\pi$ becomes biased (for a detailed discussion, see §3.4):

$$\mathbb{E}\underbrace{[\mathbf{e}_A(R-b)]}_{=:\hat{\mathbf{g}}^{\text{ALT}}(R,A)} \neq \nabla_{\boldsymbol{\theta}} J_\pi = \mathbb{E}[\mathbf{g}^{\text{ALT}}(R, A)]. \tag{3.8}$$

This biasedness of the alternate estimator gives it some desirable properties that we discuss in §3.4.

Also note that the regular gradient estimator $\mathbf{g}^{\text{REG}}(R, A)$, in general, would have a non-zero component corresponding to each action and therefore would update all the action preferences at each timestep. In contrast, the alternate gradient estimator has exactly one non-zero component and would only update the preference corresponding to the sampled action at each timestep. This fact, as we show in §3.6, allows the gradient bandit algorithm employing the alternate estimator to become computationally per-timestep faster than the gradient bandit algorithm with the regular estimator.

Before we discuss all these properties in further detail, we give two examples that help us develop some intuition behind the working of these two gradient estimators. The first example illustrates how the alternate estimator can utilize the reward noise to escape a saturated region.

Example 1. *Consider a 3-armed bandit with $\mathcal{A} = \{a_0, a_1, a_2\}$, the expected reward $\mathbf{r} = [0\ 0\ 1]^\top$, and the policy $\boldsymbol{\pi} = [1\ 0\ 0]^\top$. The rewards are perturbed with a normally distributed noise $\epsilon \sim \mathcal{N}(0, 1)$, so that upon picking action A, the agent receives the reward $r(A) + \epsilon$. Since $r_\pi = 0$, $R - r_\pi = 0 + \epsilon - 0 = \epsilon$. And because the agent samples $A = a_0$ at each timestep, we obtain*

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} J_\pi &= \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \mathbf{1}) = [1\ 0\ 0]^\top \odot [0\ 0\ 1]^\top = [0\ 0\ 0]^\top, \\
\mathbf{g}^{REG}(A, R) &= (R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi}) = \epsilon \cdot \Big([1\ 0\ 0]^\top - [1\ 0\ 0]^\top\Big) = [0\ 0\ 0]^\top, \text{ and} \\
\mathbf{g}^{ALT}(A, R) &= (R - r_\pi)\, \mathbf{e}_A = \epsilon \cdot [1\ 0\ 0]^\top = [\epsilon\ 0\ 0]^\top.
\end{aligned}$$

This example shows that at a sub-optimal corner[1], the agent with the alternate estimator effectively does a random walk until it escapes that region and can choose other actions to get a non-zero reward signal. Further, doing a random walk at the optimal corner is not problematic,

[1]This examples requires the policy to lie at the boundary of the probability simplex which is unsatisfiable for softmax policies, as this would require some of the action preferences to be infinitely large. However, it is easy to construct a similar example for policies that are near, but not at, the corner and show that a similar reasoning holds.

since there is an attractive gradient signal that pulls the agent towards this corner as soon as it moves away. The second example shows how an optimistic baseline can also help reduce policy saturation by providing a signal that is not mere noise.

Example 2. *Consider the 3-armed bandit and the policy of the previous example. Now assume that the agent doesn't have access to r_π and instead uses an optimistic baseline $b = \delta \gg 0$. Again, the agent will sample $A = a_0$ at each timestep. Therefore:*

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi &= \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \, \mathbf{1}) = [1\ 0\ 0]^\top \odot [0\ 0\ 1]^\top = [0\ 0\ 0]^\top, \\
\hat{\mathbf{g}}^{\,REG}(A, R) &= (R - b)(\mathbf{e}_A - \boldsymbol{\pi}) = (\epsilon - \delta) \cdot \Big([1\ 0\ 0]^\top - [1\ 0\ 0]^\top \Big) = [0\ 0\ 0]^\top, \text{ and} \\
\hat{\mathbf{g}}^{\,ALT}(A, R) &= (R - b)\, \mathbf{e}_A = (\epsilon - \delta) \cdot [1\ 0\ 0]^\top = [(\epsilon - \delta)\ 0\ 0]^\top.
\end{aligned}$$

As this example shows, with a suitable baseline (in this case a positive baseline that can override the reward noise), the alternate estimator can use the signal from the baseline to escape the saturated policy region, without relying solely on the reward noise. We formalize these two points in the sections ahead.

3.3 Alternate Estimator Utilizes Reward Noise

In this section, we first show that the expected policy gradient at saturated regions, or equivalently the *corners* of the probability simplex boundary (see Figure 3.2), is zero. We then show that at these corners, the variance of the regular estimator is zero as well, but the variance of the alternate estimator is equal to the variance of the reward noise which in general is non-zero. The consequence of these two facts is that whenever the policy is saturated, i.e. it places a high probability mass on some actions, the expected gradient becomes close to zero, and therefore the policy weights are not updated. Further, since the variance of the regular estimator at these corners is also zero, there is neither any gradient signal nor any noise, and the agent is unable to escape the sub-optimal region. In contrast, the alternate estimator, despite being zero in expectation, has a non-zero variance and therefore the agent can utilize the reward noise to escape the sub-optimal region.

Proposition 6. *Define $\mathcal{I}_c := \{a \mid r(a) = c\}$ for some $c \in \mathbb{R}$. Consider a constant c such that*[2] *$\mathcal{I}_c \neq \emptyset$ and that the policy is saturated on the actions in the set $\mathcal{I}_c$, i.e. $\sum_{a \in \mathcal{I}_c} \pi(a) = 1$. Then the expected policy gradient for softmax policies is zero:*

$$\nabla_{\boldsymbol{\theta}} \mathcal{J} = \mathbb{E}[\mathbf{g}^{REG}(A, R)] = \mathbb{E}[\hat{\mathbf{g}}^{\,REG}(A, R)] = \mathbb{E}[\mathbf{g}^{ALT}(A, R)] = \mathbf{0}\,.$$

Proof. We set $\nabla_{\boldsymbol{\theta}} \mathcal{J} = \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \mathbf{1}) = \mathbf{0}$ to obtain a system of $|\mathcal{A}| + 1$ linear equations in $|\mathcal{A}|$

[2] We can always find such a c by choosing it to be equal to the expected reward of some action.

variables:

$$\pi(a) \cdot (r(a) - r_\pi) = 0, \ \forall a \in \mathcal{A} \qquad \text{and} \qquad \sum_{a \in \mathcal{A}} \pi(a) = 1. \tag{3.9}$$

Consider $\mathcal{I}_c = \{a \mid r(a) = c\}$ where c is equal to the expected reward for some action. Then for a policy satisfying $\sum_{a \in \mathcal{I}_c} \pi(a) = 1$, we immediately see that $\pi(a) = 0, \forall a \notin \mathcal{I}_c$ and $r_\pi = c = r(a), \forall a \in \mathcal{I}_c$. Consequently, this policy would also satisfy Eqs. 3.9 and thereby $\nabla_{\boldsymbol{\theta}} \mathcal{J} = \mathbf{0}$. □

The above proposition shows that the expected gradient bandit update $\nabla_{\boldsymbol{\theta}} \mathcal{J}_\pi$ is zero only at the simplex boundary which can happen either at any one of the corners, or at a point on one of the face (i.e. a sub-space) of the simplex such that all the actions on that face have the same expected reward. The next proposition considers the variance of the estimators at the simplex corners.

Proposition 7. *Let $\sigma(a)^2 := \mathbb{V}[R|A = a]$ denote the conditional variance of the reward corresponding to action a. Assume that the policy is saturated on the action c, i.e., $\pi(c) = 1$. Also assume that $\sigma(c)^2 \neq 0$. Then, the variance of the regular PG estimator (with or without a baseline) is zero: $\mathbb{V}[\mathbf{g}^{REG}(A)] = \mathbb{V}[\hat{\mathbf{g}}^{REG}(A, R)] = \mathbf{0}$. Whereas, the variance of the alternate PG estimator (with or without a baseline) is non-zero: $\mathbb{V}[\mathbf{g}^{ALT}(A, R)] = \mathbb{V}[\hat{\mathbf{g}}^{ALT}(A, R)] = \sigma(c)^2 \, \mathbf{e}_c$.*

Proof. As defined in the proof statement, the conditional variance of the rewards given an action a is denoted using $\sigma(a)^2$, i.e. $\sigma(a)^2 := \mathbb{V}[R|A = a]$. Further, define $\boldsymbol{\sigma}^2$ to be the conditional variance vector with $[\boldsymbol{\sigma}^2]_k = \sigma(a_k)^2$. Also for brevity, let $\mathbf{x}^2 := \mathbf{x} \odot \mathbf{x}$ for any vector $\mathbf{x}$. Then, it is straightforward to show the following statements:

$$\mathbb{E}[R] = \sum_a \pi(a) \sum_x p(x|a) x = \sum_a \pi(a) r(a) = r_\pi, \tag{3.10}$$

$$\begin{aligned} \mathbb{E}[R^2] &= \mathbb{E}_A \Big[\mathbb{E}_R[R^2|A] \Big] = \mathbb{E}_A \Big[\mathbb{V}[R|A] + \mathbb{E}[R|A]^2 \Big] = \sum_a \pi(a) \Big(\sigma(a)^2 + r(a)^2 \Big) \\ &= \boldsymbol{\pi}^\top (\boldsymbol{\sigma}^2 + \mathbf{r}^2), \end{aligned} \tag{3.11}$$

$$\mathbb{E}[\mathbf{e}_A] = \sum_a \pi(a) \, \mathbf{e}_a = \boldsymbol{\pi}, \tag{3.12}$$

$$\mathbb{E}[\mathbf{e}_A \, R] = \sum_a \pi(a) \, \mathbf{e}_a \sum_x p(x|a) x = \sum_a \pi(a) \, \mathbf{e}_a \, r(a) = \boldsymbol{\pi} \odot \mathbf{r}, \tag{3.13}$$

$$\mathbb{E}[\mathbf{e}_A \, R^2] = \mathbb{E}_A \Big[\mathbf{e}_A \, \mathbb{E}_R[R^2|A] \Big] = \sum_a \pi(a) \, \mathbf{e}_a \Big(\sigma(a)^2 + r(a)^2 \Big) = \boldsymbol{\pi} \odot (\boldsymbol{\sigma}^2 + \mathbf{r}^2). \tag{3.14}$$

Next, let $\boldsymbol{\eta}$ be an action independent vector and b be an action independent scalar. And assume that the variance for vector variables is calculated elementwise, i.e. the variance and covariance of arbitrary vector random variables $\mathbf{X}$ and $\mathbf{Y}$ are defined as $\mathbb{V}[\mathbf{X}] := \mathbb{E}[\mathbf{X} \odot \mathbf{X}] - \mathbb{E}[\mathbf{X}] \odot \mathbb{E}[\mathbf{X}]$ and

$\mathrm{Cov}(\mathbf{X}, \mathbf{Y}) := \mathbb{E}[\mathbf{X} \odot \mathbf{Y}] - \mathbb{E}[\mathbf{X}] \odot \mathbb{E}[\mathbf{Y}]$ respectively. Then,

$$\begin{aligned}
&\mathbb{V}[(\mathbf{e}_A - \boldsymbol{\eta})(R - b)] \\
&= \mathbb{V}[\mathbf{e}_A\, R - \mathbf{e}_A\, b - \boldsymbol{\eta}\, R + \boldsymbol{\eta}\, b] = \mathbb{V}[\mathbf{e}_A\, R - \mathbf{e}_A\, b - \boldsymbol{\eta}\, R] \\
&= \mathbb{V}[\mathbf{e}_A\, R] + \mathbb{V}[\mathbf{e}_A\, b] + \mathbb{V}[\boldsymbol{\eta}\, R] - 2\,\mathrm{Cov}(\mathbf{e}_A\, R, \mathbf{e}_A\, b) - 2\,\mathrm{Cov}(\mathbf{e}_A\, b, \boldsymbol{\eta}\, R) + 2\,\mathrm{Cov}(\mathbf{e}_A\, b, \boldsymbol{\eta}\, R) \\
&= \mathbb{V}[\mathbf{e}_A\, R] + \mathbb{V}[\mathbf{e}_A] b^2 + \mathbb{V}[R]\, \boldsymbol{\eta}^2 - 2\,\mathrm{Cov}(\mathbf{e}_A\, R, \mathbf{e}_A) b - 2\,\mathrm{Cov}(\mathbf{e}_A\, R, R) \odot \boldsymbol{\eta} + 2\,\mathrm{Cov}(\mathbf{e}_A, R) \odot \boldsymbol{\eta}\, b.
\end{aligned} \tag{3.15}$$

Simplifying the above expression for a policy saturated at action c, i.e. $\boldsymbol{\pi} = \mathbf{e}_c$, gives us

$$\begin{aligned}
&\mathbb{V}[\mathbf{e}_A\, R] + \mathbb{V}[\mathbf{e}_A] b^2 + \mathbb{V}[R]\, \boldsymbol{\eta}^2 - 2\,\mathrm{Cov}(\mathbf{e}_A\, R, \mathbf{e}_A) b - 2\,\mathrm{Cov}(\mathbf{e}_A\, R, R) \odot \boldsymbol{\eta} + 2\,\mathrm{Cov}(\mathbf{e}_A, R) \odot \boldsymbol{\eta}\, b \\
&= \mathbf{e}_c \odot \boldsymbol{\sigma}^2 + \mathbf{0} + \sigma(c)^2\, \boldsymbol{\eta}^2 - \mathbf{0} - 2\, \boldsymbol{\sigma} \odot \mathbf{e}_c \odot \boldsymbol{\eta} + \mathbf{0} \\
&= \sigma(c)^2\, \mathbf{e}_c + \sigma(c)^2\, \boldsymbol{\eta}^2 - 2\sigma(c)^2 \eta(c)^2\, \mathbf{e}_c = \begin{cases} \sigma(c)^2\, \mathbf{e}_c & \text{if } \boldsymbol{\eta} = \mathbf{0} \text{ (i.e. for alternate estimator), or} \\ \mathbf{0} & \text{if } \boldsymbol{\eta} = \boldsymbol{\pi} = \mathbf{e}_c \text{ (i.e. for the regular estimator).} \end{cases}
\end{aligned}$$

This concludes the proof. □

Note that these two propositions again require the policy to have support only at a subset of all the actions. This condition cannot be fulfilled for softmax policies since the action preferences for these actions will need to be infinitely large. Despite that, using continuity arguments, we can still reason that for the regular estimator, both the expected gradient and its variance vanish in the proximity of the simplex boundary, whereas the alternate estimator will have non-zero variance.

3.4 Alternate Estimator Utilizes the Bias in the Critic Estimate

The alternate estimator can also utilize the bias in the critic estimate b to escape the saturated regions of the policy space. If the critic estimate b is not equal to r_π, then the alternate estimator becomes biased, whereas the regular estimator does not. And for the alternate estimator, this bias helps make the policy more uniform thereby moving the agent out of the saturated region. To see this biasedness, consider a baseline $b \neq r_\pi$ and a vector $\boldsymbol{\eta}$ which we will set equal to either $\boldsymbol{\pi}$ for the regular estimator, or $\mathbf{0}$ for the alternate estimator. Then it is straightforward to see that

$$\begin{aligned}
\mathbb{E}[(\mathbf{e}_A - \boldsymbol{\eta})(R - b)] &= \mathbb{E}[\mathbf{e}_A\, R] - \mathbb{E}[\mathbf{e}_A\, b] - \mathbb{E}[R]\, \boldsymbol{\eta} + \boldsymbol{\eta}\, b = \boldsymbol{\pi} \odot \mathbf{r} - \boldsymbol{\pi}\, b - r_\pi\, \boldsymbol{\eta} + \boldsymbol{\eta}\, b \\
&= \begin{cases} \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \mathbf{1}) & \text{if } \boldsymbol{\eta} = \boldsymbol{\pi}, \qquad \text{and} \\ \boldsymbol{\pi} \odot (\mathbf{r} - b\mathbf{1}) & \text{if } \boldsymbol{\eta} = \mathbf{0}\,. \end{cases}
\end{aligned} \tag{3.16}$$

Therefore,

$$\nabla_{\boldsymbol{\theta}} \mathcal{J} = \mathbb{E}[\hat{\mathbf{g}}^{\,\mathrm{REG}}(A, R)] = \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi \mathbf{1}) \neq \boldsymbol{\pi} \odot (\mathbf{r} - b\mathbf{1}) = \mathbb{E}[\hat{\mathbf{g}}^{\,\mathrm{ALT}}(A, R)].$$

Interestingly, this biased update $\mathbb{E}[\hat{\mathbf{g}}^{\,\mathrm{ALT}}(A, R)]$ is not the gradient of any function. To show this, we adopt the same process as Barnard (1993) did for the TD update.

Proposition 8. *The biased expected gradient bandits update is not a gradient of any objective function, i.e. for no function $\mathcal{L}$, $\nabla_{\boldsymbol{\theta}} \mathcal{L} = \boldsymbol{\pi} \odot (\mathbf{r} - b)$.*

Proof. We prove this statement by contradiction. Assume that the biased update is the gradient of some function $\mathcal{L}$, i.e. $\frac{\partial \mathcal{L}}{\partial \theta_a} = \pi(a) \cdot (r(a) - b)$. Then,

$$\frac{\partial^2 \mathcal{L}}{\partial \theta_c \partial \theta_a} = \pi(a)[\mathbb{I}(a = c) - \pi(c)](r(a) - b) \quad \Rightarrow \quad \frac{\partial^2 \mathcal{L}}{\partial \theta_c \partial \theta_a} \neq \frac{\partial^2 \mathcal{L}}{\partial \theta_a \partial \theta_c}, \quad \text{for } a \neq c,$$

which gives a contradiction[3]. As a result, the biased update cannot be the gradient of any objective function. □

The above result suggests that we cannot use the usual optimization analysis, such as setting the gradient equal to zero or computing second order derivatives, to study the properties of this biased update. Therefore, we instead try to understand its behavior by looking at its fixed point. It is now essential to introduce a few more terms. Let $r_1 \leq r_2 \leq \cdots \leq r_k$ represent an ordering of the true rewards for the bandit problem. Let $\pi_t = e^{\boldsymbol{\theta}^{(t)}} / \mathbf{1}^\top e^{\boldsymbol{\theta}^{(t)}}$ represent the softmax policy at timestep t. Here, $\boldsymbol{\theta}^{(t)}$ represents the action preference vector at timestep t; which is updated using the rule $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \alpha\, \mathbb{E}[\hat{\mathbf{g}}^{\,\mathrm{ALT}}(A, R)]$ for $\alpha > 0$.

Lemma 8.1. *The fixed points of the biased gradient bandit update can be categorized as follows: **(1)** If there exists an $n \in \{1, 2, \ldots, k-1\}$ such that $r_1 \leq \cdots \leq r_n < b < r_{n+1} \leq \cdots \leq r_k$, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{\,ALT}(A, R)]$ is never equal to zero and has no fixed point. **(2)** If $b = r(a)$ for at least one action, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{\,ALT}(A, R)] = 0$ at any point on the face of the probability simplex given by $\sum_{a \in \mathcal{I}_b} \pi(a) = 1$ with $\mathcal{I}_b = \{a | r(a) = b\}$. **(3)** If $b < r_1$ or $b > r_k$, then $\mathbb{E}_\pi[\hat{\mathbf{g}}^{\,ALT}(A, R)] = 0$ at a point π^* within the simplex boundary given by $\pi^*(a) = \frac{1}{r(a) - b} \left(\sum_{c \in \mathcal{A}} \frac{1}{r(c) - b} \right)^{-1}$, $\forall a \in \mathcal{A}$.*

Proof. Let $\mathbf{g} := \mathbb{E}[\hat{\mathbf{g}}^{\,\mathrm{ALT}}(A, R)]$, and recall that we update the policy parameters using $\boldsymbol{\theta}^{\mathrm{new}} = \boldsymbol{\theta}^{\mathrm{old}} + \alpha\, \mathbf{g}$, and then find the policy by applying the softmax function (Eq. 2.11). Now, at the fixed

[3] In contrast, for the true gradient bandit update $\nabla_{\boldsymbol{\theta}} \mathcal{J} = \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi)$,

$$\frac{\partial^2 \mathcal{J}}{\partial \theta_c \partial \theta_a} = -\pi(a)\pi(c)\big[r(a) + r(c) - 2r_\pi\big] + \pi(a)\mathbb{I}(a = c)\big[r(a) - r_\pi\big] \quad \Rightarrow \quad \frac{\partial^2 \mathcal{J}}{\partial \theta_c \partial \theta_a} = \frac{\partial^2 \mathcal{J}}{\partial \theta_a \partial \theta_c}, \quad \text{for } a \neq c.$$

point, the policy should stop changing, i.e. $\boldsymbol{\pi}^{\text{old}} = \boldsymbol{\pi}^{\text{new}}$. This means that for each action $a \in \mathcal{A}$,

$$\frac{e^{\theta_a^{\text{old}}}}{\sum_b e^{\theta_b^{\text{old}}}} = \frac{e^{\theta_a^{\text{new}}}}{\sum_b e^{\theta_b^{\text{new}}}} = \frac{e^{\theta_a^{\text{old}}+\alpha g_a}}{\sum_b e^{\theta_b^{\text{old}}+\alpha g_b}} = \frac{e^{\theta_a^{\text{old}}} \cdot e^{\alpha g_a}}{\sum_b e^{\theta_b^{\text{old}}+\alpha g_b}} \quad \Leftrightarrow \quad g_a = \frac{1}{\alpha} \log \frac{\sum_b e^{\theta_b^{\text{old}}+\alpha g_b}}{\sum_b e^{\theta_b^{\text{old}}}} =: \kappa, \tag{3.17}$$

where we introduced κ, a constant independent of the action a. Therefore, at the fixed point of the gradient bandit update, each element of the expected gradient is equal to the constant κ.

Let us use this knowledge to solve for the policy at the fixed point of the biased PG update $\mathbf{g} = \boldsymbol{\pi} \odot (\mathbf{r} - b\mathbf{1})$. We have the following system of $|\mathcal{A}| + 1$ linear equations in $|\mathcal{A}| + 1$ variables:

$$\pi(a) \cdot (r(a) - b) = \kappa, \ \forall a \in \mathcal{A} \qquad \text{and} \qquad \sum_{a \in \mathcal{A}} \pi(a) = 1. \tag{3.18}$$

We now consider the three cases given in the lemma statement separately:

Case 1 (b lies between the expected rewards): We know that there exist two actions a_n and a_{n+1} such that $r(a_n) = r_n < b < r_{n+1} = r(a_{n+1})$, then

$$\pi(a_n) \cdot (r(a_n) - b) < 0 \qquad \text{and} \qquad \pi(a_{n+1}) \cdot (r(a_{n+1}) - b) > 0.$$

Consequently, for no value of κ, Eqs. 3.18 could be satisfied, and those equations do not possess any fixed point.

Case 2 ($r(a) = b$ for atleast one action): If the expected reward for any action a is equal to the baseline b, then $\kappa = 0$, and the fixed point solution corresponds to the face of the simplex where the expected reward for each of the actions is equal to b:

$$\sum_{a \in \mathcal{I}_b} \pi(a) = 1, \quad \text{with} \quad \mathcal{I}_b = \{a \mid r(a) = b\}. \tag{3.19}$$

Case 3 (b is either less than or greater than all of the expected rewards): We assume that either $b > r(a)$ or $b < r(a)$ for all actions $a \in \mathcal{A}$. Now we solve for κ: From Eqs. 3.18, $\pi(a) = \frac{\kappa}{r(a)-b}$, $\forall a \in \mathcal{A}$, and therefore

$$\sum_a \pi(a) = 1 \quad \Rightarrow \quad \sum_a \frac{\kappa}{r(a) - b} = 1 \quad \Rightarrow \quad \kappa = \left(\sum_a \frac{1}{r(a) - b} \right)^{-1}.$$

Therefore, the fixed point of the biased update is given by

$$\pi(a) = \frac{1}{r(a) - b} \left(\sum_{c \in \mathcal{A}} \frac{1}{r(c) - b} \right)^{-1} \quad \text{for all actions } a \in \mathcal{A}. \tag{3.20}$$

This completes the proof[4]. □

Before moving forward, we state a result (Lemma 7.10, Lafferty et al., 2008) which will be useful in proving the next theorem.

Lemma 8.2. *Let X be a bounded random variable with bounds u and l, i.e. $l \leq X \leq u$. Then*

$$\mathbb{E}\left[e^X\right] \leq e^{(u-l)^2/8+\mathbb{E}[X]}.$$

To keep the analysis simple, we assume that for all actions a, $b > r(a)$ or $b < r(a)$. Then, depending on how b is set, the fixed point π^* can act either as an attractor or a repellor.

Theorem 1. *Assume that $\pi_t \neq \pi^*$. If the baseline is pessimistic, i.e. $b < r_1$, then for any $\alpha > 0$, the fixed point π^* acts as a repellor: $D_{KL}(\pi^*\|\pi_{t+1}) > D_{KL}(\pi^*\|\pi_t)$, where D_{KL} denotes the KL-divergence. And if the baseline is optimistic, i.e. $b > r_k$, then given a sufficiently small positive stepsize α, the fixed point π^* acts as an attractor: $D_{KL}(\pi^*\|\pi_{t+1}) < D_{KL}(\pi^*\|\pi_t)$.*

Proof. We first express π_{t+1} in terms of π_t. If we define

$$\zeta(a) := \alpha[r(a) - b], \tag{3.21}$$

then for each action a, the action preference would be updated using $\boldsymbol{\theta}_a^{(t+1)} = \boldsymbol{\theta}_a^{(t)} + \pi_t(a)\zeta(a)$. Applying the softmax function on these updated action preferences gives us π_{t+1}:

$$\pi_{t+1}(a) = \frac{e^{\boldsymbol{\theta}_a^{(t)} + \pi_t(a)\zeta(a)}}{\sum_{c\in\mathcal{A}} e^{\boldsymbol{\theta}_c^{(t)} + \pi_t(c)\zeta(c)}} = \frac{e^{\boldsymbol{\theta}_a^{(t)}} \cdot e^{\pi_t(a)\zeta(a)}}{\sum_c e^{\boldsymbol{\theta}_c^{(t)}} \cdot e^{\pi_t(c)\zeta(c)}} = \frac{\pi_t(a) \cdot e^{\pi_t(a)\zeta(a)}}{\sum_c \pi_t(c) \cdot e^{\pi_t(c)\zeta(c)}}. \tag{3.22}$$

Let us now compute the KL-divergence between the fixed point π^* and the policy at timestep $t+1$:

$$\begin{aligned} D_{\mathrm{KL}}(\pi^*\|\pi_{t+1}) &:= \sum_{a\in\mathcal{A}} \pi^*(a) \log \frac{\pi^*(a)}{\pi_{t+1}(a)} = \sum_a \pi^*(a) \log \left(\frac{\pi^*(a)}{\pi_t(a)} \cdot \frac{\sum_c \pi_t(c) \cdot e^{\pi_t(c)\zeta(c)}}{e^{\pi_t(a)\zeta(a)}} \right) \\ &= \sum_a \pi^*(a) \log \frac{\pi^*(a)}{\pi_t(a)} + \sum_a \pi^*(a) \log \left(\sum_c \pi_t(c) \cdot e^{\pi_t(c)\zeta(c)} \right) - \sum_a \pi^*(a)\pi_t(a)\zeta(a) \\ &= D_{\mathrm{KL}}(\pi^*\|\pi_t) + \log \left(\sum_c \pi_t(c) \cdot e^{\pi_t(c)\zeta(c)} \right) - \left(\sum_c \frac{1}{\zeta(c)} \right)^{-1}, \end{aligned} \tag{3.23}$$

[4]Interestingly, using this style of reasoning to analyze the true expected gradient bandit would not give us any new information. Even if we assume that $g_a = \pi(a) \cdot (r(a) - b) = \kappa$, $\forall a \in \mathcal{A}$, we find that κ is always zero for the true gradient bandit update. To see this, sum these $|\mathcal{A}|$ equations to get

$$\sum_a \pi(a) \cdot (r(a) - r_\pi) = |\mathcal{A}|\kappa \quad \Rightarrow \quad \sum_a \pi(a) r(a) - r_\pi \sum_a \pi(a) = |\mathcal{A}|\kappa \quad \Rightarrow \quad \kappa = 0.$$

where in the first line we used the expression for π_{t+1} from Eq. 3.22 and in the last line we used the expression $\pi^*(a) = \frac{1}{r(a)-b}\left(\sum_{c\in\mathcal{A}} \frac{1}{r(c)-b}\right)$ from Eq. 3.20, to simplify the last term. We now consider the two cases (corresponding to an optimistic and a pessimistic baseline) separately.

