Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for a degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Jafer Hasnain April 10, 2024

Reinforcement Learning with Manifold Optimization

By

Jafer Hasnain

Elizabeth Newman, Ph.D
Adviser

Department of Mathematics

Elizabeth Newman, Ph.D
Adviser

Matthias Chung, Ph.D
Committee Member

Lars Ruthotto, Ph.D
Committee Member

2024

Reinforcement Learning with Manifold Optimization

By

Jafer Hasnain

Elizabeth Newman, Ph.D
Adviser

An abstract of
a thesis submitted to the Faculty of
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2024

Abstract

Reinforcement Learning with Manifold Optimization
By Jafer Hasnain

Reinforcement learning is a rapidly advancing field which in recent years has expanded out of its niche origins to become one of the most potent paradigms available within machine learning. The explosion of deep learning in the past decade has only furthered this growth, leading an acceleration in the development of state-of-the-art RL methods for highly complex use cases. Simultaneously, the study of information geometry has expanded the understanding of the geometric spaces traversed by RL algorithms as the field matures. The first part of this work provides an introduction to reinforcement learning from its first principles, followed by an exploration of two categories of differential/information geometric methods applied to RL. The experiments in this work aim to broaden the understanding of this manifold structure and draw intuitive conclusions regarding the utility of such techniques for improving RL methods.

Reinforcement Learning with Manifold Optimization

By

Jafer Hasnain

Elizabeth Newman, Ph.D
Adviser

A thesis submitted to the Faculty of
Emory College of Arts and Sciences of Emory University
in partial fulfillment of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2024

Acknowledgments

I greatly appreciate the patience, perspectives, enthusiasm, encouragement, flexibility, challenging ideas, and overall positivity from Dr. Elizabeth Newman throughout this project, which has become the best learning experience I have had during my time at Emory. I would also like to thank Dr. Matthias Chung and Dr. Lars Ruthotto for their swift agreement and enthusiasm to take part in my committee.

Contents

List of Figures

Chapter 1

Introduction

This work explores the field of reinforcement learning from a geometric perspective. Chapter 2 introduces reinforcement learning through its historical origins from the field of optimal control, followed by a construction of foundational RL theory based on the framework of Markov Decision Processes. This chapter concludes with an exposition of policy-gradient methods as the basis for the majority of state-of-the-art algorithms used today. The first portion of Chapter 3 aims to explore the effects of imposing a manifold structure implied from the action space on the semi-gradient SARSA algorithm with the goal of developing a method of curvilinear descent upon its parameters, concluding with an experimental implementation of the developed methods and discussion of the results. The second portion of Chapter 3 draws from the field of information geometry in order to formalize the geometric structure of statistical manifolds and examine the derived methods for RL that have appeared in recent years, as well as explore the difficulties, complications, and considerations that must be made.

The ultimate objective of this work is to provide a motivation for the development of reinforcement learning methods that leverage the geometry of the parameter spaces they traverse.

Chapter 2

Background

This chapter serves as a brief introduction to reinforcement learning (RL) — including the terminology, notation, and methods relevant for the subsequent chapters. Specifically, the development of ideas in this chapter is meant to motivate and elucidate the concept of RL with function approximation, which serves as the basic framework for all of the following material in this work.

2.1 Introduction to Reinforcement Learning

Reinforcement learning (RL) is a paradigm within machine learning that concerns the following question: How can an agent within an environment maximize its expected reward? More specifically, the objective of RL is to ascertain the optimal *policy* — i.e. the best action with respect to reward — for the agent in each possible state of the environment. Evidently, such a broad question opens the door for the application of this framework to a huge class of problems, particularly those that can be formulated in the language of optimal control. As will be shown, the generality of this framework gives way to its extensibility, where a prudent use of assumptions, approximations, and heuristics allows agents to be trained in even the most treacherous environments.

2.1.1 History and examples

The earliest theoretical underpinnings of reinforcement learning come from the study of optimal control where, for a dynamical system, the goal is to obtain a control policy such that an objective function is maximized [38]. In the mid-1950's, mathematician Richard Bellman formulated such problems in a language that later became ubiquitous in modern RL. In particular, he introduced a *principle of optimality*, which provides a necessary condition for an optimal policy as having an optimal substructure, resulting directly in his groundbreaking method of *dynamic programming* [7]. His formulation of decision processes centers around ascribing a "value" to each state as the optimal expected payoff from beginning in that state. With dynamic programming, the *Bellman equation* for discrete-time problems (and the similarly named *Hamilton-Jacobi-Bellman equation* for continous-time problems) provide a direct way to obtain the optimal value function for a system via the convergent process of *value iteration*. The discrete-time case — more precisely described in terms of *Markov decision processes* — is of particular relevance to reinforcement learning, where it serves as the fundamental framework through which problems are formulated.