Case 1 (Pessimistic baseline): With the assumption that $\alpha > 0$ and $b < r(a)$, we immediately see that $\zeta(a) > 0$ for all actions a. Noting that $\zeta(a) > 0$, we apply Cauchy-Schwartz inequality:

$$\left(\sum_c \pi_t(c)^2\zeta(c)\right)\left(\sum_c \frac{1}{\zeta(c)}\right) > \left(\sum_c \pi_t(c)\sqrt{\zeta(c)}\cdot\frac{1}{\sqrt{\zeta(c)}}\right)^2 = 1$$

$$\Rightarrow \qquad \sum_c \pi_t(c)^2\zeta(c) > \left(\sum_c \frac{1}{\zeta(c)}\right)^{-1}, \tag{3.24}$$

where we could use a strict inequality in the first line because for no constant ρ does $\frac{\pi_t(a)}{\zeta(a)} = \rho$ for all actions a; if this were true, it would imply that $\pi_t = \pi^*$, which in turn would violate the assumption made in the theorem statement. Next, recall that $\log(\cdot)$ is a concave function and then apply Jensen's inequality followed by the previous result to get:

$$\log\left(\sum_c \pi_t(c)\cdot e^{\pi_t(c)\zeta(c)}\right) > \sum_c \pi_t(c)\log\left(e^{\pi_t(c)\zeta(c)}\right) = \sum_c \pi_t(c)^2\zeta(c) > \left(\sum_c \frac{1}{\zeta(c)}\right)^{-1}. \tag{3.25}$$

Above inequality combined with Eq. 3.23 gives us the desired result:

$$D_{\mathrm{KL}}(\pi^*\|\pi_{t+1}) > D_{\mathrm{KL}}(\pi^*\|\pi_t).$$

Case 2 (Optimistic baseline): In contrast to the previous case, the assumptions $\alpha > 0$ and $b > r(a)$, imply that $\zeta(a) < 0$ for all actions a. We will first use Lemma 8.2 to upper bound the term $\log\left(\sum_c \pi_t(c)\cdot e^{\pi_t(c)\zeta(c)}\right)$ and then obtain the condition on α (as given in Eq. 3.29) to show that the fixed point is an attractor. Consider a random variable $X : \mathcal{A} \to \mathbb{R}$ defined as $X(c) = \pi_t(c)\zeta(c)$ and distributed according to π_t, i.e. $\mathbb{P}(X = c) = \pi_t(c)$. Then $\mathbb{E}[X] = \sum_c \pi(c)^2\zeta(c)$ and X is bounded between $l := \alpha \min_c \pi_t(c)[r(c)-b]$ and $u := \alpha \max_c \pi_t(c)[r(c)-b]$. Using Lemma 8.2 along with the fact that $\log(\cdot)$ is an increasing function gives us:

$$\log\left(\sum_c \pi_t(c)e^{\pi_t(c)\zeta(c)}\right) \le \frac{(u-l)^2}{8} + \sum_c \pi(c)^2\zeta(c). \tag{3.26}$$

Now the question is whether there exists a stepsize $\alpha > 0$, such that the following inequality holds:

$$\frac{(u-l)^2}{8} + \sum_c \pi(c)^2\zeta(c) \overset{?}{<} \left(\sum_c \frac{1}{\zeta(c)}\right)^{-1} \tag{3.27}$$

And the answer is yes! To see this, again apply Cauchy-Schwartz inequality, only this time for $\zeta(c) < 0$, to obtain $\sum_c \pi_t(c)^2\zeta(c) < \left(\sum_c \frac{1}{\zeta(c)}\right)^{-1}$. Therefore, there does exist an $\alpha > 0$ that satisfies Ineq. 3.27. With Eq. 3.23, Ineq. 3.26, and an appropriate condition on α, the intended result that π^* is an attractor[5] readily follows:

$$\log\left(\sum_c \pi_t(c)e^{\pi_t(c)\zeta(c)}\right) \leq \frac{(u-l)^2}{8} + \sum_c \pi(c)^2\zeta(c) < \left(\sum_c \frac{1}{\zeta(c)}\right)^{-1}$$
$$\Rightarrow \qquad D_{\mathrm{KL}}(\pi^* \| \pi_{t+1}) < D_{\mathrm{KL}}(\pi^* \| \pi_t).$$

All that remains is to find the expression for α that satisfies Ineq. 3.27:

$$\alpha < \frac{8}{\left(\max_c \pi_t(c)[r(c)-b] - \min_d \pi_t(d)[r(d)-b]\right)^2}\left[\left(\sum_c \frac{1}{r(c)-b}\right)^{-1} - \sum_c \pi_t(c)^2[r(c)-b]\right]. \tag{3.29}$$

Note that the above upper bound on α depends on the policy at timestep t, the choice of the baseline, and also the complete bandit reward structure (which is not known to the agent a priori). □

We present an example in Figure 3.1, which shows that the upper bound on the stepsize, given in Eq. 3.29, is not tight. This figure also serves as a verification of the above proof.

The above theorem illustrates an important property of the alternate estimator: with an optimistically initialized baseline, the agent is updated towards a more uniform distribution π^*. For an optimistic baseline, this distribution has a higher probability of picking the action with the maximum reward as compared to other actions; this can be seen from the expression for π^* in Lemma 8.1 or Figure 3.6 (bottom row, second column). Therefore, if the agent were stuck in a sub-optimal corner of the probability simplex, an optimistic baseline would make the policy more uniform and encourage exploration[6]. And even though π^* is different from the optimal policy, the agent with

[5]Theorem 1 only shows where the biased update converges to for an optimistic baseline; an analogous result for a pessimistic baseline is missing. We believe that given a pessimistic baseline, the biased update will converge to the policy corner which is closest (in some metric) to the initial policy, as suggested by Figure 3.6 (bottom-right). The metric used to identify this corner, where the the policy would converge to, **might** look something like this:

$$c = \arg\max_{a \in \mathcal{A}} [\pi_t(a) - \pi^*(a)] = \arg\max_{a \in \mathcal{A}} \left[\pi_t(a) - \frac{1}{r(a)-b}\left(\sum_d \frac{1}{r(d)-b}\right)^{-1}\right], \tag{3.28}$$

where c denotes the action corresponding to the concerned corner. Having identified this corner, the next step would be to show that for any $\alpha > 0$, first, the following holds true

$$D_{\mathrm{KL}}(\mathbf{e}_c \| \pi_{t+1}) < D_{\mathrm{KL}}(\mathbf{e}_c \| \pi_t) \quad \Leftrightarrow \quad \sum_a \pi_t(a)e^{\pi_t(a)\zeta(a)} < e^{\pi_t(c)\zeta(c)},$$

where $\zeta(c)$ is as defined in Eq. 3.21, and second, that the condition on c (given by Eq. 3.28) remains true after the policy is updated. However, at this stage, all this is a mere conjecture.

[6]On the flip side, with a pessimistically initialized baseline, the alternate estimator can pre-maturely saturate towards a sub-optimal corner; see Figure 3.6 (bottom right).

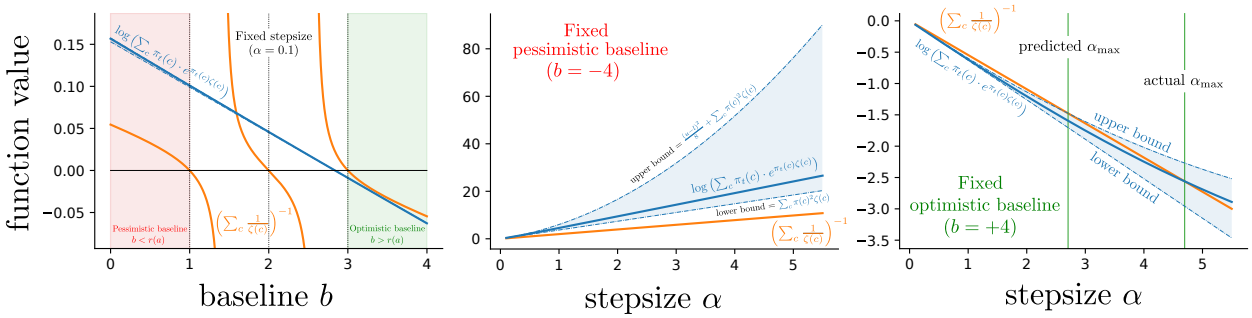

Figure 3.1: The verification of the inequalities used in Theorem 1. We plot the key terms used in the proof: $\left(\sum_c \frac{1}{\zeta(c)}\right)^{-1}$, $\log\left(\sum_c \pi_t(c) \cdot e^{\pi_t(c)\zeta(c)}\right)$, its lower bound $\sum_c \pi(c)^2\zeta(c)$, and its upper bound $\frac{(u-l)^2}{8} + \sum_c \pi(c)^2\zeta(c)$, for a three armed bandit problem with $\mathbf{r} = [1\ 2\ 3]^\top$ and the fixed policy $\boldsymbol{\pi} = [0.2\ 0.1\ 0.7]^\top$. The left figure shows the variation of the key terms with a changing baseline and a fixed stepsize α. In the middle and the right plots, we fix the baseline to be pessimistic or optimistic and then vary the stepsize α. The rightmost subplot also serves as the concrete example where the maximum stepsize predicted by Eq. 3.29, for the fixed point to remain attractive given an optimistic baseline, is not tight.

an alternate estimator can still reach the optimal policy, because as the agent learns and improves its baseline estimate, the alternate PG estimator becomes asymptotically unbiased. And hopefully by this time, the agent has already escaped the saturated policy region.

One might wonder: but why the biased update is more helpful than the true gradient; after all the true gradient, unlike the biased update, doesn't just move the policy towards a point close to the optimal corner, it updates it exactly towards the optimal corner. The reason for this is that the true gradient update can be vanishingly small in saturated regions and therefore take an excessive amount of time to escape saturation (Mei et al., 2020b; Li et al., 2021). Whereas, as we hypothesize, the biased update will have a much larger update magnitude and therefore result in a faster escape. Even though, we don't theoretically show this, Example 2, the policy update plots shown in Figure 3.6 (bottom row, second column), and our experiments on bandits with an optimistic baseline initializations (see §4.3) do suggest this.

3.5 Visualizing the Stochastic PG Estimators for Bandits

In this section, we study the properties of the **stochastic** regular and the alternate estimators. In the previous sections we primarily looked at the expected policy updates, and now we will focus on the sample based updates. We do so by plotting the stochastic policy gradient updates and the variance corresponding to the two estimators for different 3-armed bandit problems and

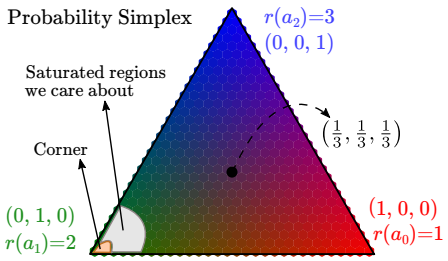

Figure 3.2: The probability simplex for a three armed bandit problem. Each point on the simplex corresponds to a policy. The color map shown is representative of the policy weight for the corresponding action. For instance, in regions that are more blue, the agent's policy has large probability mass on the the action a_2. The reward structure for this problem is $\mathbf{r} = [1\ 2\ 3]^\top$ with the individual components corresponding to the expected reward on the right, left, and the top corners respectively. We also highlight one corner of the simplex, and the saturated policy region near that corner. In our experiments, we focus on these regions neighbouring the corners, i.e. we focus on policies that are quite saturated but still have non-zero components for all the actions.

various policies on the probability simplex (see Figure 3.2). These visualizations also illustrate the properties discussed in the previous sections.

The policy-update plots were created by drawing a vector from the initial policy to the updated policy after one gradient ascent step using the corresponding PG estimator with a fixed stepsize[7]. For each estimator, we show four different plots: three for the stochastic estimator, each corresponding to one of the sampled actions (a_0, a_1, or a_2); and the fourth one corresponding to the updates made by expected gradient (which itself can be the true policy gradient or the biased one). While calculating the estimator, we also added a noise of constant (unit) magnitude to the expected reward. The blue and grey arrows correspond to the policy change direction for positive and negative reward noises respectively. For clarity, let us illustrate this process with an example. Consider the bandit problem with $\mathcal{A} = \{a_0, a_1, a_2\}$, the reward structure $\mathbf{r} = [0\ 0\ 1]^\top$, and a positive reward noise. Let the old policy be $\boldsymbol{\pi}^{\text{old}} = [1/3\ 1/3\ 1/3]^\top$ which will be updated using the stochastic regular estimator without baseline with the sampled action being a_2 and a stepsize of $\alpha = 0.4$. The action preferences corresponding to this policy can be computed by taking $\log(\cdot)$ of the action probabilities. This gives us $\boldsymbol{\theta}^{\text{old}} = [\log(1/3)\ \log(1/3)\ \log(1/3)]^\top$. Next, we calculate the gradient update using the regular estimator without the baseline: $\hat{\mathbf{g}}(R, A) = (R - 0) \cdot (\mathbf{e}_A - \boldsymbol{\pi}) = (r(a_2) + 1 - 0) \cdot ([0\ 0\ 1]^\top - [1/3\ 1/3\ 1/3]^\top) = 2 \cdot [-1/3\ -1/3\ 2/3]^\top$. We then update the action preferences: $\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \alpha\, \hat{\mathbf{g}}(R, A)$ and in the end take the softmax of these action preferences to obtain the updated policy $\boldsymbol{\pi}^{\text{new}}$. Finally, we plot a blue colored vector (since a positive noise was added to the reward) from $\boldsymbol{\pi}^{\text{old}}$ to $\boldsymbol{\pi}^{\text{new}}$ on the probability simplex. We repeat this process for different initial

[7]We chose stepsizes that resulted in a visually appropriate arrow length. To make the arrow lengths comparable across the estimators, the stepsize was kept same for the regular and the alternate estimators in the same setting (such as for the same choice of the baseline or the bandit reward structure).

policies to obtain the final plots (for instance, see Figure 3.3).

The variance plots were created by plotting the elementwise variance for the different stochastic PG estimators on the probability simplex. Each hexagonal point on the simplex corresponds to a particular policy value and the color specifies the variance of a particular component of the PG estimator. The variance was calculated using Eq. 3.15. The top three rows show the variance for the three components and the last row shows the sum of variance along the three components.

Bandit Problem with $\mathbf{r} = [0\ 0\ 1]^\top$ and Non-zero Reward Noise

Figure 3.3 shows the policy update plots and the variance heatmaps for the two estimators on a 3-armed bandit problem with the reward structure $\mathbf{r} = [0\ 0\ 1]^\top$ and reward noise $\epsilon \sim \mathcal{N}(0, 1)$. Each column corresponds to the updates and the variance for either the regular estimator with true r_π, the regular estimator without baseline ($b = 0$), the alternate with true r_π, or the alternate estimator without baseline ($b = 0$). We include the zero baseline case because in our experiments, when we learn the baseline, we often initialize it to zero.

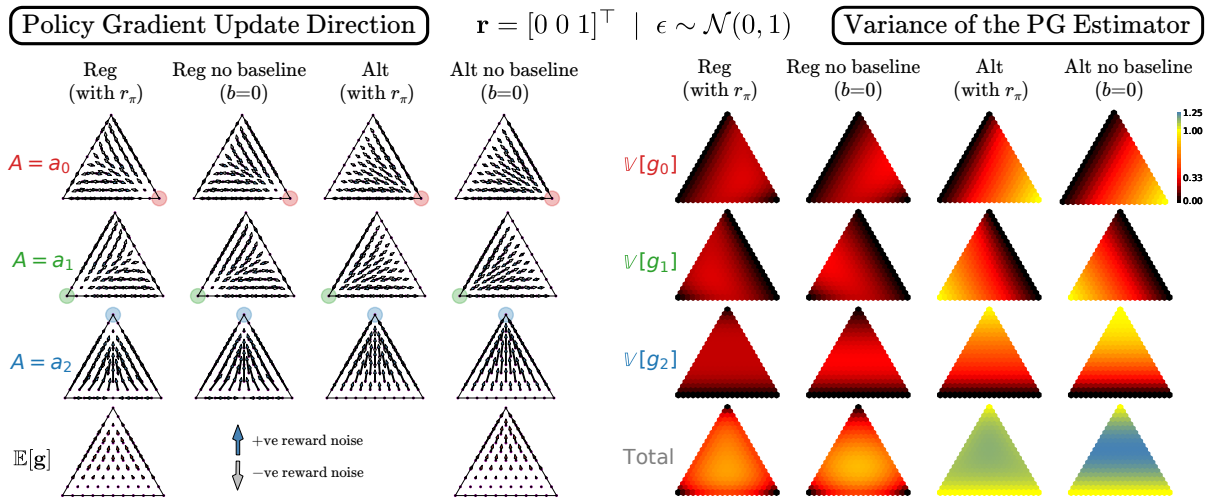

Figure 3.3: Policy updates and the variance plots for the PG estimators on a 3-armed bandit problem with $\mathbf{r} = [0\ 0\ 1]^\top$ and a non-zero reward noise. **(Update Plots)** The top three rows show the policy updates corresponding to the different stochastic estimators upon taking a particular action. Last row shows the expected update. Since, the alternate estimator with true r_π and both the regular estimators result in the same expected update, we show only a single plot for these three settings. **(Variance Plots)** The first three rows depict the respective component of the variance vector (of the different gradient estimators) and the last row shows the total variance.

From the update plots, we see that the stochastic alternate estimator has much higher update magnitudes than the regular version. For instance, consider a policy saturated say at action a_0, i.e. it is near the vicinity of the right corner. With the alternate gradient estimator, there is a chance

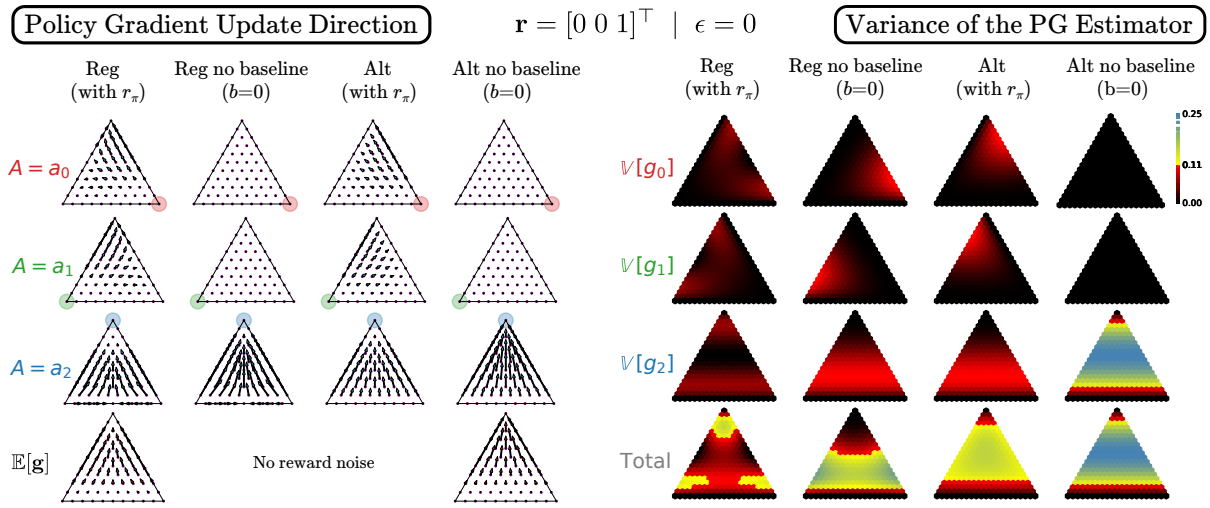

Figure 3.4: Policy updates and the variance for PG estimators on a 3-armed bandit problem with $\mathbf{r} = [0\ 0\ 1]^\top$ and no reward noise.

that the policy will move towards the center of the simplex, i.e. it will be updated towards a more uniform distribution. Whereas, with the regular estimator, the agent will likely remain stuck. This observation is also reconciled by the expressions for the estimators: the term $\mathbf{e}_A - \boldsymbol{\pi}$ in the regular estimator becomes very small as $\boldsymbol{\pi}$ saturates on action A; whereas the alternate estimator only has the term $\mathbf{e}_A$ and its magnitude remains the same irrespective of the saturation. As discussed earlier, the alternate estimator with no baseline is biased. However, its ascent direction points in the same direction as the true gradient update. From the variance plots, we notice that the variance at the corner for both the regular estimators is zero, and for the alternate estimators is non-zero.

So near a corner of the simplex, the alternate estimator utilizes the reward noise to essentially perform a random walk. At sub-optimal corners, this random movement is good since it might make the policy more uniform, from where it is easier to move towards the optimal corner. And at the optimal corners, this random movement is not particularly bad either because optimal corners represent stable equilibrium points: in expectation the gradient signal towards them.

Bandit Problem with $\mathbf{r} = [0\ 0\ 1]^\top$ and Zero Reward Noise

Now we study how these estimators behave when we take away the noise from the rewards in the bandit problem. Figure 3.4 shows how the policy is updated and the variance of the gradient estimators for a bandit problem with no reward noise. From these plots, we see that the benefit of the alternate estimator (large update magnitudes at the corners) disappears. With neither any reward signal nor any noise, the alternate estimator cannot escape the corners (see the experiments in §4.1). However, the alternate estimator still has a higher update magnitude as compared to the

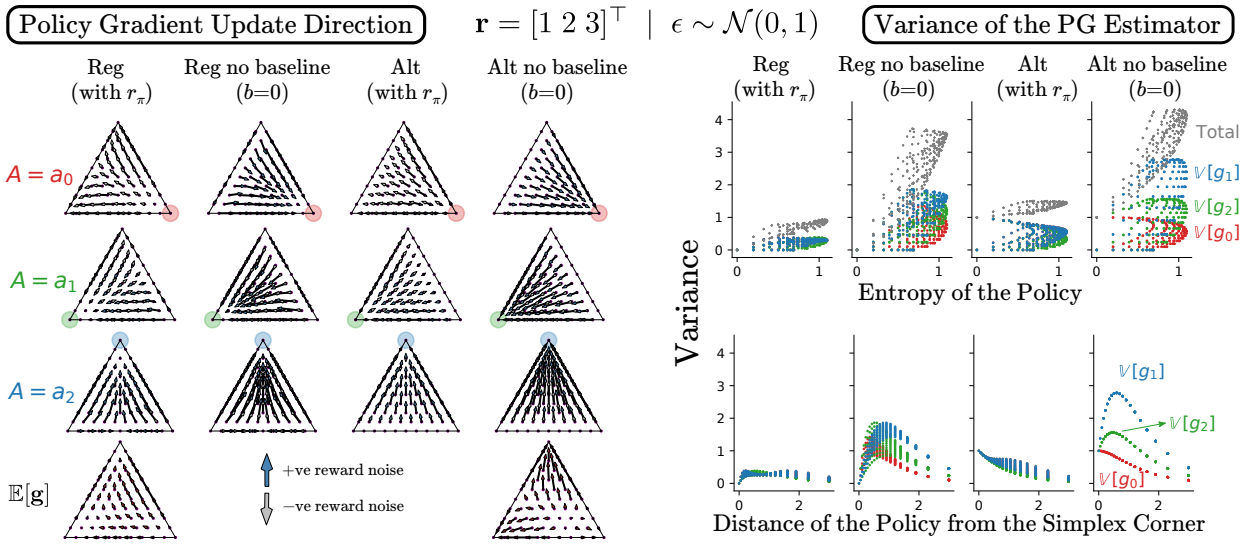

Figure 3.5: Policy updates and the entropy plots for PG estimators on a 3-armed bandit problem with $\mathbf{r} = [1\ 2\ 3]^\top$ and a non-zero reward noise.

regular estimator near the corners. From the variance plots, we see that, as shown by Proposition 7, the variance for all the estimators is zero at the corners; no reward noise means no variance.

Bandit Problem with $\mathbf{r} = [1\ 2\ 3]^\top$ and Non-zero Reward Noise

In the previous settings, there was no reward signal at the sub-optimal corners. Now we consider a case in which there is a non-zero reward signal at the sub-optimal corners and the noise is comparable to it. From Figure 3.5 (left), we see that the alternate estimator (with true r_π) works better as compared to the regular estimator: it has high noise at sub-optimal corners which might make the policy more uniform. However, with $b = 0$, this is no longer true. In fact, with the alternate estimator, we are driven into the sub-optimal corner faster than the regular without the baseline. Since $b = 0$ is pessimistic for the reward structure $\mathbf{r} = [1\ 2\ 3]^\top$, the policy will move away from the center of the simplex (Theorem 1). So initially, the alternate estimator will drive the policy into the corner. Then as its baseline estimate improves and becomes closer to r_π, there is some chance that the agent will escape saturation (although our experiments in Figure 4.7 show that the alternate estimator with a pessimistic baseline is often inferior to the regular estimator).

In Figure 3.5 (right), the top row shows the variance of the policy against its entropy, and the bottom row shows the variance of the particular estimator against the policy's distance (measured using KL divergence) from that corner, i.e. for each action a, we plot $\mathbb{V}[g_a]$ against $D_{\mathrm{KL}}(\mathbf{e}_a \,\|\pi)$. The results are similar as the previous variance plots. These entropy figures, unlike the variance heatmaps, can also be plotted for bandit problems with more than three actions.

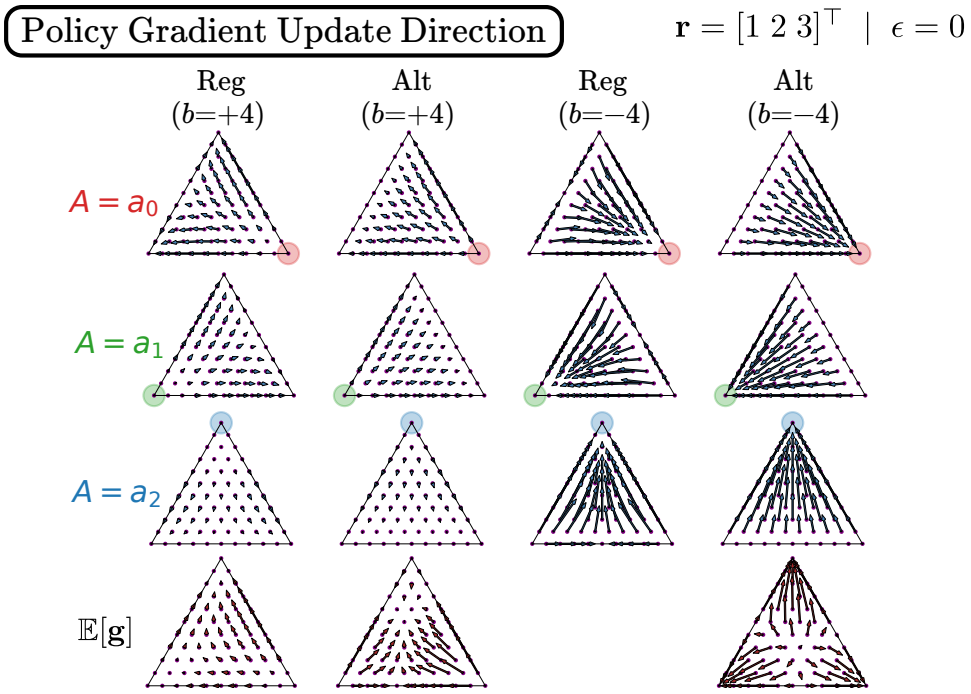

Figure 3.6: Policy updates for the bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ and no reward noise. The regular and the alternate estimators use an optimistic ($b = +4$) or pessimistic ($b = -4$) baseline. As discussed in §3.4, both the regular estimators are unbiased, whereas both the alternate estimators are biased.

Bandit Problem with $\mathbf{r} = [1\ 2\ 3]^\top$ and an Optimistic or a Pessimistic Baseline

In Figure 3.6, we show the policy update plots for optimistic or pessimistic baselines. We fix the reward structure to $\mathbf{r} = [1\ 2\ 3]^\top$ and do not add any reward noise (the results with non-zero reward noise were effectively similar to these ones, and therefore we skipped them). For an optimistic baseline, we see that both the regular and the alternate stochastic estimators make the policy more uniform. The alternate estimator again has higher update magnitudes compared to the regular estimator. The last row illustrates the point made by Theorem 1, that the alternate estimator with an optimistic baseline attracts the policy to a point on the simplex. Further, this attractive signal, as seen from the figure, is much stronger than the expected gradient signal offered by the regular estimator. The expected regular estimator moves the policy towards the optimal corner albeit with very small update magnitudes, especially at the corner. The trends are similar for the pessimistic baseline, except that the policy is updated aggressively towards a corner. The variance plots for this settings looked very similar to those from the previous settings, and therefore we did not include them.

3.6 Per-timestep Computational Complexity of Gradient Bandits

We now show that the per-timestep cost of a discrete gradient bandit algorithm using the alternate estimator has a weaker dependence on the number of actions than the regular estimator. The gradient bandit algorithms involves multiple operations during each timestep. At the start of learning, the agent has to initialize the action preferences. Then at each timestep, it needs to compute the gradient estimate, update the action preferences and the policy, and sample an action from the policy. The sampling process itself requires the agent to maintain a sampling data structure which also needs to be updated each time the policy changes. The total per-timestep computational cost of the algorithm depends on all of these operations, which we summarize in Table 3.1.

Computational Process	**Regular Estimator + Vose's Alias Sampling**	**Alternate Estimator + Tree Sampling**
Initialize $\boldsymbol{\theta}$	$\mathcal{O}(\lvert\mathcal{A}\rvert)$	$\mathcal{O}(\lvert\mathcal{A}\rvert)$
Compute $\hat{\mathbf{g}}$ and update $\boldsymbol{\theta}$	$\mathcal{O}(\lvert\mathcal{A}\rvert)$	$\mathcal{O}(1)$
Compute $\boldsymbol{\pi}$	$\mathcal{O}(\lvert\mathcal{A}\rvert)$	–
Update the sampling data structure	$\mathcal{O}(\lvert\mathcal{A}\rvert)$	$\mathcal{O}(\log \lvert\mathcal{A}\rvert)$
Sample $A \sim \pi$	$\mathcal{O}(1)$	$\mathcal{O}(\log \lvert\mathcal{A}\rvert)$
Total cost	$\mathcal{O}(\lvert\mathcal{A}\rvert + n_{\text{iter}}\lvert\mathcal{A}\rvert)$	$\mathcal{O}(\lvert\mathcal{A}\rvert + n_{\text{iter}} \log \lvert\mathcal{A}\rvert)$

Table 3.1: The key steps involved, along with their respective computational cost, while running the gradient bandit algorithm with the regular or the alternate estimator. We denote the total number of iterations by n_{iter}. Also, it is reasonable to assume that $n_{\text{iter}} \gg |\mathcal{A}|$.

The regular estimator updates each component of the action preference vector $\boldsymbol{\theta}$ at each timestep, which has a complexity that is linear in the number of actions. Let us assume, for illustration purposes, that the agent with the regular estimator uses Vose's alias sampling method[8] for obtaining actions from the policy. As a result, the agent needs to build an *alias table* which takes a linear time. Once this table is built, the agent can then sample an action in constant time from this table. However, updating the alias table each time has a linear cost as well. Therefore, the total per-timestep cost for this agent becomes $\mathcal{O}(|\mathcal{A}|)$. In contrast, the alternate estimator updates only a single component of the action preference vector at each timestep. Because of this, a gradient bandit instance using the alternate estimator can be sped up to have a total per-time-step complexity of $\mathcal{O}(\log |\mathcal{A}|)$. To achieve this, we employ a sampling procedure based on a balanced tree data-structure, which we call as the *sampling tree*. Although, the sampling tree method needs logarithmic time for sampling a single action, it can be updated in logarithmic time as well if only a single action preference changes at each timestep (as opposed to the alias method which would require a linear time even if only a single action preference were to change).

[8]For a reference on different sampling methods, see `https://www.keithschwarz.com/darts-dice-coins/`.

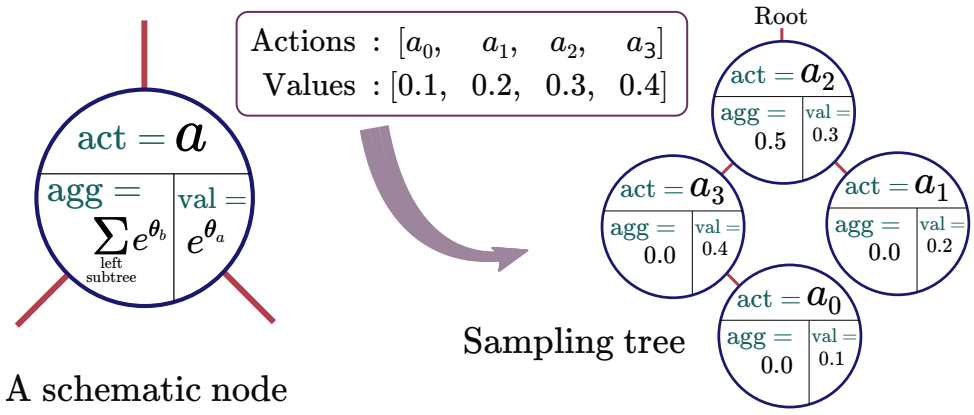

Figure 3.7: A typical sampling tree node and one possible tree for a softmax policy with four actions. The values refer to exponentiated action preference e^{θ_a} for that particular action.

Using a Balanced Binary Tree for Sampling Actions

The sampling tree is a balanced binary tree (for a reference on our notation, see Chapter 12, Cormen et al., 2009). It has one node corresponding to each action that stores a reference and the exponentiated action preference corresponding to that action. It also stores the sum of all the exponentiated action preferences corresponding to the actions lying in the left subtree of this node. Figure 3.7 shows a schematic diagram of such a node, and one possible sampling tree[9]. We now discuss how the agent uses this data structure for sampling actions.

The cost for building the sampling tree for the first time is linear in the number of actions. The agent would need to initialize the action preferences, create a balanced tree, and then calculate the aggregated exponentiated preferences of actions in the left-subtree (using Algorithm 4). And all of these operations are clearly linear in the size of the action space.

Algorithm 4 Calculate Left Sub-tree Aggregate

```
Call COMPUTE-AGGREGATE(sampling tree, head node of the tree)
 # function definition
function COMPUTE-AGGREGATE(T, z)
    if z = T.nil then # if z is a leaf node
        return 0
    L ← COMPUTE-AGGREGATE(T, z.left)
    R ← COMPUTE-AGGREGATE(T, z.right)
    z.agg ← L
    return L + z.val + R
```

[9]The sampling tree is not necessarily a search tree. However, one can apply procedures similar to those discussed here without changing the asymptotic computational complexity, to balanced search trees such as the Red-black trees (Chapter 13, Cormen et al., 2009) to obtain a sampling tree that is also a search tree. This might be required if one needs to maintain a list over the actions ordered by their probability; say, to pick the greedy action at each timestep.

The sampling process itself takes logarithmic time. To sample an action from the policy, the agent first samples a key from a uniform distribution[10] over the interval $\left[0, \sum_{i=1}^{|\mathcal{A}|} e^{\theta_i}\right)$, and then calls the ACTION-SELECT function (see Algorithm 5) on the sampling tree with this key. Note that the ACTION-SELECT function runs in $\log(|\mathcal{A}|)$ time for a balanced sampling tree.

Algorithm 5 Selecting an Action from the Sampling Tree

Generate $x \sim \mathcal{U}\left[0, \sum_{i=1}^{|\mathcal{A}|} e^{\theta_i}\right)$ `# 𝒰 refers to the uniform distribution`
Call ACTION-SELECT(head node of the tree, x)
`# function definition`
function ACTION-SELECT(z, x)
  if $x < z.agg$ **then**
    return ACTION-SELECT($z.left, x$)
  else if $z.agg \leq x < z.agg + z.val$ **then**
    return $z.act$
  else `# this refers to the case when` $x \geq z.agg + z.val$
    $x_{new} \leftarrow x - z.agg - z.val$
    return ACTION-SELECT($z.right, x_{new}$)

The next proposition explains the functioning of the ACTION-SELECT function.

Proposition 9. *Let $T(z_1, z_2, \ldots, z_n)$ denote a sampling tree of size n with nodes $z_1, z_2, \ldots, z_n$ arranged in an inorder traversal (see Chapter 12, Cormen et al., 2009). Also, without loss of generality assume that the node z_i corresponds to the action a_i and stores the value e^{θ_i}. Then given the sampling tree T and a search key $x \in \left[0, \sum_{i=1}^{n} e^{\theta_i}\right)$ as the input, the* ACTION-SELECT *function returns the action $z_k.act = a_k$ such that the index k satisfies the relation $\sum_{i=1}^{k-1} e^{\theta_i} \leq x < \sum_{i=1}^{k} e^{\theta_i}$.*

Proof. We will use induction to prove this statement. First, consider a tree of size one, $T(z_j)$ with $z_j.val = e^{\theta_j}$ and $z_j.agg = 0$. By the assumption on the search key, $x \in \left[0, e^{\theta_j}\right)$, i.e. the second condition in Algorithm 5 is true, and therefore the function returns $z_j.act$. Next, assume that the function does return the intended action for any tree of size $l \leq m$ where m is some arbitrary positive number for an appropriate key x. We now prove the proposition statement for a key $x \in \left[0, \sum_{i=1}^{l+1} e^{\theta_i}\right)$, and an arbitrary tree of size $l+1$, $T(z_1, z_2, \ldots, z_{r-1}, z_r, z_{r+1}, \ldots, z_{l+1})$ where z_r represents the root node and $1 \leq r \leq l+1$. Then, depending on the value of x, we get three cases:

- *Case 1* ($x < z_r.agg = \sum_{i=1}^{r-1} e^{\theta_i}$): the function gets called again on the sub-tree $T(z_1, \ldots, z_{r-1})$, of size $r-1 \leq l$, with the key $x \in \left[0, \sum_{i=1}^{r-1} e^{\theta_i}\right)$.
- *Case 2* ($\sum_{i=1}^{r-1} e^{\theta_i} = z_r.agg \leq x < z_r.agg + z_r.val = \sum_{i=1}^{r} e^{\theta_i}$): the function returns $z_r.act$.

[10] It is standard to assume that sampling a single value from a uniform distribution takes constant time.

- *Case 3* ($x \geq z_r.agg + z_r.val = \sum_{i=1}^{r} e^{\theta_i}$): the function gets called on $T(z_{r+1}, \ldots, z_{l+1})$, a sub-tree of size $l - r + 1 \leq l$, with the modified key $x_{new} \in \left[0, \sum_{i=r+1}^{l+1} e^{\theta_i}\right)$.

This closes the induction, and thus the proposition is true for sampling trees of all sizes. □

The above proposition shows that the action a_k is selected whenever the search key x lies in the range $\left[\sum_{i=1}^{k-1} e^{\theta_i}, \sum_{i=1}^{k} e^{\theta_i}\right)$. However, since $x \sim \mathcal{U}\left[0, \sum_{i=1}^{|\mathcal{A}|} e^{\theta_i}\right)$, the probability of action a_k being chosen is equal to $\frac{e^{\theta_k}}{\sum_{i=1}^{|\mathcal{A}|} e^{\theta_i}}$. Therefore, using Algorithm 5 is indeed equivalent to sampling from a softmax policy with action preferences $\theta_1, \theta_2, \ldots, \theta_{|\mathcal{A}|}$.

When an agent, using the alternate estimator, updates the action preference for the sampled action, it only needs to update the exponentiated preference for that action in the tree and the aggregated values for all nodes lying on the path from this nodes upto the root of the tree. The procedure shown in Algorithm 6 accomplishes this in a logarithmic time. Also note that the structure of the tree remains unchanged during all these operations[11].

Algorithm 6 Fix Aggregates Upstream after an Update

Call FIX-UPSTREAM(sampling tree, node corresponding to the sampled action a, $\theta_a^{\text{new}} - \theta_a^{\text{old}}$)

```
# function definition
function FIX-UPSTREAM(T, z, δ)
    repeat # repeat until node z is the root node
        if z.p.left = z then # update the parent's cumulant if z is the left child
            z.p.agg ← z.p.agg + δ
        z ← z.p # move up the tree
    until z.p ≠ T.nil
```

3.7 Summary

In this chapter, we introduced the alternate policy gradient estimator for bandits. We showed that the alternate estimator is unbiased when the baseline is equal to the average reward for the policy r_π and becomes biased when the baseline is not equal to the expected reward. We illustrated these properties with multiple visualizations on different 3-armed bandit problems. We also demonstrated that the regular estimator has very small update magnitudes in the saturated regions, whereas the alternate estimator can use the reward noise and its biasedness to escape the saturated policy regions. Finally, we showed that the per-timestep computational complexity of a gradient bandit algorithm using the alternate estimator is logarithmic in the number of actions.

[11] We end this section with the following quote from Donald Knuth that is quite appropriate here: "Beware of bugs in the above code; I have only proved it correct, not tried it."

Chapter 4

Experiments on Multi-Armed Bandits

This chapter presents multiple experiments with gradient bandits on different 3-armed bandit tasks. All the bandit problems had the action set $\mathcal{A} = \{a_0, a_1, a_2\}$, a fixed reward structure $\mathbf{r}$, and a normally distributed reward noise ϵ. The experimental results serve as a demonstration and the verification of the properties of the gradient bandits estimators discussed in the previous chapter.

By default, in all the experiments, we trained five different gradient bandit agents with a softmax policy: for different agents, the policy weights were updated using either the expected gradient $\nabla\mathcal{J} = \boldsymbol{\pi} \odot (\mathbf{r} - r_\pi)$, the regular estimator with true average reward $\mathbf{g}^{\text{REG}}(R, A) = (R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi})$ or a learned baseline $\hat{\mathbf{g}}^{\text{REG}}(R, A) = (R - b)(\mathbf{e}_A - \boldsymbol{\pi})$, or the alternate estimator with the true average reward $\mathbf{g}^{\text{ALT}}(R, A) = \mathbf{e}_A(R - r_\pi)$ or a learned baseline $\hat{\mathbf{g}}^{\text{ALT}}(R, A) = \mathbf{e}_A(R - b)$. The baseline in each case was learned using a running average: $b_{t+1} = (1 - \beta)b_t + \beta R_t$, with the stepsize β. The exact pseudocode for gradient bandits is given in Algorithm 1. The action preferences for the softmax policy were initialized to $\theta_{a_1} = \theta_{a_2} = 0$; and $\theta_{a_0} = 0$ for uniform policy initialization or $\theta_{a_0} \in \{5, 10, 50\}$ for saturated policy initialization depending on the degree of saturation.

Each experiment was run for 1000 timesteps. For each estimator, we performed a sweep over the policy stepsize $\alpha \in \{2^{-6}, 2^{-5}, \ldots, 2^1\}$ and the baseline stepsize $\beta \in \{2^{-4}, 2^{-3}, \ldots, 2^0\}$. The sensitivity plots show the mean final performance on the last 50 timesteps averaged over 150 runs for different parameter settings. To reduce clutter, we didn't label all the βs in the sensitivity plots.

4.1 Alternate Estimator can Escape Saturated Policy Regions

Experiments in this section demonstrate that the alternate estimator performs competitively with the regular estimator for uniform policy initialization and clearly outperforms it in the case of a saturated policy initialization when there is reward noise present. They also show that in the

absence of reward noise, the alternate estimator fails to escape the saturated policy regions. The bandit reward structure is $\mathbf{r} = [0\ 0\ 1]^\top$ with either normal noise $\epsilon \sim \mathcal{N}(0, 1)$ or zero reward noise.

Learning Curves

Figure 4.1 shows the learning curves for the best parameter configuration based on final performance (see sensitivity plots) for the five different agents using different PG estimators for two settings: uniform policy initialization and a sub-optimally saturated policy initialization. The regular and alternate estimators, used either the true reward function r_π or learned a reward baseline that was initialized to zero. For the uniform policy case, we observe that all the methods achieved a good performance. More interestingly, for saturated policy initialization, the alternate estimators (both r_π and with a baseline b) learned good behaviors, whereas the expected PG and the regular estimators failed to learn anything. Further, alternate with a learned baseline converged faster than alternate with r_π. We attribute this result to a better exploration afforded by the baseline.

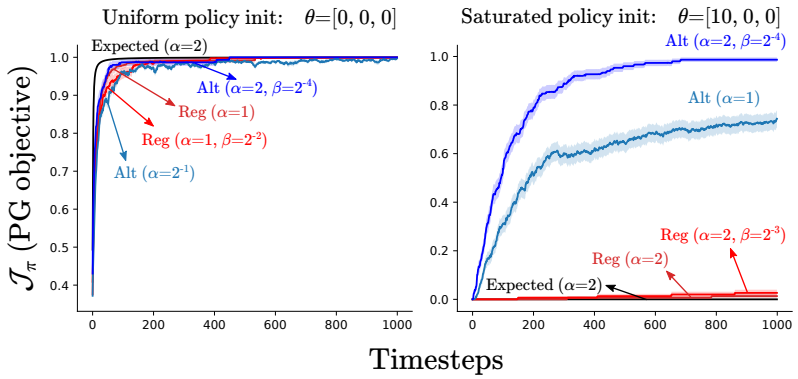

Figure 4.1: Learning curves for gradient bandits. The reward structure is $\mathbf{r} = [0\ 0\ 1]^\top$ with noise $\epsilon \sim \mathcal{N}(0, 1)$. The estimators without an accompanying value of β use the true r_π. The results are averages over 150 independent runs and the shaded regions represent the standard error.

Parameter Sensitivity Plots and Performance with and without Reward Noise

Figure 4.2 shows the parameter sensitivity of the gradient estimators for different policy saturations and reward noises. The graphs plot the mean final performance[1] during last 50 steps of the 1000 timestep run of each method. The performance was measured by analytically calculating $\mathcal{J}_\pi = \mathbb{E}_\pi[R]$. We observe that for uniform initialization, all the methods (irrespective of the magnitude of reward noise) had a similar performance. In contrast, for saturated policies, the

[1]For small stepsizes, the final performance of all the methods are very similar; it is only at higher stepsizes that we start to see a deviation of the performance of both the stochastic PG estimators from each other and the expected gradient bandits update; for a detailed discussion on stochasticity in PG methods, also see Mei et al. (2021).

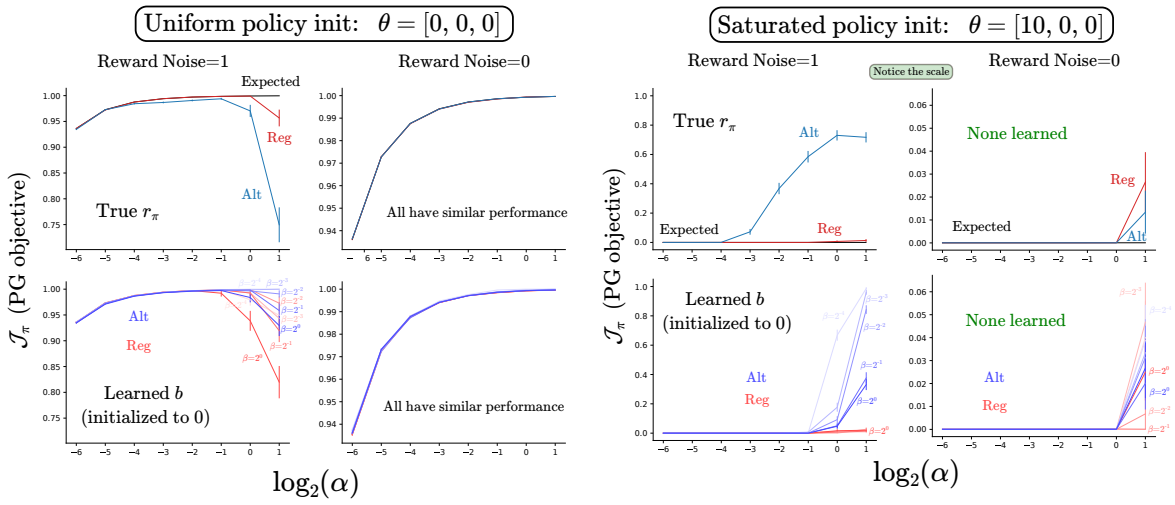

Figure 4.2: Parameter sensitivity plots for the different estimators on the bandit problem with $\mathbf{r} = [0\ 0\ 1]^\top$. We show results for eight different combinations: the estimators using r_π or a learned baseline, uniform or saturated policy initialization, and with or without reward noise. The results are averaged over 150 runs and the bars represent standard error. The x axis shows the policy stepsize α and the curve labels show the reward baseline stepsize β.

alternate estimators with reward noise were able to learn for a range of stepsizes. However, without reward noise, none of the methods escaped the saturation and their policies did not improve. Note that the baseline was initialized to zero which is not optimistic for $\mathbf{r} = [0\ 0\ 1]^\top$ and therefore the alternate estimator could not utilize the bias in the baseline (as discussed in §3.4). This experiment verifies our claim that reward noise helps the alternate estimator to escape the sub-optimal regions in the policy space; without reward noise, the agent is unable to perform a random walk.

Behavior of Action Preferences under Different Estimators

To understand how the action preferences are influenced by different estimators, we plotted their values in Figure 4.3 (with non-zero reward noise) and Figure 4.4 (with zero reward noise) using a set of good performing stepsize parameters. We observe that the behavior of the action preferences for the regular estimator with and without reward noise is similar: the preferences increased quite a bit during the first 100 steps and then did not change much, resulting in flat lines. This probably happened because the policy had saturated and the estimator became negligibly small, i.e. with high probability $\mathbf{e}_A \approx \boldsymbol{\pi}$ and $R \approx r_\pi$, and thus $\mathbf{g}^{\text{REG}} = (R - r_\pi)(\mathbf{e}_A - \boldsymbol{\pi}) \approx \mathbf{0}$. In contrast, reward noise affected the behavior of the alternate estimator (with r_π) quite strongly: the graph for θ_2 with reward noise demonstrates the characteristic random walk pattern. Whereas, in the case of no reward noise, this graph becomes smooth and keeps increasing, since even at saturation, because of missing the $-\boldsymbol{\pi}$ term, the alternate estimator $\mathbf{g}^{\text{ALT}} = (R - r_\pi)\,\mathbf{e}_A$ remains somewhat large. The

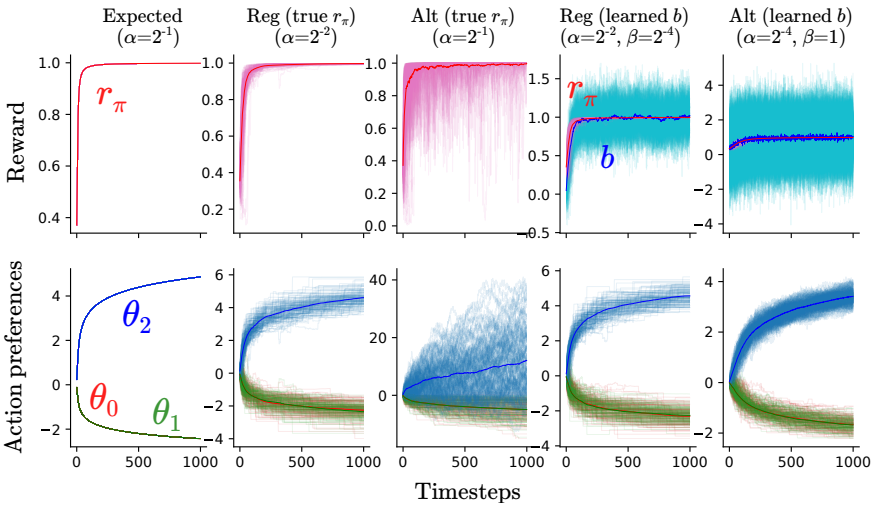

Figure 4.3: **(Top)** Learning curves for expected reward r_π and value of the baseline b as the policy π is updated using different estimators on the bandit problem $\mathbf{r} = [0\ 0\ 1]^\top$ with noise $\epsilon \sim \mathcal{N}(0, 1)$. **(Bottom)** Learning curves for action preferences corresponding to the different estimators. In all cases, the finer lines depict the 150 individual runs and the thick lines show the averages.

patterns for alternate with a learned b are much less clear, probably because of additional effects from the bias in the baseline. We also observe that the alternate estimator with reward noise is noisier than the regular variant; this is expected because the random walk style behavior gives it a higher variance than the regular version (refer Figure 3.3). Alternate without reward noise seems to be less noisy as compared to the regular variant which is again reasonable since in absence of reward noise, alternate has a lower variance than the regular estimator (refer Figure 3.4).

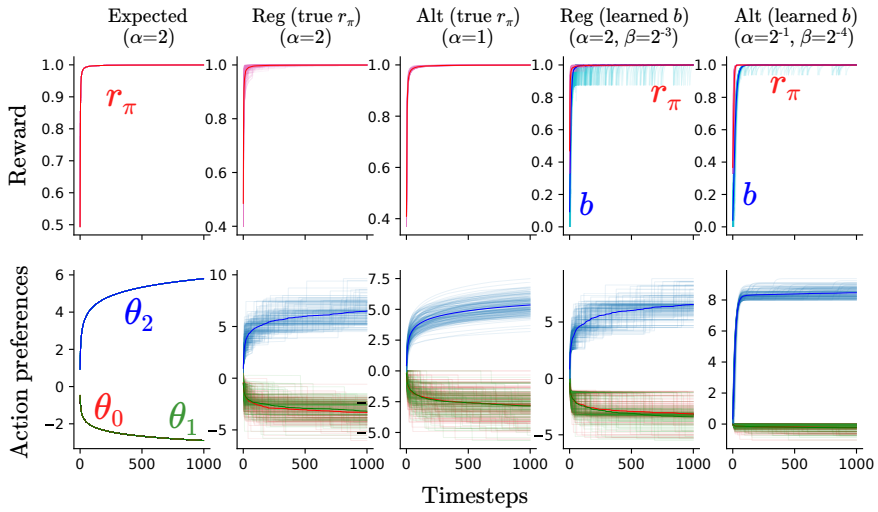

Figure 4.4: Average reward r_π, value of the learned baseline b, and the action preferences against timesteps for different methods on the bandit problem $\mathbf{r} = [0\ 0\ 1]^\top$ and no reward noise.

4.2 Gradient Bandit Estimators with Increasingly Saturated Policy Initializations

In this experiment, we trained the estimators on the bandit task $\mathbf{r} = [1\ 2\ 3]^\top$ with noise $\epsilon \sim \mathcal{N}(0, 1)$ and four different policy saturations, ranging from a uniform policy to a policy highly saturated on the worst action. Figure 4.5 shows the parameter sensitivity plots depicting the final performance of the methods during the last 50 steps of their 1000 timestep run. The results show that as the policy saturation increases, the performance of all the methods degrade. Further, the alternate estimator (with true r_π) is able to handle high saturations (except for $\boldsymbol{\theta}^{\text{init}} = [5\ 0\ 0]^\top$) better than the regular estimator. Whereas, the alternate estimator with a learned baseline initialized to zero (which is pessimistic for this reward structure) did not perform well. This figure, with a baseline initialized to $b = 0$, also serves as a standard (or a control set) against which the results of the next section can be compared, where we explicitly study the effect of an optimistic ($b = +4$) and a pessimistic ($b = -4$) reward baseline on the two estimators' performance.

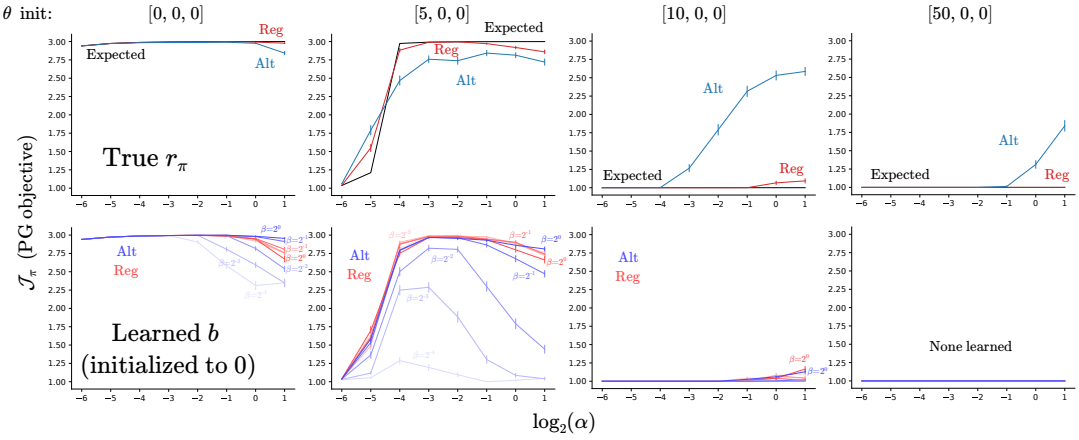

Figure 4.5: Parameter sensitivity of the PG estimators against increasingly saturated policy initializations on the 3-armed bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ with $\epsilon \sim \mathcal{N}(0, 1)$. The top row shows the performance of the estimators with the true r_π. Bottom row shows the performance for a learned baseline, initialized to zero. The subplot headers show how the action preferences were initialized.