In 1960, engineer Ron Howard extended Bellman's value iteration to introduce *policy iteration*, which extended the understanding of the intimate relationship between policy evaluation and improvement. Indeed, most modern reinforcement methods can be framed in the context of *general policy iteration* [14]. A 2002 article by Howard retrospectively describes the development of policy iteration as in the realm of operations research (in fact, the method was developed for the Sears' catalogue mailing system), as well as the general excitement around the MDP framework at the time [15]. The advancements made with Markov decision processes in the context of optimal control, however, still lacked the "learning" aspect crucial in the modern study.

Sutton [38] describes the origin of this aspect as the early exploration of "trial-

and-error" methods in pursuit of artificial intelligence. While Deep RL — i.e. RL with neural networks — is a much later development, Minsky's SNARCs (Stochastic Neural Analog Reinforcement Computers) provide one of the earliest computational attempts for trial-and-error learning. Furthermore, the 1960s saw the groundbreaking work of British researcher Donald Michie on trial-and-error learning in the form of his famous MENACE (Matchbox Educable Noughts and Crosses Engine) for tic-tac-toe [27]. Michie's BOXES algorithm used for MENACE was also used to build the GLEE (Game Learning Expectimaxing Engine) method for the cart-pole i.e. pole balancing problem, a test environment that will be encountered later in this work. Similarly groundbreaking was Arthur Samuel's 1959 program for playing checkers, which utilized techniques that bear resemblance to later *temporal-difference* (TD) methods. The use of board games as a test environment and backdrop against which major RL breakthroughs are made is a constant theme throughout the history of the field.

In the early-1980's, key developments by Richard Sutton and Andrew Barto provided the synthesis of the aforementioned developments with conceptual tools in neuroscience and animal psychology, eventually leading to the development of temporal-difference learning. The advent of TD methods can be identified as a recognizable turning point from the disparate threads of work in optimal control and trial-and-error towards a more unified branch of machine learning — one notably distinct from the supervised/unsupervised learning dichotomy. Crucially, temporal-difference methods improved upon the Monte-Carlo methods of the time by introducing the concept of *bootstrapping* — i.e. on-line improvement of predictions as the agent progresses within an episode. The gap between Monte-Carlo and TD methods was further blurred by Sutton's development of TD(λ) [37], which was utilized by Gerald Tesauro in conjunction with an artificial neural network for TD-Gammon (for backgammon). The success of TD-Gammon in 1992 was a clear demonstration of the capability of RL to meet and even surpass the capabilities of humans in certain environments.

Christopher Watkin's introduction of *Q-learning* in his 1989 PhD thesis represented a clear advancement in the direction of off-policy learning, a distinction that will be explored in a following section [42].

The methods as described thus far are overwhelmingly — in their purest form — *tabular* methods for problems modeled as discrete MDPs, and thus suffer from the "curse of dimensionality" (coined by Bellman [7]). The inescapable constraint on computability raised by the massive state-action spaces encountered in real-world problems has been recognized from the beginning — indeed by Bellman himself [7]. Such difficulties have motivated the development and usage of function approximations from the beginning, ranging from simple linear approximations to the state-of-the-art neural networks of today's deep learning. The critical works introducing the standard methods for function approximated RL — and especially *deep reinforcement learning* — will be cited, examined, and explored in the following work.

2.1.2 Markov decision processes

Markov decision processes (MDP) serve as the basic, idealized mathematical framework within which reinforcement learning problems can be formulated. Formally, an MDP is a stochastic process that satisfies the Markov property — i.e. that the conditional probability distribution of future states depends only on the current state (otherwise known as "memory-less"). This framework describes the trajectory of an agent as it interacts with an environment. There are a number of notational conventions in common use surrounding the study of MDPs. It is thus of note to the reader that the conventions used in this work will primarily follow those of Sutton (1998) [38].