4.3 Effect of Bias in the Reward Baselines on the Estimators

In this section, we study the effect of initializing the reward baseline optimistically ($b = +4$) or pessimistically ($b = -4$). The reward was set to $\mathbf{r} = [1\ 2\ 3]^\top$ with noise $\epsilon \sim \mathcal{N}(0, 1)$[2].

[2]The results for the bandit problem with optimistic or pessimistic baselines, and no noise (not shown here) were qualitatively similar to the ones presented here; the signal from the bias in the baseline is quite strong and affects both the estimators much more than the noise in the sampled rewards.

Optimistically or Pessimistically Initialized Baseline

Figure 4.6 and Figure 4.7 show the sensitivity plots for the regular and alternate estimators for an optimistically and a pessimistically initialized baseline respectively. These results show that the optimistic initialization greatly helped the alternate estimator in escaping saturated policy regions. Whereas, a pessimistic initialization significantly hampered the performance of the alternate estimator, making it worse than the regular estimator even for the uniform policy. In contrast, the baseline initialization affected the regular estimator in a milder way. Also with an optimistic initialization, the alternate estimator preferred smaller stepsizes allowing it to enjoy the optimism effect for longer; and vice-versa for a pessimistic initialization.

These results also verify the fact that the alternate estimator is asymptotically unbiased: the sensitivity plots show that the agent was able to converge to the optimal corner (corresponding to action a_2) and thus obtained a reward of about 3. As the agent's estimate of r_π improved, i.e. the baseline b became closer to the average reward r_π, the bias in the alternate estimator disappeared.

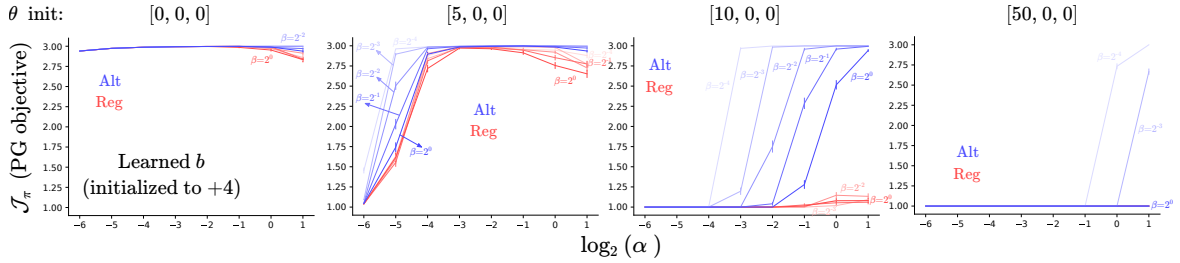

Figure 4.6: Parameter sensitivity plots for the different estimators for an **optimistically** initialized reward baseline ($b = +4$) on the three armed bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ and $\epsilon \sim \mathcal{N}(0, 1)$.

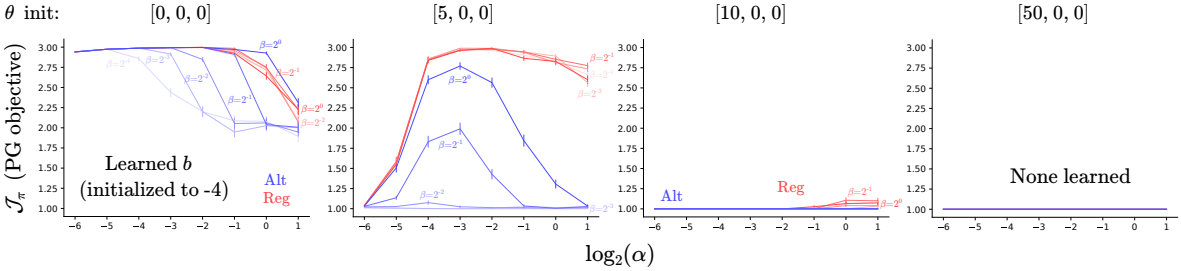

Figure 4.7: Parameter sensitivity plots for the different estimators for a **pessimistically** initialized reward baseline ($b = -4$) on the three armed bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ and $\epsilon \sim \mathcal{N}(0, 1)$.

Fixed Baseline

Now we study what happens when we fix the baseline and not learn it, thereby maintaining the biasedness of the alternate estimator. Figure 4.8 shows the parameter sensitivity plots for this

setting for multiple policy initializations. From the results, we see that the regular estimator was not affected by a fixed baseline (since it remains unbiased with any baseline): it was able to learn a good, but not the optimal, policy for a uniform policy initialization. In contrast, the alternate estimator was impacted heavily: for an optimistic baseline, it converged to a value of about 2.4 and for a pessimistic baseline it converged to a value of zero. The values to which the alternate estimator converged remained the same irrespective of the policy initialization. This hints at the fact that the update signal offered by the alternate estimator, due to bias in the critic, is quite strong and can override the high saturation in softmax policies.

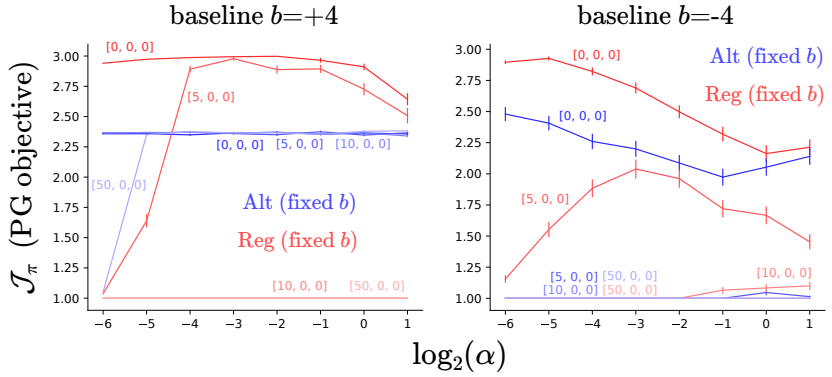

Figure 4.8: Bias in the alternate PG estimator. Parameter sensitivity of the PG estimators for a fixed baseline on bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ with $\epsilon \sim \mathcal{N}(0, 1)$. The baseline in these experiments was fixed and not updated, and therefore, the alternate estimator remained biased throughout the learning. The individual labels on the curves show the action preference initializations.

This experiment, additionally, serves as a verification of Lemma 8.1. Recall from the lemma that for a fixed baseline, the fixed point of the expected alternate update is given by $\pi^*(a) = \frac{1}{r(a)-b}\left(\sum_c \frac{1}{r(c)-b}\right)^{-1}$. Evaluating this expression for this experiment gives $\boldsymbol{\pi}^* = [0.18\ 0.27\ 0.55]^\top$. Using $\mathbf{r} = [1\ 2\ 3]^\top$, we can calculate $\mathcal{J}_{\pi^*} = \boldsymbol{\pi}^{*\top}\mathbf{r} = 2.36$. And unsurprisingly, this value matches the value shown in Figure 4.8 (left) to which the alternate estimator converged under various policy initializations. Further, as we had predicted from the update plots shown in Figure 3.6 (bottom-right) and remarked in Footnote 5 (§3.4), the alternate estimator with a pessimistic baseline converged to the nearest corner. For policies saturated on action a_0, this corner corresponded to action a_0 as well and thus the agents obtained a reward of 1 for all the policy saturations considered.

4.4 Can the Regular Estimator be Modified to Deal with Saturated Policies?

In this section we consider whether certain modifications to the regular estimator can make it effective in dealing with saturated softmax policies.

Dropping the Reward Baseline

From our previous experiments, we know that the higher variance of the alternate estimator helps it in escaping the corners of the probability simplex. So we investigate whether a similar increase in variance obtained by removing the reward baseline from the regular estimator will help it in escaping the corners as well. Figure 4.9 shows the sensitivity of the final mean performance (during the last 50 timesteps of a 1000 timestep run) to the stepsize α for the regular estimator without any baseline on the bandit task $\mathbf{r} = [1\ 2\ 3]^\top$ with $\epsilon \sim \mathcal{N}(0, 1)$. The results show that the regular estimator without a baseline cannot be a good competitor to the alternate estimator for escaping the saturated policy regions. Upon careful thought, this should not be surprising: the regular estimator, with or without the baseline, has zero variance at simplex corners (see Proposition 7 and Figure 3.3). Consequently, there is neither any gradient signal nor any noise near saturated regions and the policy gets stuck. This is unlike the alternate estimator which not just has a higher variance, but a variance structure that is better suited to escaping the saturated policy regions.

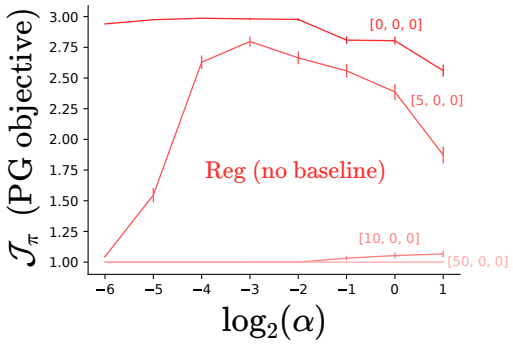

Figure 4.9: Parameter sensitivity plots for the regular estimator without a reward baseline. The individual curve labels show the action preference initializations.

Adding Normally Distributed Noise to the Policy Updates

The reward noise utilized by the alternate estimator ultimately gets translated into noisy updates for the action preferences, whereas this translation doesn't go through very well for the regular estimator, especially at the corners (refer Example 1). Therefore, in this experiment, we study the effect of adding an artificial noise to the gradient updates for both the estimators. In particular, we trained the different estimators on the bandit problem $\mathbf{r} = [1\ 2\ 3]^\top$ with $\epsilon \sim \mathcal{N}(0, 1)$. And to make the updates noisy, we added a normally distributed noise to the gradient updates, i.e. for each action a, the action preferences were updated as follows: $\theta_a^{\text{new}} = \theta_a^{\text{old}} + \alpha(g_a + \xi)$, where $\xi \sim \mathcal{N}(0, \texttt{grad-noise})$; we also swept over the the parameter $\texttt{grad-noise} \in \{0, 0.1, 0.5, 1, 2\}$. We experimented with a policy saturated to $\boldsymbol{\theta} = [10\ 0\ 0]^\top$ at the beginning of training; and additionally

a uniform policy to gauge whether this additional update noise had a detrimental effect on the estimators' performance, since with a uniform policy there is already a strong gradient signal and the agent does not require any extraneous noise for moving towards the optimal policy. Figure 4.10 shows the sensitivity of the final performance for the expected gradient, and the regular and the alternate estimators using the true r_π. Figure 4.11 shows similar results for the regular and alternate estimators with a learned baseline, initialized to zero.

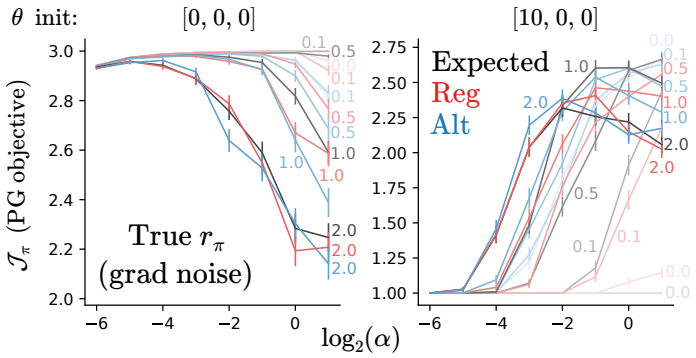

Figure 4.10: Sensitivity plots for the gradient bandit estimators with true r_π and an additional normally distributed gradient noise. The individual graph labels $(0, 0.1, 0.5, 1, 2)$ show the standard deviation of the normal noise added to the gradient. The subplot headers show the policy saturation at the beginning of the experiment.

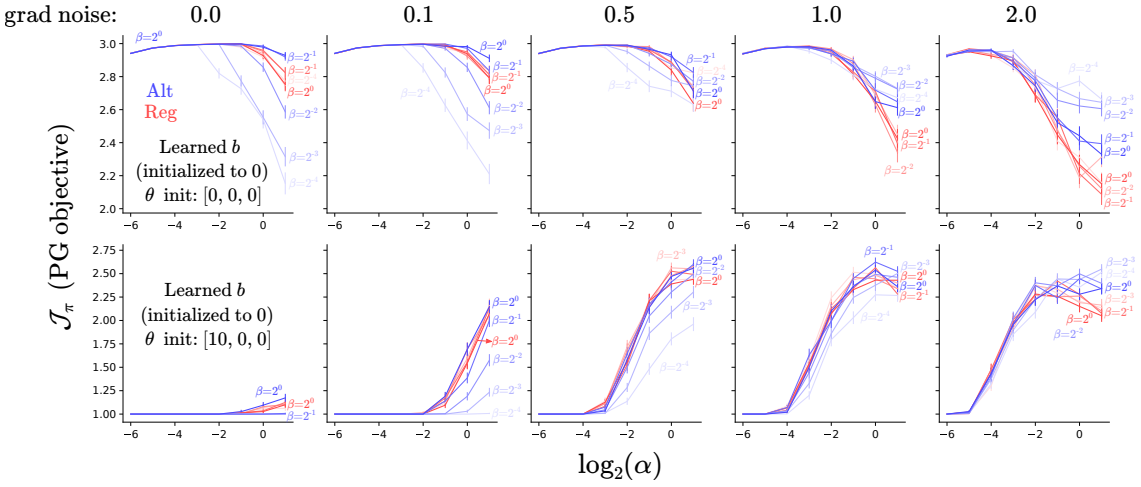

Figure 4.11: Sensitivity plots for the regular and the alternate estimators with added gradient noise and a learned baseline initialized to zero. The figure header shows the standard deviation of the Gaussian gradient noise. Top and bottom rows correspond to uniform and saturated policy initializations respectively. Interestingly, for the uniform policy case and a high standard deviation of the noise, the alternate estimator is more robust to the gradient noise than the regular estimator.

The results from this experiment suggest that adding a normal noise to the gradient bandit update works well for escaping the saturated policy regions in this bandit setting for all the five

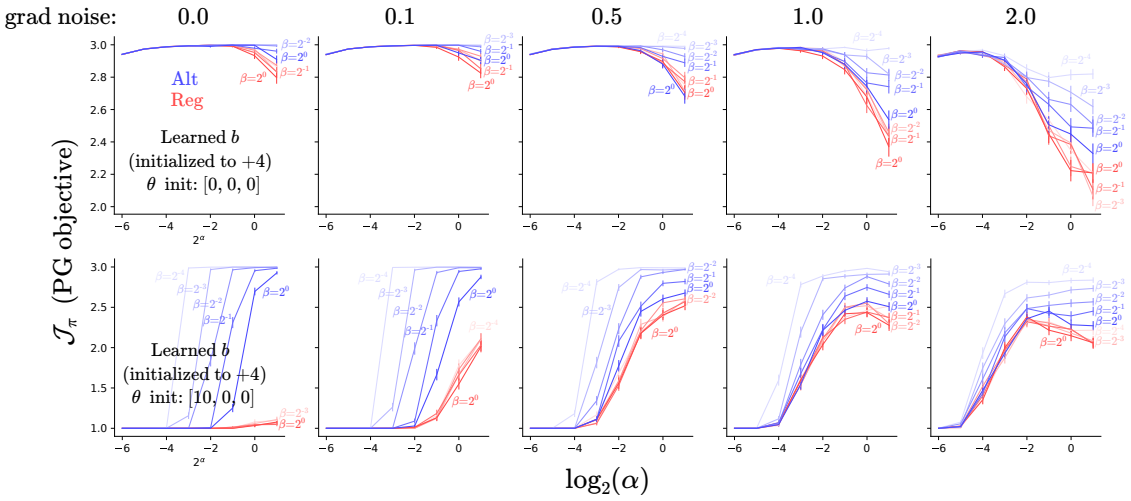

Figure 4.12: Sensitivity plots for the estimators with gradient noise and an optimistic baseline.

estimators; in particular, the regular estimator can escape the saturation with a high enough noise. At first, this observation might seem to weaken the case for the alternate estimator. However, note that adding gradient noise also worsens the performance of all the estimators for the uniform policy case, which is clearly an undesirable property. Further, recall that the alternate estimator benefits much more from an optimistic baseline than it does from the reward noise. In contrast to the noise, an optimistic baseline actually offers an clear signal that moves the agent out of saturation and towards a more uniform policy distribution. Then, the obvious follow-up question to ask is whether adding gradient noise to the regular estimator makes it more effective than the alternate estimator with an optimistically initialized baseline. Figure 4.12, which shows the sensitivity plots for the regular and the alternate estimator with both an optimistic baseline and added gradient noise, answers this question in a negative. Comparing the results from Figure 4.12 with those from Figure 4.11 shows that the alternate estimator benefits from an optimistic baseline much more than the regular estimator benefits from added gradient noise. More importantly, even though adding gradient noise helped with this particular bandit instance, as we show in §6.3, adding gradient noise doesn't help the regular estimator overcome saturated policies in MDPs.

4.5 Summary

The experiments in this chapter demonstrated that the alternate estimator has a superior (or competitive) performance as compared to the regular estimator in a variety of 3-armed bandit problems. The experimental results also verify the properties stated in the previous chapter, i.e. the alternate estimator can utilize an optimistic baseline and reward noise to effectively overcome policy saturation and learn a good policy.

Chapter 5

Alternate Policy Gradient Estimator for MDPs

The alternate gradient estimator can be readily extended to MDPs, where it enjoys properties analogous to those it enjoyed in the bandit case. We begin by introducing the vector notation for MDPs and then present the main derivations.

5.1 Manipulating Vector Variables to Calculate PG Estimators

We initially derive the regular and the alternate policy gradient estimators for the following PG expression (this is the exact same form as discussed in §2.7):

$$\nabla \mathcal{J}_\pi = \sum_s \nu_\pi(s) \sum_a \pi(a|s) \nabla \log \pi(a|s) q_\pi(s,a), \tag{5.1}$$

We chose this particular equation because it has the simplest form amongst all the PG expressions we have considered till now[1]: it doesn't have a sum over the sampled trajectories (like the REINFORCE expression, Eq. 2.26), nor has the pesky discounting factor (from the online AC expression, Eq. 2.31). We could have equivalently begun with the REINFORCE or the online AC expressions as well, and we eventually do consider them anyways in §5.3. Now our goal is to express Eq. 5.1 in a vector form. To do so, let us introduce some notation: for a given state s, let $\boldsymbol{\pi}(\cdot|s) \in \mathbb{R}^{|\mathcal{A}|}$ be the policy vector with $[\boldsymbol{\pi}(\cdot|s)]_a := \pi(a|s)$ and $\mathbf{q}_\pi(s,\cdot) \in \mathbb{R}^{|\mathcal{A}|}$ be the action value vector with $[\mathbf{q}_\pi(s,\cdot)]_a := q_\pi(s,a)$. Continuing from Eq. 5.1, the vector form for policy gradient can be readily

[1] This not quite true though, since Eq. 5.1 packs all the complex terms and summations inside the occupancy measure $\nu_\pi := \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(S_k = s)$. Nevertheless for our purposes, this expression is the easiest to deal with.

obtained as follows:

$$\nabla_{\mathbf{w}} \mathcal{J}_\pi = \sum_s \nu_\pi(s) \sum_a \pi(a|s) \nabla_{\mathbf{w}} \log \pi(a|s) q_\pi(s,a) = \sum_s \nu_\pi(s) \sum_a \nabla_{\mathbf{w}} \pi(a|s) q_\pi(s,a)$$
$$= \sum_s \nu_\pi(s) \big[\nabla_{\mathbf{w}} \boldsymbol{\pi}(\cdot|s)\big] \, \mathbf{q}_\pi(s,\cdot), \tag{5.2}$$

where $\nabla_{\mathbf{w}} \boldsymbol{\pi}(\cdot|s)$ represents the transpose of the Jacobian matrix (see §2.8). To obtain the PG estimators from Eq 5.2, we will re-write the term $\nabla_{\mathbf{w}} \boldsymbol{\pi}(\cdot|s)$ to explicitly separate the gradient of the action likelihood $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s)$ from that of the preference $\nabla_{\mathbf{w}} \boldsymbol{\theta}(s)$. After that, we will analytically calculate the gradient of the action likelihood, assuming a softmax policy. Before moving forward, we recall the definition of the softmax policy for MDPs and introduce some additional notation along the way. The softmax policy, for an arbitrary state s and action a, is given by

$$\pi(a|s) = \frac{e^{[\boldsymbol{\theta}_{\mathbf{w}}(s)]_a}}{\sum_{c \in \mathcal{A}} e^{[\boldsymbol{\theta}_{\mathbf{w}}(s)]_c}}, \tag{5.3}$$

where $\boldsymbol{\theta}_{\mathbf{w}} : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ denotes a vector function, parameterized by $\mathbf{w}$, that gives the action preference vector for a given state input. The action preference for a specific action is the corresponding element of this vector: for instance, $[\boldsymbol{\theta}_{\mathbf{w}}(s)]_a$ gives the action preference corresponding to action a. In practice, $\boldsymbol{\theta}_{\mathbf{w}}$ could be implemented using, say, a neural network with weights $\mathbf{w}$. Such a network would take a state as the input and output the action preferences corresponding to all the different actions of the action space. For brevity, we will drop $\mathbf{w}$ and simply write $\boldsymbol{\theta}_{\mathbf{w}}$ as $\boldsymbol{\theta}$, even though the action preferences depend on the parameters $\mathbf{w}$. With this definition of $\boldsymbol{\theta}$ and $\boldsymbol{\pi}$, we can use Proposition 4 to get[2]

$$\nabla_{\mathbf{w}} \boldsymbol{\pi}(\cdot|s) = \big[\nabla_{\mathbf{w}} \boldsymbol{\theta}(s)\big] \big[\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s)\big]. \tag{5.4}$$

Putting the above expression in Eq. 5.2 gives us the following form for the policy gradient:

$$\nabla_{\mathbf{w}} \mathcal{J}_\pi = \sum_s \nu_\pi(s) \big[\nabla_{\mathbf{w}} \boldsymbol{\theta}(s)\big] \nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s) \, \mathbf{q}_\pi(s,\cdot). \tag{5.5}$$

In the above expression, the gradient of the action likelihood, i.e. $\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s)$, and the gradient of action preferences which solely depend on the states, i.e. $\nabla_{\mathbf{w}} \boldsymbol{\theta}(s)$, are explicitly separated. We could split these terms, by using Eq. 5.4, because the softmax policy processes the states and the actions in a sequential manner: it first computes the action preferences corresponding to a particular state and then calculates the probability of a given action utilizing these preferences[3].

[2]The last term in Eq. 5.4 should technically be written as $\nabla_{\boldsymbol{\theta}(s)} \boldsymbol{\pi}(\cdot|s)$. However, for the sake of brevity, we dropped the function argument "(s)"; the dependence of $\boldsymbol{\theta}$ on state s should be clear from the context.

[3]Another example of such a sequential processing is Q-learning (Watkins and Dayan, 1992; Mnih et al., 2013), where the agent first calculates all state-action values (such as the last layer of the DQN network) for each of the actions, and then uses one of these state-action values based on the particular action sampled at that point.

In the analysis that follows we will fix π to be a softmax policy and analytically work out the gradients corresponding to the action likelihood. Though, we leave the gradient corresponding to states which will be computed later using, say, an automatic differentiation package based on the specific function approximator used (see §5.4 for the linear case). Using Eq. 3.2, we get

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s)\, \mathbf{q}_\pi(s,\cdot) &= \big(\mathbf{I} - \boldsymbol{\pi}(\cdot|s)\mathbf{1}^\top\big)\, \mathrm{diag}(\boldsymbol{\pi}(\cdot|s))\, \mathbf{q}_\pi(s,\cdot) \\
&= \mathrm{diag}(\boldsymbol{\pi}(\cdot|s))\, \mathbf{q}_\pi(s,\cdot) - \boldsymbol{\pi}(\cdot|s)\mathbf{1}^\top\, \mathrm{diag}(\boldsymbol{\pi}(\cdot|s))\, \mathbf{q}_\pi(s,\cdot) \\
&= \boldsymbol{\pi}(\cdot|s) \odot \mathbf{q}_\pi(s,\cdot) - \boldsymbol{\pi}(\cdot|s)\, \boldsymbol{\pi}(\cdot|s)^\top\, \mathbf{q}_\pi(s,\cdot) \qquad (5.6) \\
&= \boldsymbol{\pi}(\cdot|s) \odot (\mathbf{q}_\pi(s,\cdot) - v_\pi(s)\mathbf{1}). \qquad (5.7)
\end{aligned}$$

Note that this expression looks exactly the same as Eq. 3.4, except that the reward vector is replaced by the action value function and the average reward is replaced by the value function. Continuing with Eq. 5.6 and moving some terms around leads to

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \boldsymbol{\pi}(\cdot|s)\, \mathbf{q}_\pi(s,\cdot) &= \mathrm{diag}(\mathbf{q}_\pi(s,\cdot))\, \boldsymbol{\pi}(\cdot|s) - \boldsymbol{\pi}(\cdot|s)\, \mathbf{q}_\pi(s,\cdot)^\top\, \boldsymbol{\pi}(\cdot|s) \\
&= [\mathrm{diag}(\mathbf{q}_\pi(s,\cdot)) - \boldsymbol{\pi}(\cdot|s)\, \mathbf{q}_\pi(s,\cdot)^\top]\, \boldsymbol{\pi}(\cdot|s) \\
&= \sum_a \pi(a|s) q_\pi(s,a) \big(\mathbf{e}_a - \boldsymbol{\pi}(\cdot|s)\big). \qquad (5.8)
\end{aligned}$$

The regular estimator for MDPs can be found by putting Eq. 5.8 into Eq. 5.5, which yields

$$\begin{aligned}
\nabla_{\mathbf{w}} \mathcal{J}_\pi &= \sum_s \nu_\pi(s) \sum_a \pi(a|s) \nabla_{\mathbf{w}} \boldsymbol{\theta}(s) \big(\mathbf{e}_a - \boldsymbol{\pi}(\cdot|s)\big) q_\pi(s,a) \\
&= \mathbb{E}_{S\sim\nu_\pi; A\sim\pi(\cdot|S)} \Big[\nabla_{\mathbf{w}} \boldsymbol{\theta}(S) \big(\mathbf{e}_A - \boldsymbol{\pi}(\cdot|S)\big) q_\pi(S,A) \Big] \\
&= \mathbb{E}_{S\sim\nu_\pi; A\sim\pi(\cdot|S)} \Big[\underbrace{\Big(\nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A - \sum_a \pi(a|S) \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_a \Big) \big(q_\pi(S,A) - v_\pi(S)\big)}_{=: \mathbf{g}^{\mathrm{REG}}(S,A)} \Big], \qquad (5.9)
\end{aligned}$$

where we subtracted the action independent baseline $v_\pi(S)$ without introducing any bias. Even though the estimator $\mathbf{g}^{\mathrm{REG}}$ looks new, it is exactly equal to the familiar PG estimator from before:

$$\mathbf{g}^{\mathrm{REG}}(S,A) = \nabla_{\mathbf{w}} \log \pi(A|S) \big(q_\pi(S,A) - v_\pi(S)\big). \qquad (5.10)$$

To see this equivalence, note that for a softmax policy π and the state-action pair (S,A),

$$\begin{aligned}
\nabla_{\mathbf{w}} \log \pi(A|S) &= \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A - \nabla_{\mathbf{w}} \log \left(\sum_a e^{[\boldsymbol{\theta}(S)]_a} \right) = \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A - \frac{\sum_a e^{[\boldsymbol{\theta}(S)]_a} \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_a}{\sum_b e^{[\boldsymbol{\theta}(S)]_b}} \\
&= \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A - \sum_a \pi(a|S) \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_a.
\end{aligned}$$

Therefore, this lengthy calculation merely resulted in a different expression for the already familiar PG estimator; it did not yield a new one and thus was little more than an exercise in manipulating the vector variables $\boldsymbol{\pi}(\cdot|s)$ and $\boldsymbol{\theta}(s)$. But now that we are warmed up, we can use this new skill to derive the alternate PG estimator for MDPs.

5.2 Deriving the Alternate PG Estimator

In order to derive the alternate PG estimator, let us again start with Eq. 5.6:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}}\, \boldsymbol{\pi}(\cdot|s)\, \mathbf{q}_\pi(s,\cdot) &= \operatorname{diag}(\mathbf{q}_\pi(s,\cdot))\, \boldsymbol{\pi}(\cdot|s) - \boldsymbol{\pi}(\cdot|s)^\top\, \mathbf{q}_\pi(s,\cdot)\, \boldsymbol{\pi}(\cdot|s) \\ &= [\operatorname{diag}(\mathbf{q}_\pi(s,\cdot)) - v_\pi(s)\, \mathbf{I}]\, \boldsymbol{\pi}(\cdot|s) \\ &= \sum_a \pi(a|s) \Big(q_\pi(s,a) - v_\pi(s) \Big)\, \mathbf{e}_a, \end{aligned} \tag{5.11}$$

where we used the fact that $v_\pi(s) = \sum_{a\in\mathcal{A}} \pi_{\mathbf{w}}(a|s) q_\pi(s,a) = \boldsymbol{\pi}_{\mathbf{w}}(\cdot|s)^\top\, \mathbf{q}_\pi(s,\cdot)$. Analogous to the previous section, we put Eq. 5.11 in Eq. 5.5 to obtain[4]

$$\begin{aligned}\nabla_{\mathbf{w}} \mathcal{J}_\pi &= \sum_s \nu_\pi(s) \sum_a \pi(a|s) \nabla_{\mathbf{w}}\, \boldsymbol{\theta}(s)\, \mathbf{e}_a \Big(q_\pi(s,a) - v_\pi(s) \Big) \\ &= \mathbb{E}_{S\sim\nu_\pi; A\sim\pi(\cdot|S)} \Big[\nabla_{\mathbf{w}}\, \boldsymbol{\theta}(S)\, \mathbf{e}_A \big(q_\pi(S,A) - v_\pi(S) \big) \Big] \\ &= \mathbb{E}_{S\sim\nu_\pi; A\sim\pi(\cdot|S)} \Big[\underbrace{\nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A \big(q_\pi(S,A) - v_\pi(S) \big)}_{=: \mathbf{g}^{\mathrm{ALT}}(S,A)} \Big]. \end{aligned} \tag{5.12}$$

One way to look at the alternate gradient estimator is that it is equivalent to taking the gradient of the softmax policy, but by dropping the normalization constant, i.e. $\nabla_{\mathbf{w}} \log \frac{e^{[\boldsymbol{\theta}(S)]_A}}{1} = \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A$. Also note that even though the regular and the alternate PG estimators are equal in expectation, in general they are not equal for any arbitrary state-action pair (S,A): $\mathbf{g}^{\mathrm{REG}}(S,A) \neq \mathbf{g}^{\mathrm{ALT}}(S,A)$.

The alternate estimator for MDPs looks very similar to the alternate estimator for bandits (Eq. 3.7). The differences are that in the alternate estimator for MDPs, the reward vector from bandits is replaced by the action value vector, the average reward is replaced by the value function, and it additionally has the gradient of the action preference corresponding to action A with respect to $\mathbf{w}$. These differences suggest that, similar to the alternate estimator for gradient bandits, the alternate estimator for MDPs is unbiased when we use the true value function $v_\pi(S)$ and becomes

[4] This derivation is not entirely novel: in fact, it was recently brought to our attention that Young (2019) studied the exact same PG estimator in a different context. Interestingly, page 37 (Young, 2019) gives the equivalent of Eq. 5.11 presented above, albeit with a different notation and a calculation that uses scalar variables. The focus of Young (2019) was on saving compute by only calculating the gradient of a single action preference (that line of work is weakly related to what we discussed in §3.6). In contrast, this project studies the viability of the alternate estimator in decreasing the effect of policy saturation on the performance of softmax policies.

biased if we use another baseline. Further, recalling that $q_\pi(S, A)$ is usually estimated by an n-step return, the alternate estimator can utilize the noise in computing these n-step returns and also the bias in the critic estimate. So, if the value function estimate $b(S)$ for state S is optimistic, i.e. $b(S) > \hat{q}_\pi(S, a)$ for all actions a, then the policy will be updated towards a more uniform distribution. We demonstrate these properties via an empirical analysis in the next chapter[5]. In the remaining sections, we present the alternate estimator in context of different PG algorithms.

5.3 Adapting the Alternate Estimator to various PG Algorithms

The form for the alternate estimator given by Eq. 5.12 needs to be adapted to a concrete policy gradient algorithm before it can be used by an RL agent. In this section, we demonstrate this adaption process for the Proximal Policy Optimization algorithm (abbreviated as PPO, Schulman et al., 2017). After that, we directly give the expressions for the alternate estimator adapted to various PG algorithms such as REINFORCE (Williams, 1991), Online Actor-Critic (Sutton and Barto, 2018), and Trust Region Policy Optimization (abbreviated as TRPO, Schulman et al., 2015).

The Alternate PPO Estimator

PPO proposes the following clipped objective that the agent maximizes at each step:

$$\mathcal{J}_{\text{PPO}} := \sum_s \nu_{\pi_{\text{old}}}(s) \sum_a \pi_{\text{old}}(a|s) \cdot \min \begin{pmatrix} \frac{\pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)} h_{\pi_{\text{old}}}(s,a), \\ \text{clip}\left[\frac{\pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)}, 1-\epsilon, 1+\epsilon\right] h_{\pi_{\text{old}}}(s,a) \end{pmatrix}, \tag{5.13}$$

where $h_{\pi_{\text{old}}}(s,a) := q_{\pi_{\text{old}}}(s,a) - v_{\pi_{\text{old}}}(s)$ is the advantage function under the old policy π_{old}, and ϵ is a fixed parameter. The PPO objective introduces an importance sampling ratio to reuse the data collected by the old policy to make multiple update steps to the new policy[6]. Further, the agent only makes updates for those samples which satisfy a particular condition that we define shortly. Using this view, the gradient of the above objective can be equivalently written as

$$\begin{aligned} \nabla_{\mathbf{w}} \mathcal{J}_{\text{PPO}} &= \sum_s \nu_{\pi_{\text{old}}}(s) \sum_a \pi_{\text{old}}(a|s) \cdot \mathbb{I}\Big(\text{cond}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a)\Big) \frac{\nabla_{\mathbf{w}} \pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)} h_{\pi_{\text{old}}}(s,a) \\ &= \sum_s \nu_{\pi_{\text{old}}}(s) \sum_a \nabla_{\mathbf{w}} \pi_{\mathbf{w}}(a|s) h_{\pi_{\text{old}}}(s,a) \mathbb{I}\Big(\text{cond}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a)\Big), \end{aligned} \tag{5.14}$$

[5]Note that we simply mentioned the properties of the alternate estimator for MDPs. This chapter is missing the analogous theoretical results, especially Theorem 1, presented earlier for bandits. However, based on empirical evidence (see §6.2, §7.3, and §7.7) and intuition, we conjecture that similar results should hold for the case of MDPs as well. Further, our hope is that it should not be too difficult to show them theoretically.

[6]Despite this reuse of the data collected by the old policy, PPO is still very much an on-policy PG algorithm since it does not correct for the state distribution mismatch between π_{old} and the current policy: $\nu_{\pi_{\text{old}}}(s) \neq \nu_{\pi_{\mathbf{w}}}(s)$.

where cond() is the Boolean condition defined as

$$\text{cond}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a) := \left(h_{\pi_{\text{old}}}(s,a) > 0 \bigwedge \frac{\pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)} < 1+\epsilon\right) \bigvee \left(h_{\pi_{\text{old}}}(s,a) < 0 \bigwedge \frac{\pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)} > 1-\epsilon\right).$$

To reduce clutter, let us define $\tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a) := h_{\pi_{\text{old}}}(s,a)\mathbb{I}\Big(\text{cond}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a)\Big)$. Then the regular PPO estimator can be written as

$$\nabla_{\mathbf{w}} \mathcal{J}_{\text{PPO}} = \mathbb{E}_{S\sim\nu_{\pi_{\text{old}}};A\sim\pi_{\text{old}}(\cdot|S)} \underbrace{\left[\nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(A|S) \frac{\pi_{\mathbf{w}}(A|S)}{\pi_{\text{old}}(A|S)} \tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(S,A)\right]}_{=:\mathbf{g}^{\text{REG-PPO}}(S,A)}.$$

Now we follow the same procedure that we used to obtain the alternate estimator in Eq. 5.11, but for the expression that appeared in the PPO gradient (Eq. 5.14):

$$\sum_a \nabla_{\mathbf{w}} \pi_{\mathbf{w}}(a|s) \tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a) = \nabla_{\mathbf{w}} \boldsymbol{\theta}(s) \sum_a \pi_{\mathbf{w}}(a|s) \left[\tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a) - \sum_c \pi_{\mathbf{w}}(c|s) \tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,c)\right] \mathbf{e}_a \,.$$

Putting the above expression into Eq. 5.14, we get the alternate estimator for PPO:

$$\begin{aligned}
&\nabla_{\mathbf{w}} \mathcal{J}_{\text{PPO}} \\
&= \sum_s \nu_{\pi_{\text{old}}}(s) \sum_a \pi_{\text{old}}(a|s) \nabla_{\mathbf{w}} [\boldsymbol{\theta}(s)]_a \frac{\pi_{\mathbf{w}}(a|s)}{\pi_{\text{old}}(a|s)} \left[\tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,a) - \sum_c \pi_{\mathbf{w}}(c|s) \tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(s,c)\right] \\
&= \mathbb{E}_{S\sim\nu_{\pi_{\text{old}}};A\sim\pi_{\text{old}}(\cdot|S)} \underbrace{\left[\nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A \frac{\pi_{\mathbf{w}}(A|S)}{\pi_{\text{old}}(A|S)} \left(\tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(S,A) - \sum_c \pi_{\mathbf{w}}(c|S) \tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}(S,c)\right)\right]}_{=:\mathbf{g}^{\text{ALT-PPO}}(S,A)}. \qquad (5.15)
\end{aligned}$$

Using a process similar to that demonstrated here, we can extend the alternate estimator to different PG algorithms. We now summarize these estimators.

Alternate REINFORCE Estimator

The regular REINFORCE estimator is given by Eq. 2.27. And the alternate estimator is given by

$$\nabla_{\mathbf{w}} \mathcal{J}_{\pi} = \mathbb{E}_{H_\infty\sim\mathbb{P}_\pi} \underbrace{\left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S_t)]_{A_t} \Big(G_t - v_\pi(S_t)\Big)\right]}_{=:\mathbf{g}^{\text{ALT-REINF}}(H_\infty)}. \qquad (5.16)$$

Alternate Online AC Estimator

The regular estimator is given by Eq. 2.32 and the alternate estimator is

$$\nabla_{\mathbf{w}} \mathcal{J}_\pi \propto \mathbb{E}_{S,K\sim d_\pi; A\sim\pi(\cdot|S); S',R\sim p(\cdot,\cdot|S,A)} \underbrace{\Big[\gamma^K \nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A \Big(R + \gamma v_\pi(S') - v_\pi(S)\Big)\Big]}_{=:\mathbf{g}^{\text{ALT-OAC}}(S,A,R,S',K)}. \tag{5.17}$$

TRPO Estimators

TRPO (Eq. 14, Schulman et al., 2015) proposes that the agent solve the following constrained optimization problem at each timestep

$$\max_{\mathbf{w}} \underbrace{\sum_s \nu_{\pi_{\text{old}}}(s) \sum_a \pi_{\mathbf{w}}(a|s) q_{\pi_{\text{old}}}(s,a)}_{=:\mathcal{J}_{\text{TRPO}}} \qquad \text{subject to } \sum_s \nu_{\pi_{\text{old}}}(s) \cdot D_{\text{KL}}\big(\pi_{\text{old}}(\cdot|s) \big\| \pi_{\mathbf{w}}(\cdot|s)\big) \leq \delta,$$

where δ is a fixed parameter.

We can re-write the gradient of the objective $\mathcal{J}_{\text{TRPO}}$ as follows

$$\nabla_{\mathbf{w}} \mathcal{J}_{\text{TRPO}} = \mathbb{E}_{S\sim\nu_{\pi_{\text{old}}}; A\sim\pi_{\text{old}}(\cdot|S)} \underbrace{\left[\nabla_{\mathbf{w}} \log \pi_{\mathbf{w}}(A|S) \frac{\pi_{\mathbf{w}}(A|S)}{\pi_{\text{old}}(A|S)} q_{\pi_{\text{old}}}(S,A)\right]}_{=:\mathbf{g}^{\text{REG-TRPO}}(S,A)}.$$

Then, it is straightforward to see that the alternate estimator would become

$$\nabla_{\mathbf{w}} \mathcal{J}_{\text{TRPO}} = \mathbb{E}_{S\sim\nu_{\pi_{\text{old}}}; A\sim\pi_{\text{old}}(\cdot|S)} \underbrace{\left[\nabla_{\mathbf{w}} [\boldsymbol{\theta}(S)]_A \frac{\pi_{\mathbf{w}}(A|S)}{\pi_{\text{old}}(A|S)} \Big(q_{\pi_{\text{old}}}(S,A) - \sum_c \pi_{\mathbf{w}}(c|S) q_{\pi_{\text{old}}}(S,c)\Big)\right]}_{=:\mathbf{g}^{\text{ALT-TRPO}}(S,A)}. \tag{5.18}$$

The alternate estimators proposed here for TRPO and PPO require the computation of the expectation of $q_{\pi_{\text{old}}}$ and $\tilde{h}^{\pi_{\mathbf{w}}}_{\pi_{\text{old}}}$ respectively, under the current policy $\pi_{\mathbf{w}}$ and across the action space. However, if the number of actions is large, this operation becomes prohibitively expensive[7]. In such cases, techniques (also see Ciosek and Whiteson, 2020) such as maintaining another function approximator, analogous to a value function, or sampling a subset of actions to approximate this expectation can be used.

[7]Moreover, if this expectation is calculated exactly, the alternate (TRPO or PPO) estimator will remain unbiased. Depending on the context this might be a good or a bad property to have. However, in this work, we do not pursue a more involved analysis of the alternate estimators for TRPO and PPO. Also note that the objectives maximized by PPO and TRPO are different from the original PG objective $\mathcal{J}_\pi := \sum_s \nu_{\pi_{\mathbf{w}}}(s) \sum_a \pi_{\mathbf{w}}(a|s) q_{\pi_{\mathbf{w}}}(s,a)$ which would probably make such an analysis more interesting.

5.4 Additional Details for Implementing Online Linear AC

In this section[8], we provide details that can be used when implementing a PG agent (we used these for the experiments presented in §7) with the estimators presented above. In particular, we calculate the term $\nabla_{\mathbf{w}}\,\boldsymbol{\theta}(S)$ (for the equations in §5.1 and §5.2) to yield analytical policy gradient update expressions. To keep things concrete, we will focus on the expected policy gradient expression, and the regular and the alternate online AC estimators for softmax policies with linear function approximation. These details can be extended in a straightforward way to work with other PG algorithms such as REINFORCE and different policy parameterizations such as the tabular policies.

Recall from Eq. 5.3, that the softmax policy is given by $\pi(a|s) = \frac{e^{[\boldsymbol{\theta}(s)]_a}}{\sum_{c\in\mathcal{A}} e^{[\boldsymbol{\theta}(s)]_c}}$. For linear function approximation and d-dimensional state features, $[\boldsymbol{\theta}(s)]_a := \mathbf{w}_a^\top\,\mathbf{x}(s)$, with $\mathbf{w}_a \in \mathbb{R}^d$ being the weight vector corresponding to action a, and $\mathbf{x}(s) \in \mathbb{R}^d$ being the feature vector for state s (obtained, say, via tile-coding). Also let $\mathbf{W} := [\mathbf{w}_1\ \ \mathbf{w}_2\ \ \cdots\ \ \mathbf{w}_{|\mathcal{A}|}]$ be a $d \times |\mathcal{A}|$ weight matrix. Further, overload the gradient notation, so that for a scalar function $\lambda(\mathbf{W})$, $\nabla_{\mathbf{W}}\lambda$ is a $d \times |\mathcal{A}|$ matrix defined as $[\nabla_{\mathbf{W}}\lambda]_{ij} = {}^{\partial\lambda}/_{\partial w_{ij}}$. Using this definition, $\nabla_{\mathbf{W}}\lambda = [\nabla_{\mathbf{w}_1}\lambda\ \nabla_{\mathbf{w}_2}\lambda\ \cdots\ \nabla_{\mathbf{w}_{|\mathcal{A}|}}\lambda]$, which in turn means that we can first calculate $\nabla_{\mathbf{w}_c}\mathcal{J}_\pi$ for each action $c \in \mathcal{A}$ and then stack these vectors to obtain $\nabla_{\mathbf{W}}\mathcal{J}_\pi$. The gradient update, for the weights $\mathbf{W}$, can be written as $\mathbf{W}^{\text{new}} = \mathbf{W}^{\text{old}} + \alpha\nabla_{\mathbf{W}}\mathcal{J}_\pi$ for some $\alpha > 0$. Using this formulation, let us now compute the expressions for the expected policy gradient, the regular estimator, and the alternate estimator.

First note that, using Proposition 2, for an arbitrary state s, and actions a and c we get that

$$\nabla_{\mathbf{w}_c}[\boldsymbol{\theta}(s)]_a = \nabla_{\mathbf{w}_c}\big(\mathbf{w}_a^\top\,\mathbf{x}(s)\big) = \mathbb{I}(c=a)\,\mathbf{x}(s). \tag{5.19}$$

Now for computing the expected policy gradient, put Eq. 5.7 in Eq. 5.5 to obtain

$$\begin{aligned}
\nabla_{\mathbf{w}_c}\mathcal{J}_\pi &= \sum_s \nu_\pi(s)\big[\nabla_{\mathbf{w}_c}\,\boldsymbol{\theta}(s)\big]\,\boldsymbol{\pi}(\cdot|s)\odot(\mathbf{q}_\pi(s,\cdot) - v_\pi(s)\mathbf{1}) \\
&= \sum_s \nu_\pi(s)\sum_a \nabla_{\mathbf{w}_c}[\boldsymbol{\theta}(s)]_a\pi(a|s)(q_\pi(s,a) - v_\pi(s)) \\
&= \sum_s \nu_\pi(s)\sum_a \mathbb{I}(c=a)\,\mathbf{x}(s)\pi(a|s)(q_\pi(s,a) - v_\pi(s)) \\
&= \sum_s \nu_\pi(s)\,\mathbf{x}(s)\pi(c|s)(q_\pi(s,c) - v_\pi(s)).
\end{aligned}$$

Now we stack together the vectors $\nabla_{\mathbf{w}_c}\mathcal{J}_\pi$ for all the actions $c \in \mathcal{A}$ to obtain

$$\nabla_{\mathbf{W}}\mathcal{J}_\pi = \sum_s \nu_\pi(s)\,\mathbf{x}(s)\Big(\boldsymbol{\pi}_{\mathbf{w}}(\cdot|s)\odot\big(\mathbf{q}_\pi(s,\cdot) - v_\pi(s)\mathbf{1}\big)\Big)^\top. \tag{5.20}$$

[8]This section is fairly straightforward and also tangential to the main presentation, and thus can be safely skipped.

Using chain rule of differentiation for vector variables (Proposition 4) and the expression for the derivative of the softmax policy with respect to action preferences (Eq. 2.12), we obtain

$$\begin{aligned}\nabla_{\mathbf{w}_c}\pi(a|s) &= \big[\nabla_{\mathbf{w}_c}\boldsymbol{\theta}(s)\big]\big[\nabla_{\boldsymbol{\theta}(s)}\pi(a|s)\big] = \sum_d \nabla_{\mathbf{w}_c}[\boldsymbol{\theta}(s)]_d \cdot \frac{\partial\pi(a|s)}{\partial[\boldsymbol{\theta}(s)]_d} \\ &= \sum_d \mathbb{I}(c=d)\,\mathbf{x}(s)\cdot\pi(a|s)\big[\mathbb{I}(d=a)-\pi(d|s)\big] = \pi(a|s)\,\mathbf{x}(s)\big[\mathbb{I}(c=a)-\pi(c|s)\big] \\ \Rightarrow\quad \nabla_{\mathbf{w}_c}\log\pi(a|s) &= \mathbf{x}(s)\big[\mathbb{I}(c=a)-\pi(c|s)\big].\end{aligned}$$

Again stacking the vectors $\nabla_{\mathbf{w}_c}\log\pi(a|s)$ for all actions $c\in\mathcal{A}$ gives us

$$\nabla_{\mathbf{W}}\log\pi(a|s) = \mathbf{x}(s)\big(\,\mathbf{e}_a - \boldsymbol{\pi}(\cdot|s)\big)^\top.$$

Finally, we can compute the regular estimator for online actor-critic. Using the expression for the regular online AC estimator (given in Eq. 2.32) for arbitrary $a\in\mathcal{A}$, $r\in\mathcal{R}$, $k\in\{0,1,\ldots\}$, and $s,s'\in\mathcal{S}$ we obtain

$$\begin{aligned}\mathbf{g}^{\text{REG-OAC}}(s,a,r,s',k) &:= \gamma^k\nabla_{\mathbf{W}}\log\pi(a|s)\big(r+\gamma v_\pi(s')-v_\pi(s)\big) \\ &= \gamma^k\big(r+\gamma v_\pi(s')-v_\pi(s)\big)\,\mathbf{x}(s)\big(\,\mathbf{e}_a-\boldsymbol{\pi}(\cdot|s)\big)^\top. \end{aligned}\tag{5.21}$$

If we stack the vectors $\nabla_{\mathbf{w}_c}[\boldsymbol{\theta}(s)]_a$ from Eq. 5.19 for different actions c, we would get

$$\nabla_{\mathbf{W}}[\boldsymbol{\theta}(s)]_a = \mathbf{x}(s)\,\mathbf{e}_a^\top.$$

Then the alternate estimator for online actor-critic (given in Eq. 5.17) can be computed as

$$\begin{aligned}\mathbf{g}^{\text{ALT-OAC}}(s,a,r,s',k) &:= \gamma^k\nabla_{\mathbf{W}}[\boldsymbol{\theta}(s)]_a\big(r+\gamma v_\pi(s')-v_\pi(s)\big) \\ &= \gamma^k\big(r+\gamma v_\pi(s')-v_\pi(s)\big)\,\mathbf{x}(s)\,\mathbf{e}_a^\top. \end{aligned}\tag{5.22}$$

5.5 Summary

In this chapter, we extended the alternate estimator to work with various policy gradient algorithms in the MDP setting. We did this by splitting the gradient of the policy with respect to the policy weights into a gradient of the action preferences with respect to the policy weights (that depends solely on the states) and the gradient of the policy with respect to the action preferences (that depends only on the actions given the state). After that, analogous to the bandit calculations, we analytically worked out the gradient of the policy with respect to the preferences for the softmax policy representation. In the end, we adapted the alternate estimator to multiple popular policy gradient algorithms.

Chapter 6

MDP Experiments with a Tabular Policy and Value Function Estimate

In this section, we demonstrate that the alternate estimator for MDPs enjoys similar benefits as the alternate estimator for bandits. We use the chain environment[1], shown in Figure 6.1, which is an episodic MDP where the expected rewards are zero everywhere expect at the rightmost transition. We train five different agents using REINFORCE (exact pseudocode given in Algorithm 2). All the agents maintain a tabular policy and, in some cases additionally, a tabular value function estimate. The policy is learned either using the expected PG, or using the REINFORCE algorithm with either the regular estimator (true v_π or a learned baseline) or the alternate estimator (again with true v_π or a learned baseline). The critic is estimated using Monte-Carlo sampling.

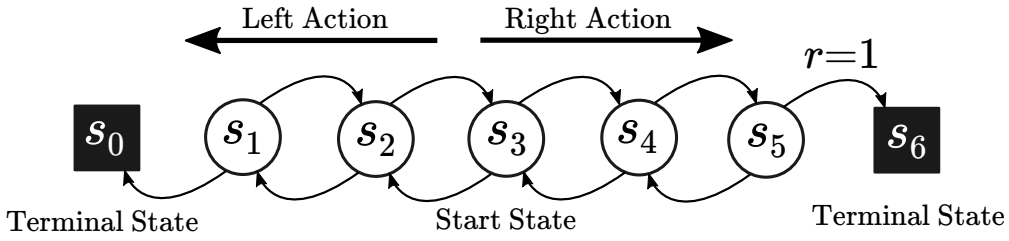

Figure 6.1: Chain Environment. At the beginning of each episode, the agent is reset to the state s_3. At each timestep, it can take the action `left` or `right`, which deterministically moves it into the corresponding next state. The episode ends when the agent reaches either of the two terminal states. The expected rewards are zero at all transitions except the one into the right most terminal state. Further, a reward noise $\epsilon \sim \mathcal{N}(0, 1)$ is added to the expected reward at each timestep.

All the methods were run for 100 episodes. The learning curves show the mean performance and standard error, and the sensitivity plots show the mean performance over the last 10 episodes. The

[1]Implementation available here: https://github.com/svmgrg/rl_environments/tree/main/LinearChain.

results were averaged over 150 independent runs. For getting a uniform policy initialization, we set $\theta_{\texttt{left}} = \theta_{\texttt{right}} = 0$ for all the five states. A saturated policy was obtained by initializing $\theta_{\texttt{right}} = 0$ and $\theta_{\texttt{left}} \in \{1, 2, 3\}$ depending on the degree of saturation, again for all the five states. The agents either used the true value function v_π (computed analytically) or learned a value function estimate b which was initialized to either zero, $+4$, or -4 for each state of the MDP. We swept over the policy stepsize $\alpha \in \{2^{-6}, 2^{-5}, \ldots, 2^1\}$ and the value function baseline stepsize $\beta \in \{2^{-4}, 2^{-3}, \ldots, 2^1\}$. We set the discount factor $\gamma = 0.9$. To reduce the runtime of the experiments, if the agent was unable to solve the task after 100 timesteps, we terminated the episode[2].

6.1 Performance of PG Estimators against Increasingly Saturated Policy Initializations

We begin by showcasing that the alternate PG estimator is competitive with the regular estimator in case of uniform policy initialization and superior to it in case of sub-optimally saturated policy initialization. These results are essentially identical to those from the bandit experiments (§4.1). Figure 6.2 shows the learning curves for the best performing parameter configuration (chosen from the sensitivity plots), and Figure 6.3 shows the stepsize sensitivity corresponding to the final performance for each parameter setting. Observe that irrespective of the policy saturation, the alternate estimator with baseline is vastly superior to all the methods (except expected PG), and that the alternate estimator with true v_π is a little better than the regular estimators. We attribute the superior performance of alternate estimator with baseline to the bias of the estimator combined with utilizing the noise in the returns, which probably allows it to have better exploration. From the sensitivity plots, we further see that as the initial policy is saturated more towards the left

[2]It might seem unsatisfying to artificially end an episode after 100 timesteps. However, not doing so resulted in some of the experiments running for about a million steps, which dramatically increased the runtime of the experiments. To put this number in perspective, an optimal policy can solve the chain environment in three steps. These million timestep long episodes occurred when the policy learned to go right with a high probability in one state and learned to go left in the very next state, effectively creating an unending loop between these two states.