Specifically, an MDP is composed of a *state space* denoted $\mathcal{S}$, an *action space* denoted $\mathcal{A}$, and a *transition probability distribution* denoted $p(s', r \mid s, a)$. This distribution describes the probability of the agent transitioning to state s' and receiving reward r, given it takes action a in state s. The Markov property ensures that the

dynamics of the environment are completely determined by $p(s', r \mid s, a)$. Since MDPs are inherently a stochastic process, it is crucial to recognize the trajectory of an agent through the environment as a sequence of random variables $S_t \in \mathcal{S}$, $A_t \in \mathcal{A}$, and by convention, $R_{t+1} \in \mathcal{R}$ where $\mathcal{R}$ is the set of possible rewards. Furthermore, the action taken in a particular state is given by the *policy* denoted $\pi(a \mid s)$ which, by representation as a conditional probability distribution, allows for the consideration of both stochastic and deterministic policies.

With this notation, the transition probability distribution is denoted as

$$p(s', r \mid s, a) = P\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\} \tag{2.1}$$

and the trajectory of an agent through an environment represented as

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3 \ldots \tag{2.2}$$

where for an *episodic* MDP, there is some finite $t = T$ such that the trajectory terminates at R_T. For a *continuing* MDP, the case of $T = \infty$ is considered [38].
In a similar fashion, the policy of an agent is formally defined as

$$\pi(a \mid s) = P\{A_t = a \mid S_t = s\} \tag{2.3}$$

as a distribution over the actions $a \in \mathcal{A}(s)$ available to the agent in state $s \in \mathcal{S}$.
In many cases, the *initial state distribution*, denoted $\mu(s) = P\{S_0 = s\}$, is of particular importance, especially regarding the "exploration vs. exploitation" considerations often made in RL.

MDPs are further categorized into *discounted* and *undiscounted* varieties, which refer to the relative weight given to long-term rewards. A *discount rate* parameter γ where $0 \leq \gamma \leq 1$ is introduced to attenuate the preference for immediate vs. future

expected reward. With this parameter in mind, the total discounted future reward at time t is denoted G_t where

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

The objective of the agent will thus be to select a policy $\pi(a \mid s)$ such that the chosen A_t maximizes the expected G_t.

Towards this objective, it is crucial to determine the value of each state in terms of the total reward that can be expected when following a particular policy. The contributions of Bellman [7] towards the utility of value functions in MDPs may be recalled from the previous section. Given a particular policy, two interrelated value functions can be constructed: the *state-value* function and the *action-value* function (alternatively called the *state-action value* function).

The state-value function $v_\pi(s)$ under a policy π is defined as

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \text{ , for all } s \in \mathcal{S} \tag{2.5}$$

i.e. the total expected reward for an agent starting in state s and following policy π.

The action-value function $q_\pi(s, a)$ under a policy π is defined as:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \text{ , for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}(s) \tag{2.6}$$

i.e. the total expected reward for an agent starting in state s, then taking action a, and thereafter following policy π.

With a characterization of state-values and action-values defined, the objective of the agent can be described as to find an *optimal policy* denoted π_* — i.e. the policy in which v_{π_*} and q_{π_*} ascribe the maximum value to each state (or state-action pair).

Formally, value functions under the optimal policy π_* are given by

$$v_{\pi_*} = \max_{\pi} v_\pi(s) \;\text{ and }\; q_{\pi_*} = \max_{\pi} q_\pi(s,a) \tag{2.7}$$

The major contribution of Bellman [7] to the study of MDPs comes from the aptly named *Bellman optimality equations* which, for finite MDPs, provide a recursive definition for each value function as

$$\begin{aligned} v_{\pi_*}(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi_*}(S_{t+1}) \mid S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\pi_*}(s')] \end{aligned} \tag{2.8}$$

with an equivalent form for $q_{\pi_*}(s,a)$ as

$$\begin{aligned} q_{\pi_*}(s,a) &= \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_{\pi_*}(S_{t+1}, A_{t+1}) \mid S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_{\pi_*}(s', a')] \end{aligned} \tag{2.9}$$

where s' and a' denote the next state and action respectively. Intuitively, these equations compute the maximum value of a state/state-action as the reward of the best possible action plus the maximum value of the resulting state/state-action, allowing for a recursive construction that relates each state/state-action to those surrounding.