We justify the 100 timestep cutoff by noting that the contribution of states after a hundred steps becomes negligibly small. This can be seen by focusing on the term γ^t in the expression for the REINFORCE estimator $\hat{g}^{\text{REINF}}$ (Algorithm 2): for our chosen discount factor, $\gamma^{100} = 0.9^{100} = 2.7 \times 10^{-5}$. We also note that in our experiments, we learned the value function using simple Monte-Carlo estimates and, contrary to the recommendation made by Pardo et al. (2018), did not bootstrap at the state where the episode was artificially timed out. We again justify this by the fact that $\gamma^{100} v_\pi(s_{\texttt{timeout}})$ is negligibly small compared to the returns observed in our experiments (see the scale of the y-axis in Figure 6.2) and is thus safe to ignore. Yet another argument in favor of keeping a timeout in our experiments is that, unlike timeouts with continuing MDPs where they are guaranteed to occur, in the chain environment the occurrence of the episodes being timed out decreases as the agent's policy improves.

Also note that we are using a discount factor $\gamma \neq 1$ for an episodic problem despite the fact that episodic problems do not necessarily require discounting. This choice was primarily made to justify timing out the episode after 100 steps. However, even if we ignore the timeout issue, keeping a discount factor around for episodic MDPs can be useful. It caps the maximum return at the fixed value of $\frac{r_{\max}}{1-\gamma}$, and helps us avoid situations such as the infinite loop between two states in the supposedly episodic chain MDP. Situations like these also violate the assumption (as discussed in Footnote 8 of §2.6) that undiscounted episodic MDPs should have a finite expected termination length.

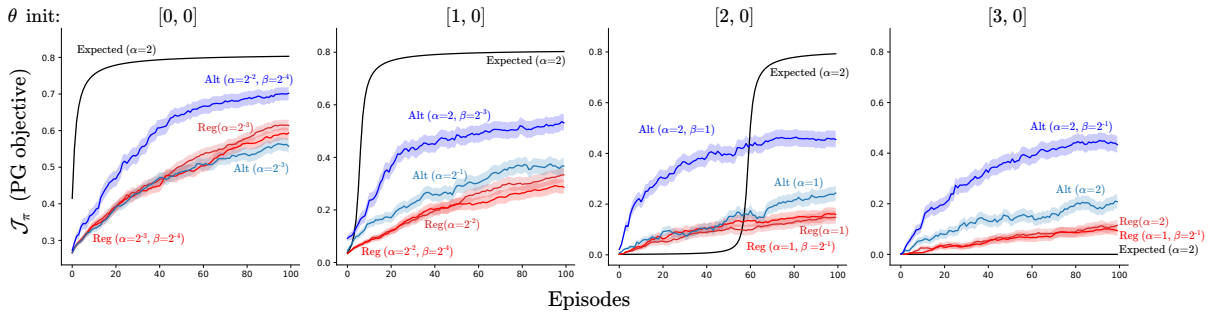

Figure 6.2: Learning curves for the REINFORCE agent with different estimators against increasingly saturated policy initializations on the five state chain environment. The figure header shows the action preference initialization for $\theta_{\texttt{left}}$ and $\theta_{\texttt{right}}$ respectively. The estimators which don't mention the baseline stepsize β used the true value function v_π, whereas the others learned a baseline using Monte-Carlo sampling. The baseline was initialized to zero for all the states.

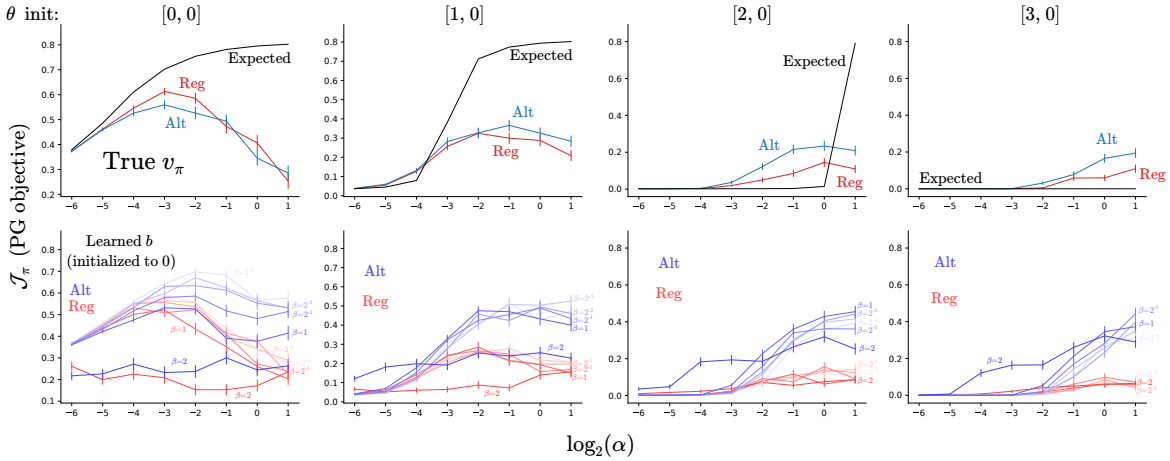

Figure 6.3: Sensitivity plots for different PG Estimators, showing the mean final performance during the last 10 episodes, against increasingly saturated policy initializations on the chain environment. The different columns correspond to the different degrees of policy saturations and the figure headers show the action preference initializations for the `left` and `right` actions respectively.

direction (in the chain environment), performance of all the methods worsens.

PG Estimators without Reward Noise

We now study the performance of different estimators when we remove the reward noise from the chain MDP. Figure 6.4 shows the learning curves and Figure 6.5 shows the parameter sensitivity plots for different estimators using REINFORCE on the chain MDP with reward noise $\epsilon = 0$. The results are somewhat similar to the bandit setting: in absence of reward noise, the performance of the alternate estimator drops with saturated policy initialization. However, there is one important

difference from the bandit setting. For MDPs, the alternate estimator finally cares about the noise in the return estimate which depends on both the reward noise and the sampling noise. Therefore, even without reward noise, in MDPs the alternate estimator can utilize the noise in estimating the returns from Monte-Carlo sampling. However, for the chain MDP, the transitions were deterministic and therefore the sampling noise was also quite less.

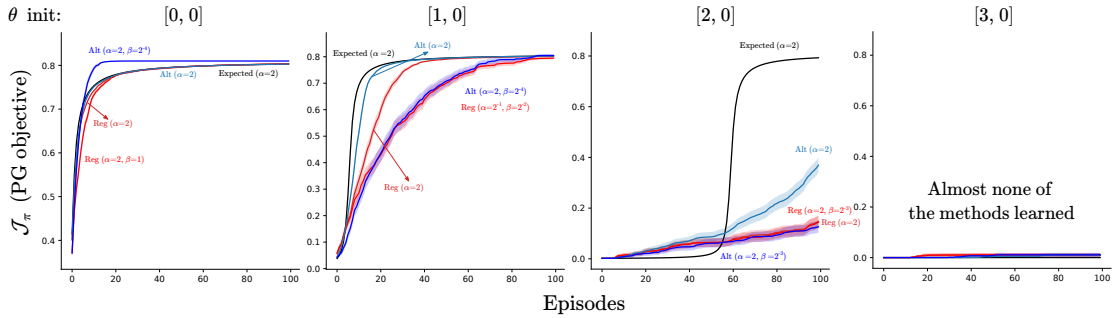

Figure 6.4: Learning curves for PG estimators without reward noise. The setting is same as Figure 6.2, except that the reward noise is set to zero. We see that, for saturated policies, the alternate estimator performs worse as compared to its performance in Figure 6.2 where it had reward noise, and **curiously** alternate with v_π performs better than alternate with a learned baseline.

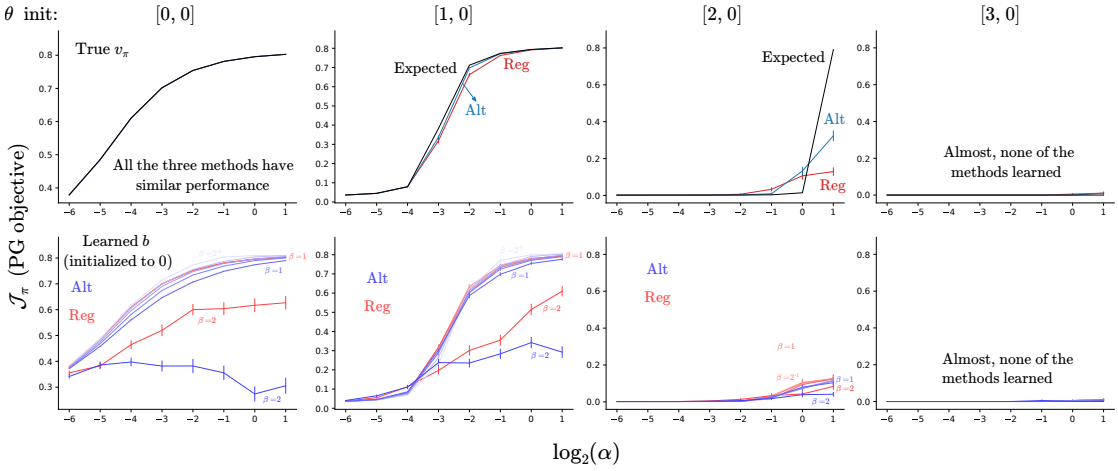

Figure 6.5: Sensitivity plots for PG estimators without reward noise. Setting is same as in Figure 6.3 except that there is no reward noise here.

6.2 Optimistic and Pessimistic Value Function Estimates

In this experiment, we study the performance of the alternate estimator when the value function baseline is initialized optimistically or pessimistically and then learned using Monte-Carlo sampling. Figures 6.6 and 6.7 show the parameter sensitivity plots for agents with optimistically ($b = +4$) or

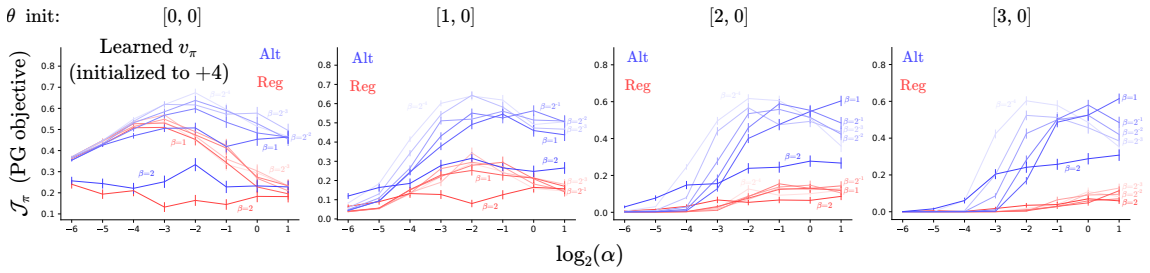

Figure 6.6: Sensitivity plots for an optimistically initialized value function estimate. The setting is similar to that in Figure 6.3, except that the baseline is initialized to +4.

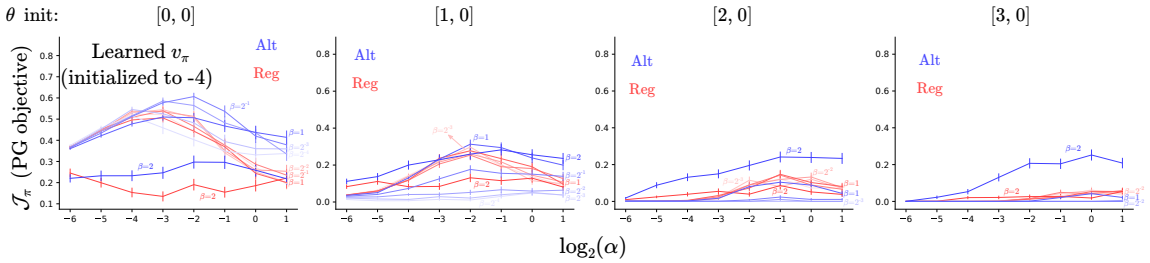

Figure 6.7: Sensitivity plots for a pessimistically initialized value function estimate. The setting is again similar to that in Figure 6.3 except that the baseline is initialized to −4.

pessimistically ($b = -4$) initialized baselines respectively, against different degrees of policy saturations. The results, which are yet again similar to the bandit setting, show that having an optimistic baseline significantly helped the alternate estimator; in particular, compare the performance of the alternate estimator on saturated policies with ($b = +4$, as shown in Figure 6.6) and without ($b = 0$, as shown in Figure 6.3) optimism. Whereas, a pessimistic baseline hurt its performance. Moreover, the alternate estimator with the optimistic baseline prefers smaller critic stepsizes (allowing it to enjoy the optimism for longer), and vice-versa for the pessimistic baseline.

Fixed Value Function Estimate

In this experiment, we fixed the value function estimate for both the estimators; which made the alternate, but not the regular, estimator biased. Figure 6.8 shows the sensitivity plots for the two estimators. This experiment is the MDP analog of the biased gradient bandit experiment (see Figure 4.8) and suggests that Theorem 1 might also hold for MDPs: since, we can show that the policy (for the optimistic case) converges to the fixed point $\pi^*(a) = \frac{1}{r(a)-b}\left(\sum_c \frac{1}{r(c)-b}\right)^{-1}$ given by Lemma 8.1. For MDPs, we need to compute this fixed point separately for each state: $\pi(\texttt{left}|s) = \pi(\texttt{right}|s) = 0.5$ for $s \in \{s_1, s_2, s_3, s_4\}$, and $\pi(\texttt{left}|s_5) = 0.43$ and $\pi(\texttt{right}|s_5) = 0.57$. Then, we analytically calculate $\mathcal{J}_{\pi^*} = v_{\pi^*}(s_3) = 0.28$ which approximately matches the value shown in Figure 6.8 (left) where the policy for the optimistic fixed baseline actually converged to; therefore,

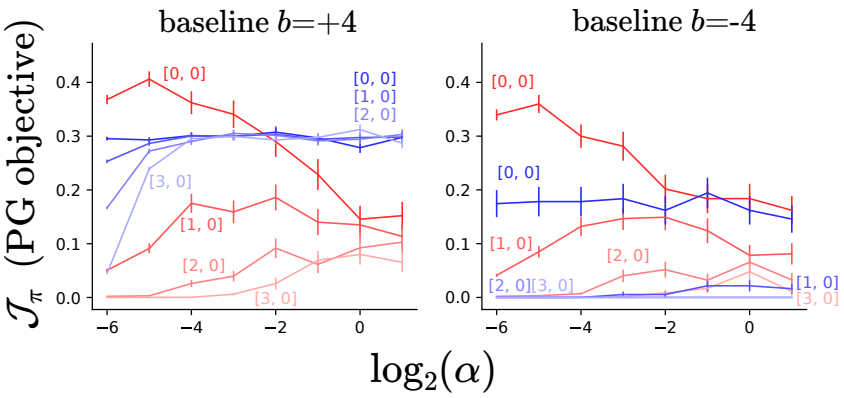

Figure 6.8: Bias in the alternate PG estimator for MDPs. Sensitivity plots for the REINFORCE agents using regular or alternate estimator with fixed baselines on the chain environment. The value function estimates were fixed to $+4$ (for left subplot) and -4 (for right subplot) and were not subsequently updated. The labels for the individual curves show the action preference initializations $[\theta_{\texttt{left}}, \theta_{\texttt{right}}]$ for all the five states.

our prediction matches with the actual results. For a pessimistic baseline, the policy converged to the nearest corner for each state (in this case action `left`) and obtained zero return.

6.3 Effect of adding Gradient Noise to the Estimators

This experiment studies the efficacy of randomly perturbing the PG estimates, with a normally distributed noise while updating the action preferences, in helping the agent escape saturated policy regions. The preferences, for a given state s and action a, were updated using $\theta_a^{\text{new}}(s) = \theta_a^{\text{old}}(s) + \alpha\big[g_a(s) + \xi\big]$ with $\xi \sim \mathcal{N}(0, \texttt{grad-noise})$. We swept over the noise magnitude $\texttt{grad-noise} \in \{0, 0.1, 0.5, 1, 2\}$. Both the estimators learned a value function estimate which was initialized to zero[3]. Figure 6.9 shows the sensitivity of the final performance for the two estimators.

These results show that adding gradient noise doesn't significantly help the regular estimator in dealing with saturated policies for MDPs: the performance of the regular estimator with gradient noise was only slightly better than its performance without gradient noise (cf. Figure 6.3). More importantly, the alternate estimator had a superior final performance compared to the regular estimator for both the uniform and the saturated policy initialization cases. This observation is in contrast to what we saw in the bandit case (§4.4), where the gradient noise seemed to help more. We explain this result by the fact that random noise merely increases the variance of the gradient estimate without providing any effective signal. And while this added variance helped the regular estimator in the simpler bandit problem, it is apparently not sufficient for the MDPs. The

[3]We do not present the results for expected PG and the regular and alternate REINFORCE estimators using the true v_π, since they were qualitatively similar to the results for the learned baseline case presented here.

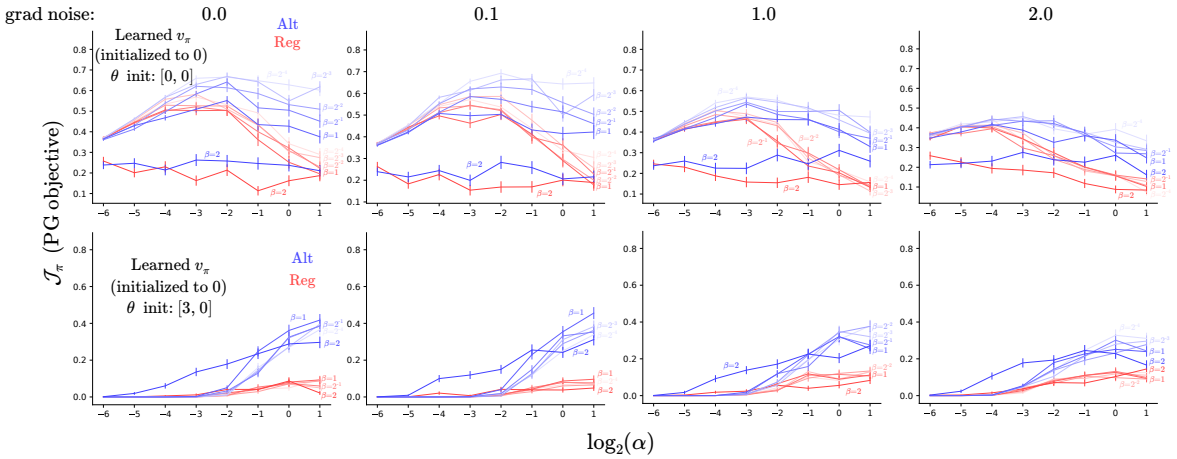

Figure 6.9: Adding gradient noise to the REINFORCE estimators. The top row shows the sensitivity plots for a uniform policy and the bottom row shows the sensitivity plots for a policy saturated as $\theta_{\text{left}} = 3$ and $\theta_{\text{right}} = 0$. Different columns correspond to different gradient noise magnitudes.

alternate estimator on the other hand relies on both the noise from the sampled return and also a much stronger signal from the bias in the critic estimate to escape the saturated policy regions.

6.4 A Uniform Policy Initialization can be Sub-optimal too!

In all the previous experiments, we found that an optimistically initialized baseline helped the performance of the alternate estimator (by making the policy more uniform). In this section, we consider an MDP, where a uniform policy is bad for exploration, and therefore an optimistic baseline by making the agent's policy more uniform hurts its performance. Figure 6.10 shows this MDP, which is a modified form[4] of the five state chain environment: instead of having just two actions, this modified chain has four actions for each state, three of which take the agent towards the left direction and the fourth towards the right direction.

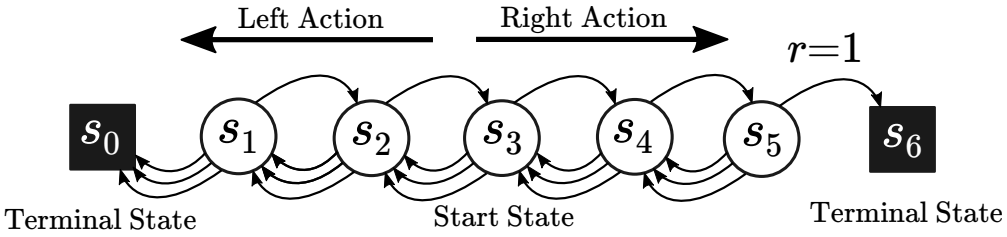

Figure 6.10: Linear chain environment where a uniform policy will explore poorly.

Figure 6.11 shows the sensitivity plots for the REINFORCE agents with the regular and al-

[4]This MDP was inspired from the "Vanishing gradient example" given in Chapter 12 of Agarwal et al. (2019).

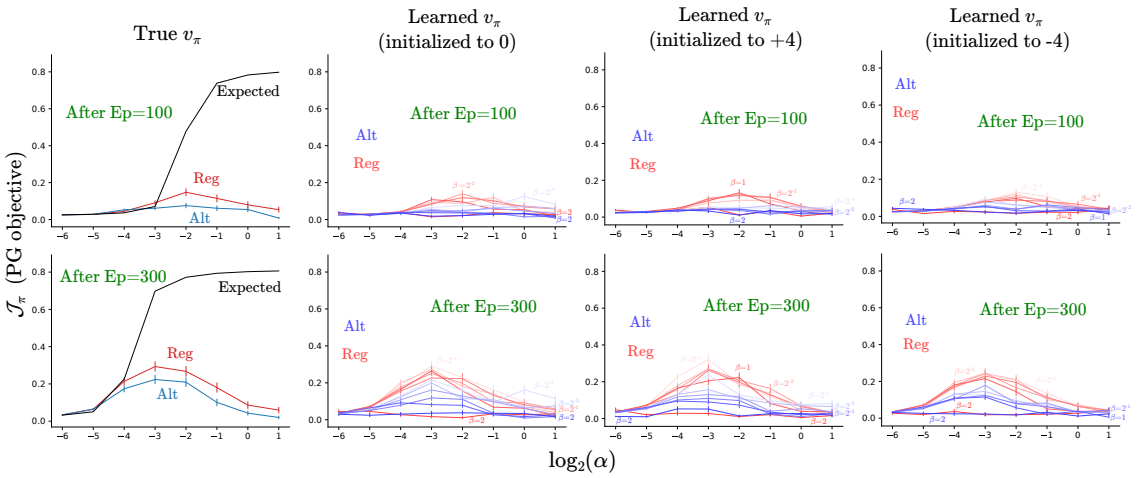

Figure 6.11: Sensitivity plots for different methods, initialized with a uniform policy, on the difficult chain environment (described in Figure 6.10). Top row shows the mean final performance (averaged over the last 10 episodes) of the estimators after 100 episodes and the bottom row shows the mean final performance (again averaged over the last 10 episodes) after 300 episodes. Different columns refer to either a true baseline or a learned baseline with different initializations. We see that even after 300 episodes, the algorithms are not able to solve the task (i.e. they are unable to achieve the maximum return of 0.81). Further, even with an optimistic baseline, the alternate estimator performs quite poorly as compared to the regular estimator.

ternate estimators for four different settings: using the true v_π, and a learned baseline initialized to either of 0, +4, or −4. The policy for all these settings was initialized uniformly, i.e. $\theta = 0$ for all states and action. The results show that in each case, the alternate estimator had a poorer performance as compared to the regular estimator. In particular, even an optimistically initialized baseline could not improve the alternate estimator's performance. We reconcile this fact by noting that in this problem the uniform random policy will actually lead to a sub-optimal exploration and an optimistic baseline will hamper the performance of the alternate estimator.

6.5 Summary

In this chapter, we demonstrated that the extension of the alternate estimator to the MDP case retained its favorable properties that we had identified in the case of gradient bandits. We ran experiments with the REINFORCE policy gradient algorithm and a tabular policy and value function estimate on the five state chain environment. Our results confirm that the alternate estimator can utilize the noise in the sampled returned estimates and also the bias in the value function estimate to escape sub-optimally saturated policies. Further, even for the case of uniform policies, we observed that the alternate method had a superior performance compared to the regular estimator on the chain MDP (except for the case of a pessimistically initialized baseline).

Chapter 7

Experiments in the MDP Setting with Function Approximation

In this chapter, we demonstrate that the desirable properties of the alternate estimator continue to hold with function approximation. We ran experiments with online actor-critic (see Algorithm 3) on various MDPs using linear function approximation (with tile-coding) and neural networks. The agent was trained using either the regular or the alternate estimator and a critic learned using TD-style updates; in these experiments, we do not have access to the underlying dynamics, and therefore we did not conduct experiments with the true value function v_π. We considered two different types of experiments: one with artificially introduced policy saturation, and another with non-stationary MDPs which naturally give rise to sub-optimally saturated policies. Experiments where we explicitly saturate the agent's policy at the beginning of learning are helpful in understanding the behavior of the estimators by providing a controlled environment. The non-stationary MDPs serve as a proof-of-concept that sub-optimally saturated policies are not merely pathological constructs, but can occur in practical problems that a user might care about.

7.1 Experimental Details for Linear Function Approximation

For experiments with linear function approximation, we used the environments[1] Acrobot (Sutton, 1996) and MountainCar (§4.3, Moore, 1990; Singh and Sutton, 1996), which are depicted in Figure 7.1. Both of these environments are episodic in nature with a continuous state space and a discrete action space consisting of three actions: `left`, `do-nothing`, and `right`. In Acrobot, the goal is to control an under-actuated double pendulum such that its end effector reaches a certain height, by

[1]We used the environment implementations available here: https://github.com/andnp/PyRlEnvs/tree/main/PyRlEnvs/domains. In this repository, MountainCar is referred to as GymMountainCar.

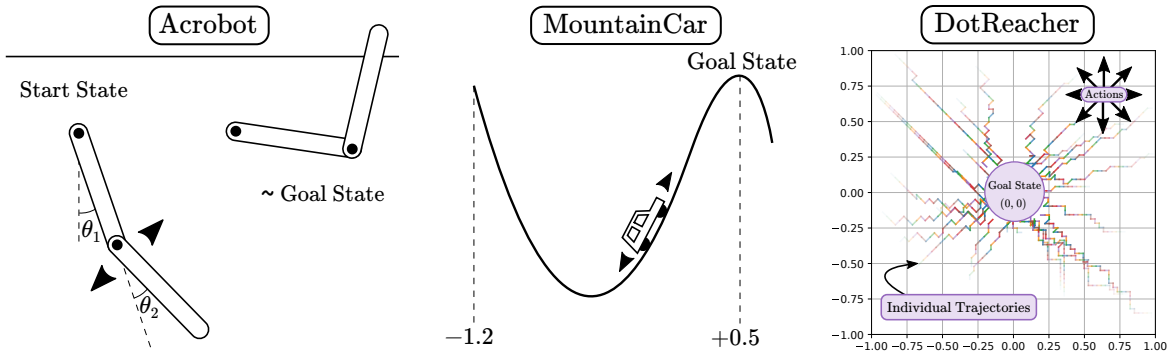

Figure 7.1: Schematic diagrams of Acrobot, MountainCar, and the DotReacher environments. For the DotReacher environment, we also plotted multiple trajectories for a particular instance of online actor-critic that learned to solve this task.

applying torque on the second joint. In MountainCar, the goal is to move an under-powered car from the bottom to the top of a hill. For both the environments, we fixed the discount factor $\gamma = 1$ and timed out the episode after 1000 timesteps if the agent was unable to solve it by then[2].

We performed sweeps over the policy stepsize $\alpha \in \{2^{-13}, 2^{-11}, \ldots, 2^5\}$ and the critic stepsize $\beta \in \{0.01, 0.05, 0.1, 0.5, 1, 2\}$. The state for Acrobot is $[\cos(\theta_1)\ \sin(\theta_1)\ \cos(\theta_2)\ \sin(\theta_1)\ \dot{\theta}_1\ \dot{\theta}_2]$, with $\dot{\theta}_1 \in [-4\pi, +4\pi]$ and $\dot{\theta}_2 \in [-9\pi, +9\pi]$. The state for MountainCar is $[\texttt{position}, \texttt{velocity}]$ (of the car), with $\texttt{position} \in [-1.2, +0.5]$ and $\texttt{velocity} \in [-0.07, +0.07]$. Both the policy (for details, see §5.4) and the value function used by the agent were linear in the state features. And the state features were constructed using tile-coding (§9.5.4, Sutton and Barto, 2018). In particular, we used four tiles across each dimension of the state space, and eight different tilings with randomly chosen offsets. We also added an "always on" bias feature and normalized the tile-coding features (by dividing them with nine) to have a unit sum.

7.2 Non-stationary MountainCar and Acrobot

This experiment investigates the performance of the online AC algorithm, with the regular and the alternate estimators, on non-stationary Acrobot and MountainCar. To induce non-stationarity

[2]For these experiments we bootstrap at the final state of the episode, i.e. if the episode times out, we bootstrap using $\hat{v}_\pi(s_{\text{timeout}})$, and if the episode actually ends, we use $v(s_\times) = 0$ for bootstrapping. This justifies the use of timeouts since (1) this process follows the recommendation made by Pardo et al. (2020), (2) this is an episodic task and as the agent's performance improves, the episodes will be timed out less, and (3) from our experiments, we observed that 1000 timesteps gave the agent enough time to encounter a true terminal state. Encountering true terminal states is important when learning with TD-style updates, since only at terminal state do we bootstrap from a true value (of zero); for all the other states, we bootstrap from an estimated value. Note that this argument is different from the numerical precision argument given in Footnote 2 of §6. Also note that with $\gamma = 1$ and a timeout of 1000, for the agents that are unable to solve the task within 1000 timesteps, the value function estimates will diverge to negative infinity; for instance, see the value of $\hat{v}_\pi(s_0)$ against time for the regular estimator in Figure 7.6.

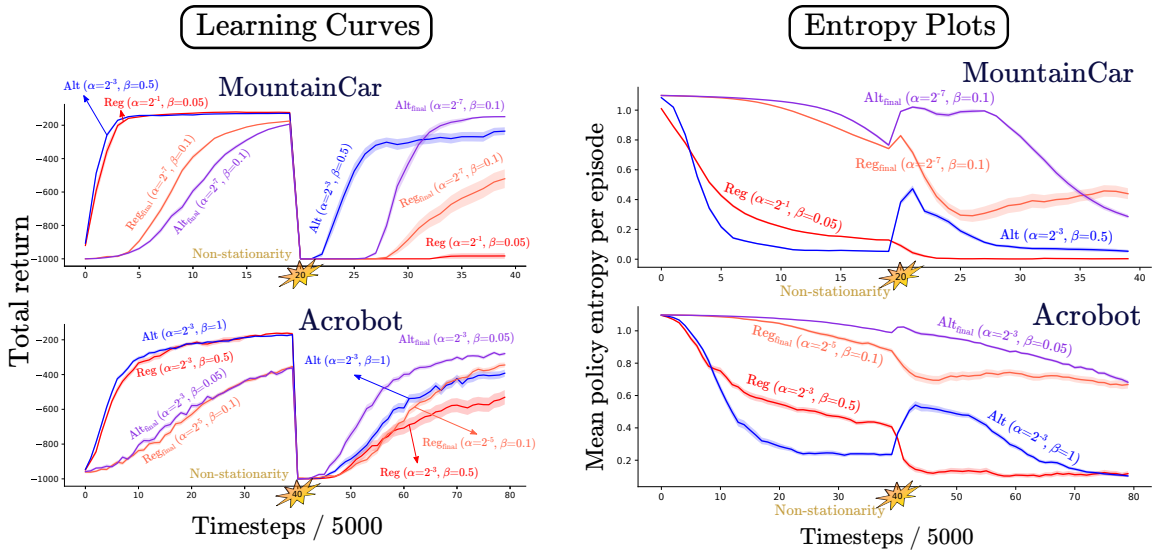

Figure 7.2: Learning curves and entropy plots for online linear AC with tile-coding on Acrobot and MountainCar. Both the estimators have one learning curve each for the best performing set of stepsizes right before the non-stationarity hit and at the end of learning. The right subplots show the entropy of the policy on the states encountered during each episode. All the curves show the mean values (thick line) and the standard error (shaded region) over 50 independent runs.

in either task, we switched the `left` and `right` actions after half-time. The policy was initialized uniformly by setting all the action preferences to zero. The critic weights were also initialized to zero. Each agent was trained for $400k$ timesteps for Acrobot, or $200k$ timesteps for MountainCar.

Figure 7.2 (left) shows the learning curves for the two estimators. For each estimator, we selected two different sets of stepsize configurations: one that had the best performance right before the non-stationarity hit, and another that had the best performance at the end of the experiment (in the figure, we denote this latter set by putting a `final` in the subscript of the corresponding estimators). For MountainCar, the alternate estimator was superior to the regular estimator for both sets of stepsizes. Remarkably, the regular estimator (red curve), with the best performing parameters at timestep $100k$, was unable to recover from the non-stationarity; whereas the alternate estimator (blue curve) was able to recover, despite having a similar performance as the regular estimator at the end of $100k$ timesteps. On Acrobot, the alternate estimator was again superior to the regular estimator, however the difference was less evident.

We attribute the superior performance of the alternate estimator to the bias in the critic estimate. For instance, consider what would happen in MountainCar for an online AC agent using a well performing stepsize configuration. At $100k$ timesteps, the agent would be able to learn a good policy and therefore the critic would learn to predict a return that is greater than -200. Now, when the non-stationarity hits, the agent's policy suddenly becomes sub-optimally saturated, and

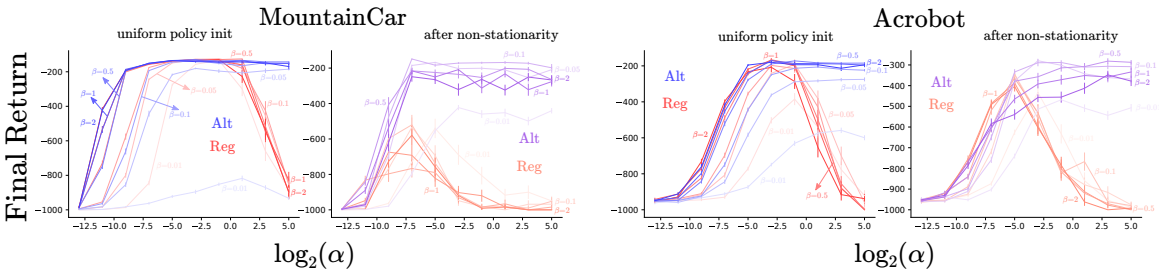

Figure 7.3: The sensitivity plots for online Actor-Critic with linear function approximation (using tile-coded state features) on MountainCar and Acrobot. The sensitivity plots show the mean performance during the last $5k$ timesteps, averaged over 50 independent runs.

thus the agent starts receiving returns much lower than -200. This in turn means that at this time, the critic estimate has become optimistic. As a result, for the alternate estimator, the critic would encourage exploration by pushing the policy towards a more uniform distribution. Figure 7.2 (right) corroborates this point: the policy entropy for the alternate, but not for the regular, estimator has a spike right around the timestep when the non-stationarity was introduced.

Figure 7.3 shows the parameter sensitivity plots for these tasks. From the sensitivity plots, we see that the alternate estimator is much less sensitive to the stepsizes and achieves a superior or competitive final performance compared to the regular estimator, both before and after the non-stationarity was introduced.

7.3 Experiments with Saturated Linear Softmax Policies

We now consider the performance of the two PG estimators on Acrobot and MountainCar with artificially saturated policies. Our results show the same trends for alternate and regular estimators in the linear function approximation setting as we saw for the bandit and the tabular setting. For obtaining saturated policies, we initialized the action preference corresponding to the `do-nothing` action with a value of either $0, 5, 10$, or 100 depending on the degree of policy saturation, whereas the preferences for the actions `left` and `right` were always initialized with zero. The baseline was initialized with different values ($0, +500$, or -500) imparting an optimistic or a pessimistic effect to the estimators. Each agent was trained for $50k$ timesteps for both the environments

Figure 7.4 shows the learning curves for for both the environments where the policy was initialized uniformly and the critic was initialized to zero. From this figure, we note that both the estimators were able to learn a good policy (a return of more than -200 is considered good on these environments). Further, the alternate estimator learned faster as compared to the regular estimator. We explain this behavior by noting that a baseline initialization of zero is effectively optimistic for a random policy (which will achieve large negative returns). Therefore, with an

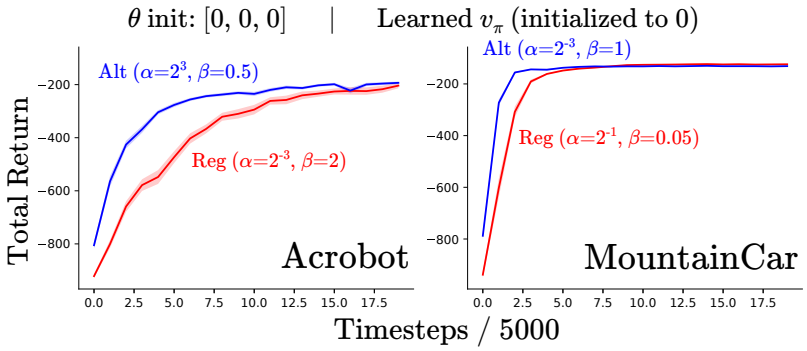

Figure 7.4: Learning curves for the best stepsize parameter configurations based on final performance on the two environments. The policy was initialized uniformly randomly and the baseline was initialized to zero. The curve show the mean performance and standard error over 50 runs.

optimistic baseline, the alternate baseline was probably able to get to the fixed point of the biased update at a fast rate. Once it was near this fixed point, as its critic estimate improved, it slowly moved towards the optimal policy. Whereas, with the regular estimator, the agent just slowly moved towards the optimal policy, and thus had a lower sample efficiency.

We show the sensitivity plots for these estimators, with different policy saturations and baseline initializations, on both Acrobot and MountainCar in Figure 7.5. The results follow similar trends that we observed for 3-armed bandits and the tabular chain environment. And impressively, these plots show that the alternate estimator can effectively escape saturated policies, even with a high policy saturation of $\boldsymbol{\theta}^{\text{init}} = [0\ 100\ 0]^\top$.

From the results in this figure, we observe that the alternate estimator had a lower parameter sensitivity and achieved a superior final performance compared to the regular estimator, for almost all the policy saturations and baseline initializations considered in this experiment; except for the (uniform policy and $b = -500$) initialization case, where the regular estimator had a better performance. We also observe that as the baseline was initialized to more negative values (to -500 instead of 0), the performance of the alternate estimator became worse. The baseline initialization affects the regular estimator much less. And as the policy initialization became more saturated, the performance of both the methods worsened. Another thing to note is that the regular estimator had a lower sensitivity to the critic stepsize β; the alternate estimator performed poorly for lower critic stepsizes than it did for higher critic stepsizes. We believe that the reason for this is that with lower critic stepsizes, the critic takes more time to track the value function due to which the alternate estimator remains biased for longer; and this probably affects the final performance in a negative way. We make one final comment about this experiment: the initializations $(0, +500$, and $-500)$ are all effectively optimistic for a saturated policy (which gets a return of less than -1000); this probably explains why the alternate estimator had such a good final performance, for most settings, in both the tasks.

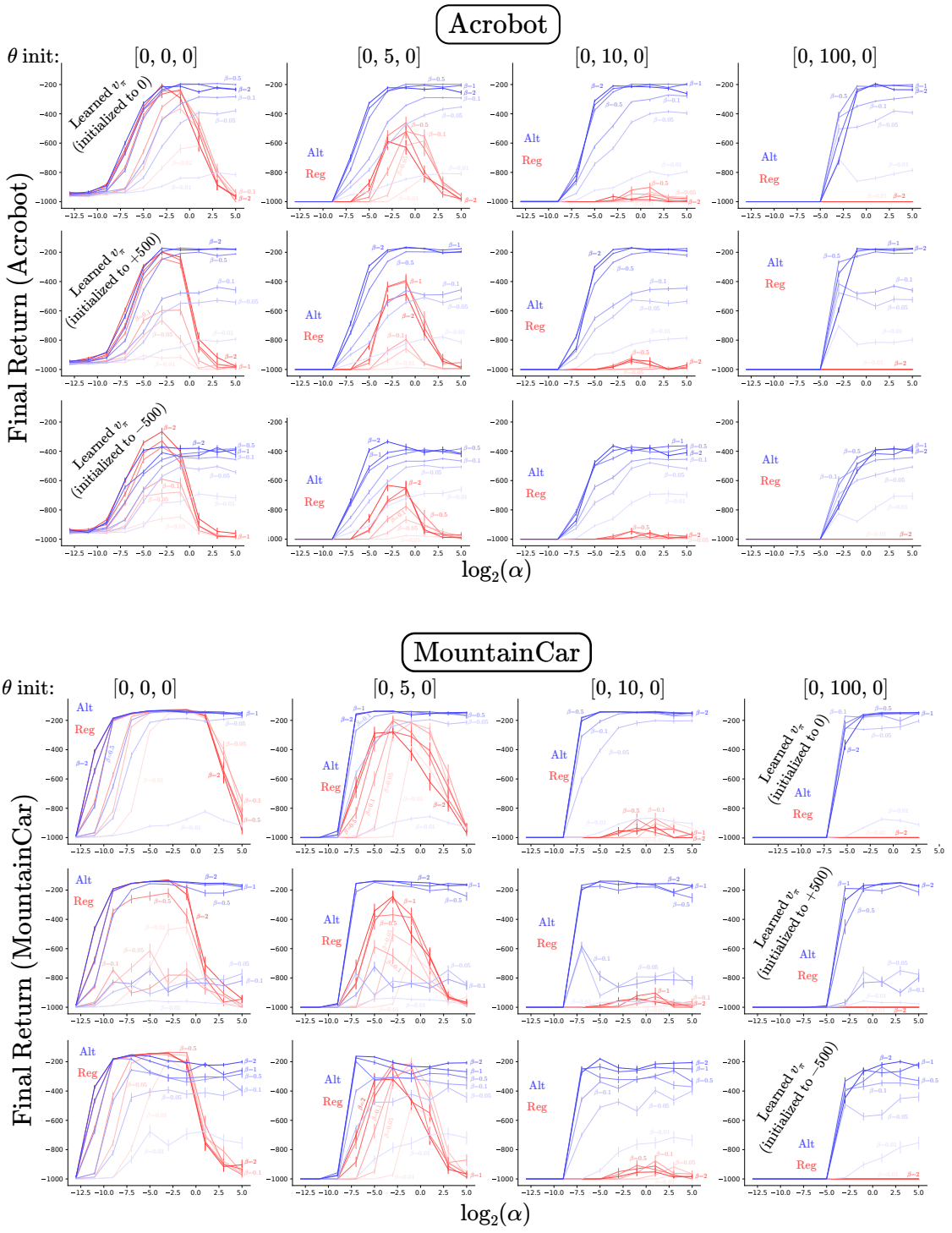

Figure 7.5: Parameter sensitivity plots for online AC with the two estimators on Acrobot (top) and MountainCar (bottom). The columns correspond to policies with different saturations and the rows correspond to differently initialized critic estimates. Each agent was run for $50k$ timesteps. The plots show the mean final performance during the last $5k$ steps, averaged over 50 runs.

7.4 Experiments on RealisticMountainCar

We now consider the performance of the two estimators on a difficult variant of the usual MountainCar environment which we call as the RealisticMountainCar[3]. Unlike the original environment that had an instantaneous dynamics model (§4.3, Moore, 1990), i.e. an acceleration applied to the car instantaneously changed its velocity, RealisticMountainCar introduces a delay between when the acceleration is applied and when the car's velocity increases, thereby making the dynamics more realistic. Moreover, as we find from our experiments, this delay makes the environment more difficult to solve for the regular PG estimator. The state-action space for this environment remains the same as the original MountainCar environment. The discount factor used was $\gamma = 1$ and the episode was cutoff after 1000 timesteps if the agent was unable to solve it by then. We performed sweeps over the policy stepsize $\alpha \in \{2^{-8}, 2^{-6}, \ldots, 2^{16}\}$ and the critic stepsize $\beta \in \{0.01, 0.05, 0.1, 0.5, 1, 2\}$. All the agents were trained for $100k$ timesteps and 50 independent runs.

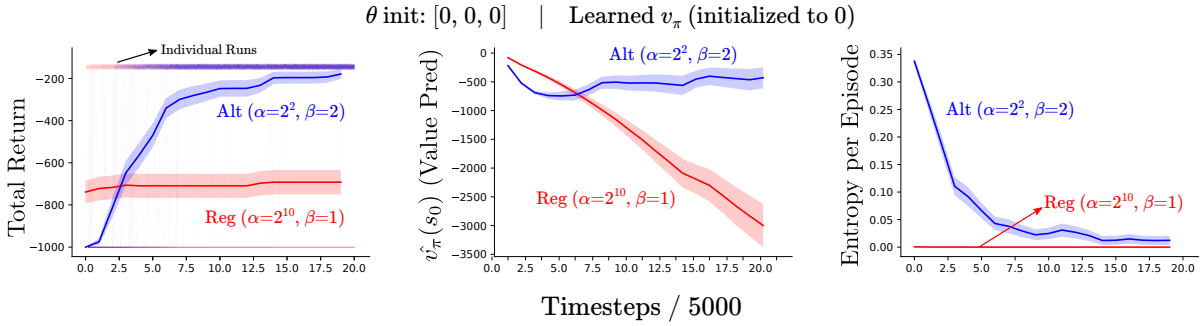

Figure 7.6: Learning curves for online AC on the RealisticMountainCar task for the best performing parameters. We also plot the value estimate of the initial state of the episode, and the mean policy entropy on the states encountered during different episodes. Even though the alternate estimator was able to learn an effective policy, the regular estimator failed to do so. Note that the regular estimator achieved its best performance at a very high stepsize that almost instantaneously pushed the policy in a highly saturated sub-optimal region; hence the learning curve for the regular estimator is flat (corresponding to an unchanging policy) and the entropy plot for the regular estimator has zero entropy throughout the learning period.

Figure 7.6 shows the performance curves, the value function estimate of the initial states, and the policy entropy for the two estimators as the agent learned. From these results, we see that the alternate estimator was effectively able to solve this task, whereas the regular estimator was unable to learn a good policy. From the individual runs for the regular estimator, we see that some of them resulted in a good policy (achieving a return of > -200), whereas most of them didn't learn (achieving the maximum possible return of -1000), giving it a net average return of about

[3]RealisticMountainCar was proposed by Andy Patterson. We used the implementations available here: https://github.com/andnp/PyRlEnvs/tree/main/PyRlEnvs/domains. In this repository, MountainCar is referred to as GymMountainCar and RealisticMountainCar is referred to as MountainCar.

-700. We also included the plots for $\hat{v}_\pi(s_0)$ where $\hat{v}_\pi$ is the critic estimate and s_0 is the start state observed at the beginning for each episode. From these plots, we see that the value function of the regular estimator diverges to negative infinity because we used a discount factor $\gamma = 1$ and the agent, for most runs, was unable to solve this task within 1000 timesteps.

We show the sensitivity plots for the mean final performance of the two estimators during the last $5k$ timesteps in Figure 7.7. These results show that the alternate estimator learns a good policy for a large range of stepsizes. Most curiously, for this experiment, the alternate estimator with a baseline initialization of $+500$ had a worse performance than the initialization of 0 and -500. We attribute this result to the fact that overcoming an optimistic initialization of $+500$ takes the longest time (during which the estimator remains biased). Further, since 0 and -500 are already optimistic for this problem, they had a good performance as well. This observation leads us to conclude that the critic for the alternate estimator should be initialized optimistically, but not too optimistically.

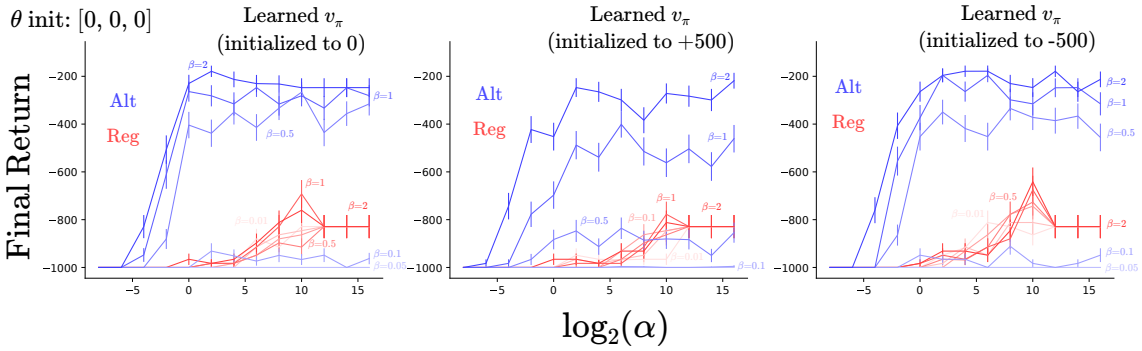

Figure 7.7: Parameter sensitivity of different PG estimators with online AC on RealisticMountainCar. The policy was initialized uniformly, and the value function was initialized to different value in each column. We see that the alternate estimator is fairly robust across parameters whereas the regular estimator exhibits poor performance consistently across different parameter settings.

7.5 Details about Experiments with Neural Networks

Now we study the online AC algorithm with a neural policy and critic. We use an environment called DotReacher. We again experimented separately with artificially saturated policy initializations and a non-stationary environment. In either case, we found that the results were similar to the linear function approximation case.

DotReacher[4], shown in Figure 7.1 is a grid world environment where the agent is spawned uniformly randomly in a 2×2 unit squared arena. For the stationary version of DotReacher, the

[4]DotReacher was proposed by Rupam Mahmood. We used the implementation available here: https://github.com/svmgrg/rl_environments/tree/main/DotReacher.

goal is to reach the $(0, 0)$ state within a tolerance of 0.1. The agent has 9 different actions: eight corresponding to eight different directions and one for no movement. Each action deterministically moves the agent 0.03 units in the corresponding direction. The agent gets a reward of -0.01 until it reaches the goal state. We fixed $\gamma = 1$ and the episode cutoff length as 1000. For the non-stationary variant, we initially set the goal state to $(-1, -1)$ and then shifted it to $(+1, +1)$ after $50k$ steps.

For both the experiments involving the stationary the non-stationary DotReacher, we swept the policy stepsize $\alpha \in \{2^{-17}, 2^{-15}, \ldots, 2^{-3}\}$, the critic stepsize $\beta \in \{2^{-17}, 2^{-15}, \ldots, 2^{-7}\}$, and used the Adam optimizer (Kingma and Ba, 2014) with the default PyTorch (Paszke et al., 2019) configuration[5]. We maintained two separate neural networks (both with `ReLU`) for the policy and the critic: the policy network dimensions were $2 \times 10 \times 10 \times 9$ and the critic network dimensions were $2 \times 10 \times 10 \times 1$. For saturating the policy, we increased the action preference corresponding to the no movement action by modifying the bias units of the last layer of the neural network.

7.6 Non-stationary DotReacher

Figure 7.8 shows the learning curves and the stepsize sensitivity plots for the online AC algorithm with the two estimators on the DotReacher environment. The policy was initialized uniformly and the critic estimate was initialized to zero. The results follow the same pattern as the experiments on the non-stationary Acrobot and MountainCar. The alternate estimator is still competitive or superior than the regular estimator, although the difference is less pronounced.

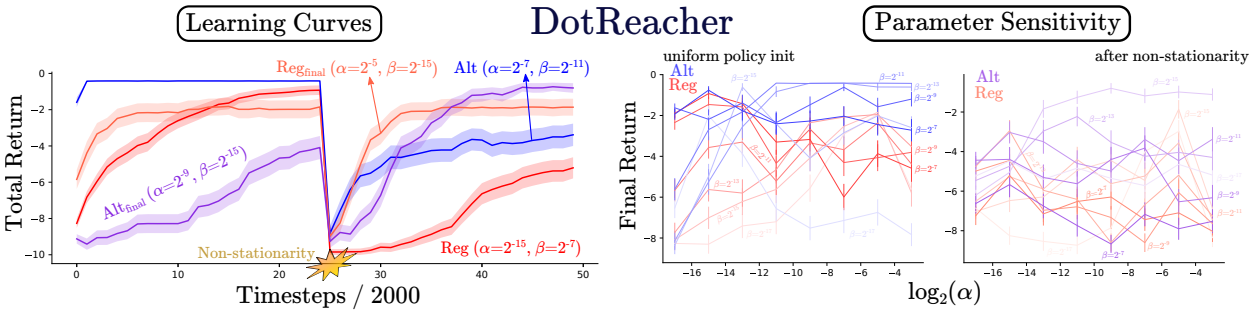

Figure 7.8: **(Left)** Learning curves for online Actor-Critic with neural networks on DotReacher ($100k$ timesteps, 50 runs) for two sets of best performing parameter configuration: one that obtained the best performance right before the non-stationarity occurred, and the other that obtained the best final performance at the end of learning. **(Right)** The parameter sensitivity plots showing the mean final performance (during the last $2k$ timesteps) at the end of $50k$ timesteps and the mean final performance at the end of $100k$ timesteps. The mean final performance was computed by averaging over the last 2000 timesteps either at half-time or at the end of training.

[5]Refer here: `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`.

7.7 Experiments with Saturated Neural Policies on DotReacher

We now consider the experiments with artificially induced policy saturations on the DotReacher environment. Figure 7.9 shows the sensitivity plots for both the estimators for different policy saturations and critic initializations. In this problem, we consider a return of more than -1 as being good. The results are same as those discussed in §7.3: the alternate estimator had a superior performance and exhibited less sensitivity to the stepsize α as compared to the regular estimator. Interestingly, for these results, we observe that for lower amounts of policy saturations ($\theta_{\text{no-action}} \in \{5, 10\}$), the regular estimator could learn policies almost as good as the alternate estimator. But for the highest saturation case, only the alternate estimator was able to escape the saturation and obtain high returns. We also see that an optimistic baseline initialization (for this problem, $b = 0$ and $b = +25$) helped the alternate estimator achieve a good performance, whereas the pessimistic baseline initialization ($b = -25$) made the performance of the alternate estimator even worse than the regular estimator (except in the case of super high policy saturation).

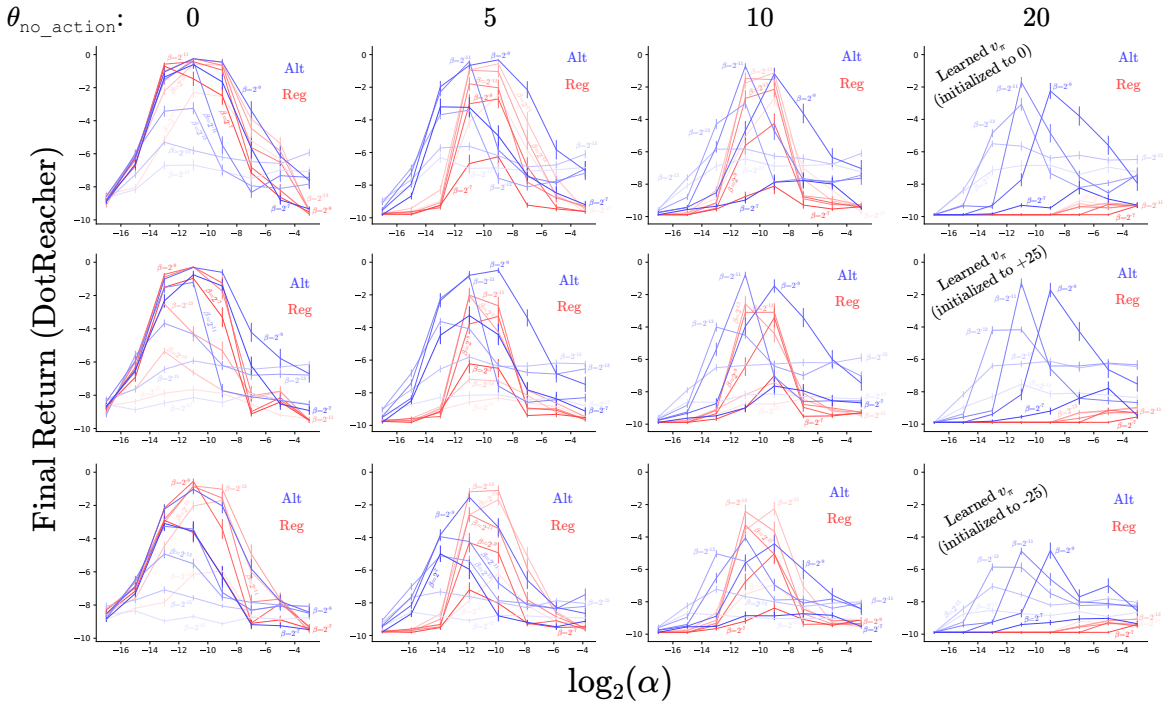

Figure 7.9: Online AC + neural networks with different baseline initializations and policy saturations on the DotReacher task. The plots show the mean final performance during the last $5k$ timesteps of the agent's $50k$ timestep lifetime. The figure header shows the action preference initialization for the no movement action; the rest of the preferences were initialized to zero. Different rows show differently initialized value function estimates (0, +25, or -25). We observe that for almost all the settings, the alternate estimator performed better than the regular estimator in recovering from saturated policies, especially for higher saturation magnitudes.