The Bellman optimality equations for finite MDPs serve their most crucial role in the method of dynamic programming and indeed allow for a direct — albeit often computationally impractical — way to solve for a unique solution for finite MDPs where the environment dynamics are totally known. While the content of this work does not directly concern dynamic programming, finite MDPs, or problems where the environment dynamics are known, the framework defined for MDPs allows for a

more precise mathematical characterization of RL problems and thus describes the objective of all RL methods: to converge to an optimal policy. As will be described in the following sections, the methods used in RL often reach this objective by iteratively refining their approximation of the value functions and converging to a deterministic policy that is greedy with respect to them. Crucially, the use of function approximation allows for a process that does not require computation over the entire state-action space which, in nearly all real-world problems, is infeasible.

2.2 Methods and algorithms

This section introduces a subset of methods and algorithms used in reinforcement learning with emphasis on those relevant for this work, as well as place them within the conventional classifications used in the field. In particular, the objective of this section is to introduce the usage of function approximation as a powerful and necessary tool for the real-world application of RL.

2.2.1 Classifications

Reinforcement learning algorithms can broadly be divided into two categories: *model-based* and *model-free*. The use of the term "model" in these terms is often a source of confusion. In this case, model-based refers to an RL algorithm that relies on knowing — or otherwise building an estimate for — the environment dynamics. Dynamic programming, for example, requires total knowledge of the transition probability distribution and rewards as is necessary for solving the Bellman equations. In a general sense, a model-based algorithm relies less on sampling and more on constructing accurate predictions for the outcome of each action, and are thus typically considered "sample-efficient" [19]. Comparatively, model-free methods are not concerned with simulating environment dynamics and instead learn value-function estimates or policies

based solely on experience accrued by interacting with the environment. Such methods thus enjoy the advantage of greater generalizability at the cost of slower, more intensive learning. Model-based methods are encountered much less frequently and indeed the majority of available RL methods, including those used in this work, are classified as model-free.

Within the category of model-free algorithms, a further distinction is made between *on-policy* vs. *off-policy* methods. This distinction is defined by the difference in the *behavior policy* — the policy that determines an agent's trajectory — and the *update policy* — that which guides the update rule for the optimal policy estimate. As the name suggests, the behavior and update policies are one and the same for on-policy methods i.e. the agent improves its own policy through experience. On the other hand, an off-policy method maintains a distinction between the behavior and update policies such that the agent's own behavior may not be related to the optimal policy estimate it is building. Since off-policy methods can improve the estimate while still exploring heavily, it may appear that they would be preferred to on-policy methods. Unsurprisingly, the reality is more nuanced. Given enough exploration, off-policy methods produce an unbiased estimate of the optimal policy, as well as being considered sample-efficient. However, on-policy methods are often easier to implement, less computationally intensive, and may produce "safer" policies since the agent is following its own estimate (thus discouraging a risky policy) [13]. Ultimately, the user must choose a paradigm that suits the specific use-case.

2.2.2 Dynamic programming

Although dynamic programming as a method is not directly relevant to this work, a cursory overview of the framework is useful as an introduction to the interplay between *policy evaluation* and *policy improvement* where, as mentioned previously, most reinforcement learning methods can framed in the context of *general policy*

iteration.

Dynamic programming, in its most common form, is applicable for solving control problems modeled as discrete finite Markov decision processes. Note that the method developed by Bellman [7] is also applicable for continuous control problems either in the usual way via discretization, or by solving the Hamilton-Jacobi-Bellman partial differential equation through numerical methods. In the usual case, "discrete" refers to discrete-time (as MDPs are required to be) and "finite" refers to a finite state and action space. The following exposition summarizes that provided by Sutton (2018) [38], with reference to Howard (1960) [14] when applicable.

Recall the Bellman optimality equations (2.8) and (2.9) respectively:

$$v_{\pi_*}(s) = \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_{\pi_*}(s')]$$

$$q_{\pi_*}(s, a) = \sum_{s',r} p(s', r \mid s, a)[r + \gamma \max_{a'} q_{\pi_*}(s', a')]$$

The main idea of *policy iteration* is begin with some non-optimal policy π_0 and alternate steps of *policy evaluation* and *policy improvement* until the the Bellman equations are satisfied — at which point the unique optimal policy has been reached. The general scheme of this process is as follows:

$$\pi_0 \xrightarrow{\text{Ev}} v_{\pi_0} \xrightarrow{\text{Im}} \pi_1 \xrightarrow{\text{Ev}} v_{\pi_1} \xrightarrow{\text{Im}} \pi_2 \xrightarrow{\text{Ev}} v_{\pi_2} \xrightarrow{\text{Im}} \ldots\ldots \xrightarrow{\text{Im}} \pi_* \xrightarrow{\text{Ev}} v_{\pi_*}$$

Policy evaluation itself is often an iterative process via

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \tag{2.10}$$

which can be shown to converge to v_π in general [14].