7.8 A Digressive Remark: Softmax Policies Learn even with Extremely Large Stepsizes

In this section, we mention a peculiar observation: online AC with the alternate estimator was able to learn a good policy even with extremely large stepsizes. The reader might have noticed from the parameter sensitivity plots presented in this chapter that the alternate estimator has a fairly good performance for a big range of the policy stepsize α. In particular, the sensitivity plots, given in Figure 7.5, for Acrobot and MountainCar show that for the stepsize range we tried (with the maximum value of α being $2^5 = 32$), the alternate estimator had a good performance throughout. More interestingly, for RealisticMountainCar (sensitivity plots given in Figure 7.7), we tried stepsizes as large as $2^{16} = 65536$, and the alternate estimator still received a fairly high final return (only slightly lower than the what the alternate estimator with the best parameter setting achieved and much higher than what the regular estimator received). Even if we take into account the normalization factor from tile-coding (which was 9), the effective stepsize still remains quite high: $2^{16}/9 = 7282$. To put this number in perspective, the default choice of the stepsize for training neural networks with the Adam optimizer (Kingma and Ba, 2014) is 0.0003. One might wonder, whether this only happens for online AC with linear function approximation setting, and the answer is no. We show the stepsize sensitivity plots for REINFORCE on the chain environment (the same experiment as discussed in §6.1 and §6.2) with a wide range of the policy stepsize in Figure 7.10. And these plots display a similar pattern: the alternate estimator learns for extremely large values of the policy stepsize as well. The first two rows of Figure 7.10 are exactly the same as Figure 6.3 except that it includes a larger sweep for the policy stepsize $\alpha \in \{2^{-6}, 2^{-4}, \ldots, 2^{14}\}$. Similarly, the third and fourth rows are identical to Figure 6.6 and Figure 6.7 respectively. We verified that none of these results were due to bugs in our experiments and now discuss some plausible explanations for this behavior.

We ascribe these observations to the nature of softmax function and the policy optimization problem over the action preferences. Note that (1) the softmax function transforms any set of action preferences, irrespective of how large they are, to a proper probability distribution; and (2) for softmax policies, the optimal point which maximizes the PG objective, lies at infinity on the action preference landscape. The second point is true because for MDPs, the optimal policy is greedy, which requires the action preference for the greedy action to be infinitely large. This makes the optimization problem over the action preferences very different from, say, an unconstrained convex optimization problem. For convex functions, an update step with a high stepsize, even in the correct direction, can make the function value worse than its present value. In contrast, a larger stepsize with softmax functions will only drive the function value closer to the optimum at a faster rate. We illustrate this point in Figure 7.11. Therefore, as long as the update direction is correct, having large stepsizes should not be a concern for softmax policies.

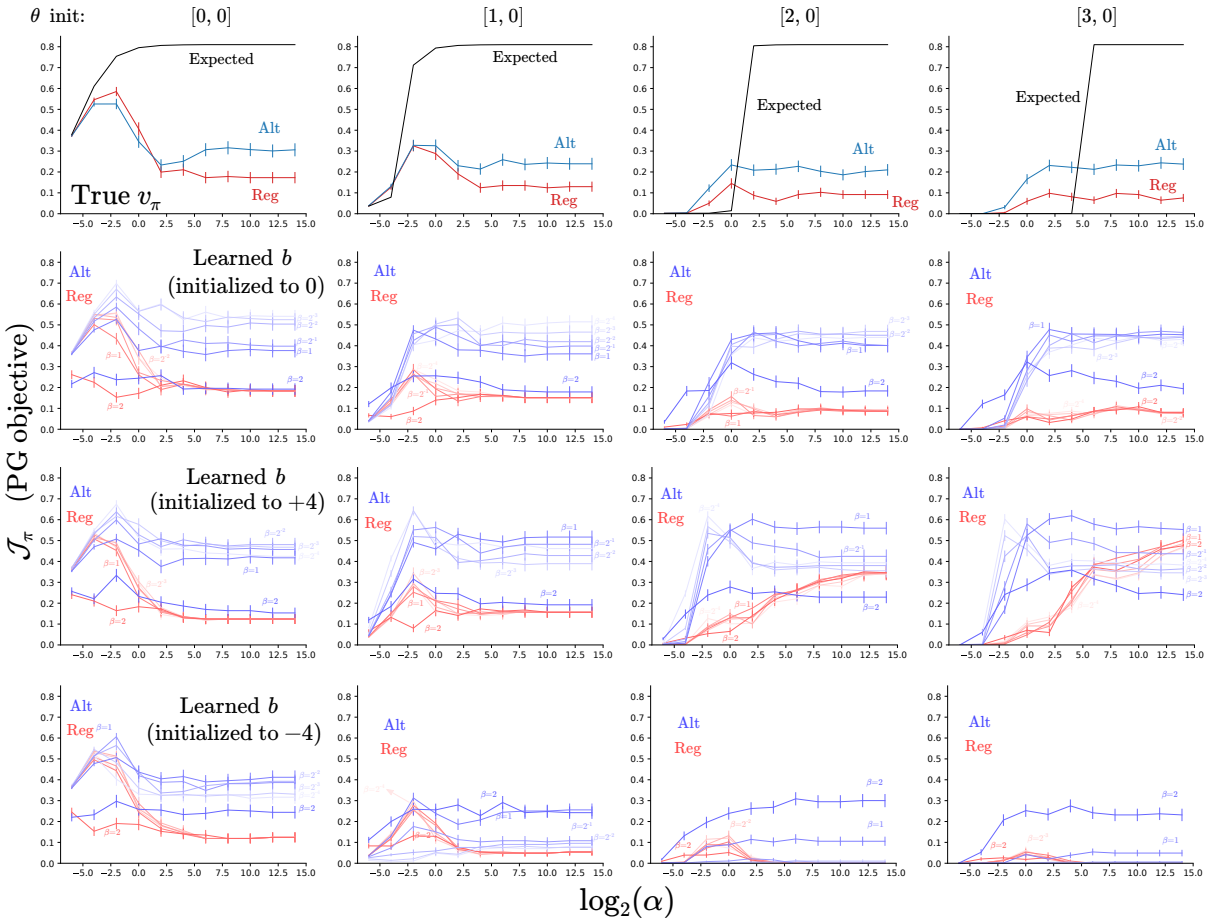

Figure 7.10: Sensitivity plots showing the final performance of **tabular** REINFORCE on the chain environment. The different columns correspond to an increasing degree of policy saturation and the figure headers show the action preference initializations for the `left` and `right` actions respectively. The first row shows the results for the true baseline v_π, and the next three rows show results for a learned baseline, which was initialized to either zero, $+4$, or -4 at the beginning of learning.

For concreteness, let us consider the stepsize sensitivity behavior of expected PG in Figure 7.10: a larger stepsize for expected PG update resulted in a faster convergence to the optimal policy and consequently a higher final return. From gradient based optimization literature, we know that the expected (policy) gradient is the the update direction that results in the fastest local improvement in the (policy) objective, and therefore using the reasons given in the previous paragraph, we would expect a large stepsize to drive the policy towards this local optimum faster. We can also observe this trend for the expected gradient bandit algorithm. Figure 3.5 (bottom-left) shows that for a bandit problem, a large policy update stepsize should still make the policy converge towards the optimal (top) corner, albeit via a different route. But for that bandit problem, there is only one optimum that is also globally optimal. Also, intuitively, the policy gradient update in the (tabular) MDP setting can be considered as (kind of) making separate gradient bandit updates for each state.

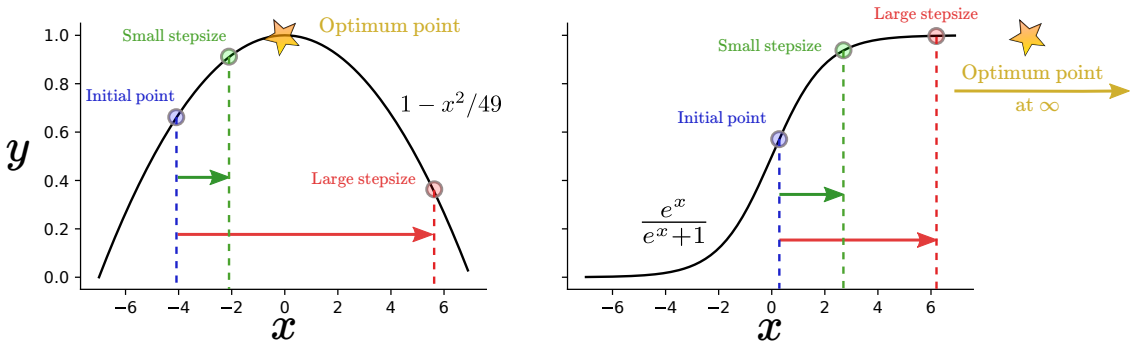

Figure 7.11: Effect of the stepsize magnitude on optimizing a function value. We consider the optimization problem on a convex function and the logistic function. Note that the logistic function $y = \frac{e^x}{e^x+1}$ is essentially equivalent (except for a translation along the x-axis) to the softmax function $\pi(a) = \frac{e^{\theta_a}}{e^{\theta_a}+\sum_{c\in\mathcal{A}\setminus a} e^{\theta_c}}$. This figure shows that given the correct update direction (positive direction for both of these problems), an update step with a very large stepsize on the convex function can make the function value worse. Whereas, in the case of a softmax function, the larger the stepsize, the more the function value increases with a single update step.

However, due to the sampling noise, the stochastic PG estimators might not always give an update direction that points towards the local optimum. Consequently, a very large stepsize could saturate the policy in a sub-optimal manner. We posit two reasons for why the alternate estimator might be better suited to handle very large policy stepsizes than the regular estimator. The first reason is that the alternate estimator can handle and escape sub-optimally saturated policies much better than the regular estimator. With very large stepsizes, the action preferences for both the estimators would be driven to large values and the policy would becomes saturated. For such cases, the regular gradient estimator becomes vanishingly small: $\hat{\mathbf{g}}^{\text{REG}}(S, A) = (R-b)(\mathbf{e}_A - \boldsymbol{\pi}) \approx 0$, since near the corners $\mathbf{e}_A \approx \boldsymbol{\pi}$ with a high probability. Therefore, even with a very large stepsize α, the net update to the action preferences $\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta}^{\text{old}} + \alpha\, \hat{\mathbf{g}}^{\text{REG}}(S, A)$ remains small, and the policy doesn't change much. In contrast, for the alternate estimator $\hat{\mathbf{g}}^{\text{ALT}}(S, A) = (R - b)\, \mathbf{e}_A$, irrespective of the degree of policy saturation, the estimator remains non-zero, which when combined with a large stepsize can be effective in changing even very large action preferences. Although this explanation used the gradient bandit estimators, similar arguments should hold for the MDP policy gradient estimators as well.

The second point is that, since the alternate estimator updates the preference corresponding to only the sampled action, there is much less generalization in the preference values across actions when using the alternate estimator. In contrast, the regular estimator updates all the action preferences and therefore has a high generalization across the action space. This distinction becomes important when the agent updates the policy using an incorrect direction and a large stepsize. With the alternate estimator, only some action preferences for some states get saturated, whereas with

the regular estimator, all the preferences for a set of states get saturated. This generalization argument is also supported by the experiments presented in this document. With large stepsizes, the performance of the estimators, especially the regular version, becomes worse as we increase the generalization of the policies: compare the performance of the two estimators in Figure 7.10 (tabular policy: no generalization); Figures 7.5 and 7.7 (linear policy with tile-coding: generalization across 8 tilings); and Figure 7.9 (neural policy: high generalization). With neural networks, none of the estimators performed well with large stepsizes because updating even a single action preference (in the wrong direction) changes all the weights lying in the initial layers of the network, ultimately affecting all the action preferences.

One experiment to test the above hypothesis could be to run online AC with both the estimators on Acrobot and MountainCar with a tabular policy. This could be achieved with a tile-coding that has super-fine meshes (that is a large number of tiles per dimension) and only a single tiling. If the performance of the regular estimator with high stepsizes improves as compared to Figure 7.5, then that would serve as a strong verification for this hypothesis. Unfortunately, a detailed investigation of this phenomenon is out of scope of this work, and therefore we do not pursue it any further.

7.9 Summary

In this chapter, we demonstrated that the alternate estimator works well with the online Actor-Critic algorithm and function approximation on a wide range of tasks[6]. In fact, for most of our settings, the alternate estimator had a strictly superior performance to the regular estimator. We also studied a particular type of non-stationary environments where the baseline estimate becomes optimistic after the non-stationarity hits, allowing the alternate estimator to adjust to the changes in the environment; in contrast, the regular estimator takes much longer to improve its policy.

[6]It might seem that RealisticMountainCar and DotReacher are two obscure environments that we chose only because they helped motivate our case. However, this is not true. We chose RealisticMountainCar because it is a more realistic version of a classic control problem. By modeling the delay between when the action is taken and when its outcome is visible to the agent, it mimics real world problems more closely than other environments. The fact that the alternate estimator has a much superior performance than the regular estimator on this task was a mere coincidence. We chose DotReacher because it is an easy to understand task which the online AC algorithm with a neural policy can quickly solve, using just online incremental updates, without requiring access to a replay buffer. And, as the reader might have noticed, the difference between the performance of the two estimators in DotReacher is actually quite small.

Chapter 8

Conclusions

In this thesis, we introduced and analyzed an alternate policy gradient estimator for softmax policies that can utilize the reward noise and the bias in the critic to escape sub-optimally saturated regions in the policy space. This estimator was derived by analytically computing the gradient of the softmax policies in the policy gradient theorem. The resulting expression for the alternate estimator is a very simple modification over the expression for the popularly used regular PG estimator. And interestingly, this simple modification gives the alternate estimator certain properties that we found to be quite useful in dealing with saturated policies.

First of these properties is that the noise in the reward signal gives the alternate estimator a non-zero variance at the corners of the probability simplex. This allows it to conduct a random walk type motion and potentially overcome policy saturation. The second property is that when this estimator is used with a value function estimate, i.e. a critic, it becomes biased. With an optimistic baseline, the biased expected update, corresponding to the alternate estimator, pushes the policy towards a relatively uniform distribution with a strong signal, which again allows it to escape saturated policies. Later, as the critic estimate improves, this bias disappears, and the alternate estimator updates the policy towards an optimum point. On the flip side, with a pessimistic baseline, the biased update pushes the policy towards saturation (possibly sub-optimal) with an equally strong signal, which might result in poor performance.

A third property of the alternate estimator is that it only updates the softmax action preferences corresponding to the sampled action. This is unlike the regular estimator which updates the action preferences corresponding to all the actions. Moreover, if we utilize this property, we can design a gradient bandits algorithm employing the alternate estimator to have a per-timestep computational complexity that is logarithmic in the number of actions. This is strictly faster than the complexity of the gradient bandits algorithm using regular estimator, which is linear in the number of actions. When the action space is large, this difference can become substantial.

We studied all these properties theoretically for the multi-armed bandit framework. We also conducted an extensive empirical analysis with different magnitudes of artificially induced policy saturations and baseline initializations on multi-armed bandits, tabular MDPs, and MDPs with function approximation. In our experiments, we used a diverse range of algorithms such as gradient bandits, REINFORCE, and online Actor-Critic with tabular, linear, and neural policies. We also showcased the efficacy of the alternate estimator on non-stationary problems, where the policies naturally become sub-optimally saturated. These experiments helped verify the properties we discussed above, and demonstrated that the alternate estimator has a superior (or competitive) performance compared to the regular estimator in a variety of situations and can work well with different PG algorithms and various function approximation schemes.

From our empirical results, we found that the alternate estimator works really well with policies that are quite saturated (but still have a non-zero support over all the actions) and with baselines that are optimistically initialized. This combination, of a saturated policy and an optimistically initialized value function baseline, is particularly well suited for MDPs where a more uniform policy is good for exploration. More importantly, this combination organically arises in certain non-stationary examples where, due to a change in the environment dynamics, the agent's behavior becomes sub-optimal and it starts receiving rewards which are lower compared to what its baseline predicts. Unsurprisingly, for MDPs where uniform policies explore poorly, this combination results in a bad performance for the alternate estimator. We also believe that the signal offered by the alternate estimator is much stronger than the signal offered by the regular estimator. Therefore, we conjecture that with a suitable critic stepsize, the alternate estimator can exhibit a faster convergence compared to the regular estimator by: (1) initially exploiting its bias to quickly move towards a more uniform policy (which can be, depending on the magnitude of the optimistic baseline, close to the optimum point), and (2) later when the critic estimate improves, using the unbiased updates to move towards the optimal policy.

The major weakness of the alternate estimator is that with pessimistic baselines, it pre-maturely saturates the policy to distributions which can be sub-optimal. In contrast, pessimistic baselines affect the regular estimator in a much milder way making it a better choice for this setting. Also, the combination of a saturated policy and a pessimistically initialized baseline can occur naturally in certain non-stationary examples, where the agent starts getting rewards which are higher than what its baseline predicts. One example is a two-armed bandit problem where the rewards for both the arms increase after a certain time in such a way that the optimal action switches from one arm to the other. Further, for problems which are not too difficult and the initial policy provides good exploration, the regular estimator should perform better than the alternate estimator, because it has a lower variance. Another case where the regular estimator can be superior is, when the critic stepsize is too low and thus the baseline takes a long time to track the true value function. This makes the alternate estimator biased and changes the point where the policy would converge to,

which depending on the problem can be catastrophic. Whereas, the regular estimator, even in such cases, remains unbiased.

To conclude, the alternate estimator makes the policy gradient algorithms more robust to saturated softmax policies, and works pretty well in a particular class of problem settings. Notably, this estimator makes existing PG methods more viable for certain non-stationary problems, and by extension for many practical real-life control tasks.

8.1 Limitations of this Project and Future Directions

In this thesis, we provided a sufficient proof-of-concept that motivates a further study of the alternate estimator. However due to the nature of this estimator, the scope of such a study will inherently be limited to discrete softmax policies and the log-likelihood family of PG estimators. Let us now discuss the limitations of the analysis conducted in this thesis and possible future works which can address them.

We conducted a very limited theoretical analysis of the alternate estimator on the multi-bandit setting, where our results were asymptotic in nature. It would be useful to extend this analysis to MDPs. A finite time analysis could also be incredibly fruitful, because based on our intuition and some empirical evidence, we suspect that the alternate estimator might have a faster convergence rate compared to the regular estimator. Another direction is to study, possibly employing a two-timescale analysis (for instance, see Wu et al., 2020), how a changing baseline with the alternate estimator affects the dynamics of policy optimization (also see Chung et al., 2021). Also, we solely analyzed the expected updates corresponding to the regular and the alternate estimators. But, it seems that some of the most interesting properties of these estimators only surface when we look at the stochastic updates. Therefore, extending the analysis to include stochasticity (see Mei et al., 2021) could yield important insights as well. Comparing the alternate estimator with policy gradient methods that use entropy regularization also seems highly relevant.

It might also be interesting to see how the property of the alternate estimator, that it only changes a single action preference for each update, could be used to come up with computationally faster policy gradient algorithms. In this thesis, we answered this question for the simplest case of gradient bandit algorithm. And Young (2019) investigated this question in connection with reservoir sampling. Another direction could be to combine the alternate estimator with techniques used in natural language processing to speed up the learning of softmax classifiers[1]. Yet another approach could be to consider the alternate estimator with neural policies, and update only a subset of the neural network weights connected to the action preference being modified at that time. This approach would ensure that only a small number of weights are updated at each timestep, but on

[1]For instance, see *candidate sampling* (https://www.tensorflow.org/extras/candidate_sampling.pdf).

average the whole network gets updated, albeit, at a slower rate.

The empirical analysis undertaken in this project was limited in certain aspects. For instance, we only considered multi-armed bandit problems with a normally distributed noise. Possibly, other bandit problems (see Lattimore and Szepesvári, 2020) can yield additional insights into the behavior of these estimators. Also, we only studied a single type of tabular MDP: the five state chain (and its difficult variant). The chain MDP is a great environment for running quick experiments whose results are relatively easy to reason about. However, it hides interesting issues in policy optimization that other tabular environments help bring out: such as CliffWorld which has the presence of multiple sub-optimal policies (Example 6.6, Sutton and Barto, 2018), or DeepSeaTreasure (Vamplew et al., 2010; Osband et al., 2020) which is a difficult exploration problem.

We also did not study the batch PG algorithms such as PPO, TRPO, or A2C (Mnih et al., 2016). And we had good reasons for avoiding them: typical implementations (especially those which perform well) of batch algorithms have a huge number of interacting parts (such as large neural networks, replay buffers, mini-batch updates, and complicated optimizers), which introduce tangential and confounding factors making a careful empirical analysis much more difficult, compute intensive, and time consuming. Having said that, an important future work would be to study the alternate estimator with these algorithms on large MDPs. Another important set of experiments would be to compare the performance of the alternate method on non-stationary tasks and with saturated policies against techniques (mentioned in §1.1) such as those which add entropy regularization or explorative bonuses to policy gradient methods, or with alternative policy parameterizations such as the escort transform.

References

Agarwal, A., Jiang, N., Kakade, S. M., Sun, W. (2019). Reinforcement Learning: Theory and Algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Technical Report.*

Agarwal, A., Kakade, S. M., Lee, J. D., Mahajan, G. (2021). On the theory of policy gradient methods: Optimality, approximation, and distribution shift. *Journal of Machine Learning Research, 22*(98), 1-76.

Ahmed, Z., Le Roux, N., Norouzi, M., Schuurmans, D. (2019). Understanding the impact of entropy on policy optimization. In *International Conference on Machine Learning* (pp. 151-160).

Auer, P., Cesa-Bianchi, N., Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning, 47*(2), 235-256.

Barnard, E. (1993). Temporal-difference methods and Markov models. *IEEE Transactions on Systems, Man, and Cybernetics, 23*(2), 357-365.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R. (2019). Dota 2 with Large Scale Deep Reinforcement Learning. *arXiv preprint arXiv:1912.06680.*

Bertsekas, D. P., Tsitsiklis, J. N. (2000). Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization, 10*(3), 627-642.

Bhatnagar, S., Sutton, R. S., Ghavamzadeh, M., Lee, M. (2009). Natural actor-critic algorithms. *Automatica, 45*(11), 2471-2482.

Bresler, G., Jain, P., Nagaraj, D., Netrapalli, P., Wu, X. (2020). Least Squares Regression with Markovian Data: Fundamental Limits and Algorithms. *arXiv preprint arXiv:2006.08916.*

Chung, W., Thomas, V., Machado, M. C., Le Roux, N. (2021). Beyond variance reduction: Understanding the true impact of baselines on policy optimization. In *International Conference on Machine Learning* (pp. 1999-2009).

Ciosek, K., Whiteson, S. (2020). Expected Policy Gradients for Reinforcement Learning. *Journal of Machine Learning Research, 21*(52).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009). *Introduction to algorithms.* MIT press.

Degris, T., White, M., Sutton, R. S. (2012a). Off-policy actor-critic. *arXiv preprint arXiv:1205.4839.*

Degris, T., Pilarski, P. M., Sutton, R. S. (2012b). Model-free reinforcement learning with continuous action in practice. In *American Control Conference (ACC)* (pp. 2177-2182). IEEE.

Ghosh, D., C Machado, M., Le Roux, N. (2020). An operator view of policy gradient methods. *Advances in Neural Information Processing Systems, 33.*

Greensmith, E., Bartlett, P. L., Baxter, J. (2004). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Journal of Machine Learning Research, 5*(9).

Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., Levine, S. (2018). Learning to walk via deep reinforcement learning. *arXiv preprint arXiv:1812.11103.*

Hennes, D., Morrill, D., Omidshafiei, S., Munos, R., Perolat, J., Lanctot, M., Gruslys, A., Lespiau, J.B., Parmas, P., Duenez-Guzman, E., Tuyls, K., (2019). Neural replicator dynamics. *arXiv preprint arXiv:1906.00190.*

Huang, J., Jiang, N. (2020). From importance sampling to doubly robust policy gradient. In *International Conference on Machine Learning* (pp. 4434-4443).

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

Kingma, D. P., Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Konda, V. R., Tsitsiklis, J. N. (2000). Actor-critic algorithms. In *Advances in neural information processing systems* (pp. 1008-1014).

Lafferty, J., Liu, H., Wasserman, L. (2008). Concentration of measure. *Available online: http://www.stat.cmu.edu/~larry/=sml/Concentration.pdf.*

Lan, Q., Mahmood, A. R. (2021). Model-free Policy Learning with Reward Gradients. *arXiv preprint arXiv:2103.05147.*

Lattimore, T., Szepesvári, C. (2020). Bandit algorithms. Cambridge University Press.

Li, G., Wei, Y., Chi, Y., Gu, Y., Chen, Y. (2021). Softmax policy gradient methods can take exponential time to converge. *arXiv preprint arXiv:2102.11270.*

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. *arXiv preprint arXiv:1809.07731.*

Mei, J., Xiao, C., Szepesvári, C., Schuurmans, D. (2020a). On the global convergence rates of softmax policy gradient methods. In *International Conference on Machine Learning* (pp. 6820-6829).

Mei, J., Xiao, C., Dai, B., Li, L., Szepesvári, C., Schuurmans, D. (2020b). Escaping the Gravitational Pull of Softmax. *Advances in Neural Information Processing Systems, 33.*

Mei, J., Dai, B., Xiao, C., Szepesvári, C., Schuurmans, D. (2021). Understanding the Effect of Stochasticity in Policy Optimization. arXiv preprint arXiv:2110.15572.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928-1937).

Moore, A. W. (1990). Efficient memory-based learning for robot control.

Neu, G., Jonsson, A., Gómez, V. (2017). A unified view of entropy-regularized markov decision processes. *arXiv preprint arXiv:1705.07798.*

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

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvári, C., Singh, S., Van Roy, B., Sutton, R., Silver, D., Van Hasselt, H. (2019). Behaviour suite for reinforcement learning. *arXiv preprint arXiv:1908.03568.*

Pardo, F., Tavakoli, A., Levdik, V., Kormushev, P. (2018). Time limits in reinforcement learning. In *International Conference on Machine Learning* (pp. 4045-4054).

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., Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems, 32*, 8026-8037.

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

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

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889-1897).

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

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning.*

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simoyan, K., Hassibis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815.*

Singh, S. P., Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Machine learning, 22*(1), 123-158.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning, 3*(1), 9-44.

Sutton, R. S. (1996). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 1038-1044.

Sutton, R. S., McAllester, D. A., Singh, S. P., Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems* (pp. 1057-1063).

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., Wiewiora, E. (2009). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *International Conference on Machine Learning* (pp. 993-1000).

Sutton, R. S., Barto, A. G. (2018). *Reinforcement Learning: An Introduction,* Second Edition. MIT Press.

Tang, Y., Valko, M., Munos, R. (2020). Taylor expansion policy optimization. In *International Conference on Machine Learning* (pp. 9397-9406).

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58-68.

Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E. (2011). Empirical evaluation methods for multiobjective reinforcement learning algorithms. *Machine learning, 84*(1), 51-80.

Vaswani, S., Bachem, O., Totaro, S., Mueller, R., Geist, M., Machado, M. C., Castro P.B., Roux, N. L. (2021). A functional mirror ascent view of policy gradient methods with function approximation. *arXiv preprint arXiv:2108.05828.*

Watkins, C. J., Dayan, P. (1992). Q-learning. *Machine learning, 8*(3-4), 279-292.

Weaver, L., Tao, N. (2013). The optimal reward baseline for gradient-based reinforcement learning. *arXiv preprint arXiv:1301.2315.*

White, M. (2017). Unifying task specification in reinforcement learning. In *International Conference on Machine Learning* (pp. 3742-3750).

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

Wu, Y., Zhang, W., Xu, P., Gu, Q. (2020). A finite time analysis of two time-scale actor critic methods. *arXiv preprint arXiv:2005.01350.*

Young, K. (2019). Learning What to Remember: Strategies for Selective External Memory in Online Reinforcement Learning Agents. *University of Alberta, M.Sc. Thesis.*