Policy improvement is more straightforward. Consider some policy π for which v_π has been evaluated. Denote the *greedy* policy π' where

$$\begin{aligned}\pi'(a \mid s) &= \arg\max_a q_\pi(s,a) \\ &= \arg\max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v_\pi(s')]\end{aligned} \tag{2.11}$$

where the selection of the maximizing action ensures that $v_{\pi'}(s) \geq v_\pi(s)$ for all s. If $v_{\pi'}(s) = v_\pi(s)$, then

$$v_{\pi'}(s) = \max_a \sum_{s',r} p(s',r \mid s,a)[r + \gamma v_{\pi'}(s')] \tag{2.12}$$

which satisfies Bellman optimality.

The method of dynamic programming via policy iteration for finite MDPs consists of the alternating application of evaluation and improvement until optimality is reached. Since both steps require knowledge of the environment dynamics i.e. the transition probability $p(s',r \mid s,a)$, the method is classified as model-based.

2.2.3 Temporal-difference methods

Temporal-difference (TD) learning is centered around the idea of *bootstrapping*, that is, constantly updating prior predictions to match new experience. The "temporal-difference" as named refers to the difference between a previous and newer estimate constructed as the agent gains experience in the environment. Sutton (1988) [37] provides a succinct demonstration

> Suppose you wish to predict the weather for Saturday, and you have some model that predicts Saturday's weather, given the weather of each day in the week. In the standard case, you would wait until Saturday and then adjust all your models. However, when it is, for example, Friday, you should have a pretty good idea of what the weather would be on Saturday – and thus be able to change, say, Saturday's model before Saturday arrives.

TD methods are classified as model-free, but include both on-policy and off-policy variants. In their simplest construction, TD methods solve the control problem by the evaluation of q_π along a certain type of policy. This evaluation is done by minimization of the *TD error* which compares a prior evaluation to a newer unbiased estimate of the true value — called the *TD target*. If with sufficient exploration the policy being evaluated converges to a greedy policy, then $q_\pi \to q_{\pi_*}$ is expected since the optimal policy is always greedy with respect to action-values. A common choice is an ε-greedy policy which, for small $\varepsilon > 0$, selects a random action with probability ε and the greedy action otherwise. The construction of the TD target is the main difference between the various TD algorithms. In general, the update rule for the iteration in the tabular case is

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[U_t - q(S_t, A_t)] \tag{2.13}$$

where α is the step-size (alternatively "learning rate") and U_t is the chosen TD target.

SARSA ("state-action-reward-state-action") is the ubiquitous on-policy TD method in which the TD target is set as the n-step discounted reward experienced by the agent. Specifically, the target is set as

$$U_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{n-1} R_{t+n} + \gamma^n q_{t+n-1}(S_{t+n}, A_{t+n}) \tag{2.14}$$

The simplest case is referred to as SARSA(0) i.e. one-step SARSA, where the update rule is

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) - q(S_t, A_t)] \tag{2.15}$$

Note that the next action A_{t+1} is sampled from the policy — typically ε-greedy — and thus the update depends only on state-action pairs visited by the agent's trajectory. For this reason, n-step SARSA is considered an on-policy algorithm. Additionally,

convergence to the optimal policy for SARSA(0) is guaranteed assuming infinite visits to every state-action pair and convergence to a greedy policy Singh (2000) [35]. These criteria motivate the usual choice for ε-greedy policies.

However, the TD target can be modified to incorporate off-policy learning. *Q-learning* is the standard off-policy TD method. It allows the agent to pursue an exploratory trajectory while simultaneously updating the action-value estimate as if a greedy policy were being followed, recalling the discussion of behavior and update policies from Section 2.2.1. The Q-learning update for the tabular case is set as

$$q(S_t, A_t) \leftarrow q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a'} q(S_{t+1}, a') - q(S_t, A_t)] \tag{2.16}$$

There are a few differences between Q-learning and SARSA that have been shown heuristically [40]. Q-learning generally converges faster than SARSA, but SARSA will often prefer a "safer" policy. Such a difference may be attributed to SARSA's on-policy nature where a ε-greedy policy may discourage risky trajectories that skirt on the brink of failure.

2.2.4 Function approximation

The discussion of reinforcement learning via Markov decision processes has thus far proceeded with the assumption that the state-action space is small enough to be stored and enumerated in its entirety. Methods that utilize a total representation of the environment are referred to as *tabular methods* in reference to their typical storage as an array. However, the majority of real-life applications have state-action spaces much too vast to be reasonably stored or computed on in their entirety, leading to the "curse of dimensionality" for tabular methods, as coined by Bellman [7]. *Approximate-solution methods* therefore provide an useful toolkit for learning optimal policies on otherwise intractable environments.

Approximate-solution methods center around the construction of a simpler model for the state (or state-action) space such that the target — typically a value or advantage function — may be usefully approximated. Most often, the goal is to construct either $\hat{v}_\pi : \mathcal{S} \to \mathbb{R}$ or $\hat{q}_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ as approximations to the true value functions $v_\pi(s)$ or $q_\pi(s, a)$ respectively — i.e. evaluate policy π. There are a wide variety of methods to construct such approximations. Most relevant to this work are *parametric methods* in which the approximations are defined with respect to some vector w of parameters.

The simplest category of parametric methods are *linear parametric methods* in which the state (or state-action) space is represented by d features. Let $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ be a feature map for the state-action space with d features. For $(s, a) \in \mathcal{S} \times \mathcal{A}$, denote the linear function approximation of $q_\pi(s, a)$ as

$$\hat{q}_w(s, a) = w^\top \phi(s, a) \tag{2.17}$$

where $w \in \mathbb{R}^d$ is the vector of parameters such that

$$w \in \arg\min_{z \in \mathbb{R}^d} \mathbb{E}_{s \sim \mu, a \sim \pi}[q_\pi(s, a) - \hat{q}_z(s, a)]^2 \tag{2.18}$$

i.e. the mean squared error is minimized. Such a policy evaluation for π is often done iteratively via gradient methods. If parameter w is obtained such that $\hat{q}_w \approx q_{\pi^*}$, then the control problem is solved simply by selecting a policy π that is greedy with respect to $\hat{q}_w$. The minimization problem as stated can be reframed as a desired to achive Bellman optimality, where the objective in this case would be to obtain the parameter w such that the Bellman error

$$\delta_w(s, a) = \mathbb{E}[R_t + \gamma \max_{a' \in \mathcal{A}(s)} \hat{q}_w(S_{t+1}, a') - \hat{q}_w(s, a) \mid S_t = s, A_t = a] \tag{2.19}$$

for each state-action pair is minimized in squared expectation, i.e.

$$w \in \arg\min_{z \in \mathbb{R}^d} \mathbb{E}_{s\sim\mu, a\sim\pi_w} \delta_z(s,a)^2 \tag{2.20}$$

This objective applies to parametric methods generally and not only to the linear case. In fact, artificial neural networks (ANNs) are encompassed by the category of *non-linear parametric methods* where — amongst other complexities — the same construction and minimization problem apply.

Deep reinforcement learning is the application of ANNs as function approximators in the same way as described thus far. The major benefit that ANNs hold over linear methods is that the set of features does not require deliberate construction. Of course, the structure of the network as designed has a bearing on the performance of the approximator, but the network's hidden layers learn the important features of the state-action space themselves — albeit in an opaque manner. As an example to demonstrate a neural network-approximated value function, consider the two-layer case in which the approximation of q is constructed as

$$\hat{q}_\theta(x) = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} a_j \sigma(W_j^\top x) \tag{2.21}$$

where the state-action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ is represented by a vector $x \in \mathcal{X} \subseteq \mathbb{R}^d$ where $d > 2$, $\|x\|_2 = 1$, and the set of rewards bounded above for any $x \in \mathcal{X}$. The parameters of this network are $\theta = (a_1, \ldots, a_m, W_1, \ldots, W_m)$ and $\sigma(y)$ is the chosen activation function. The training procedure thus proceeds on θ until a θ^* is obtained such that $\hat{q}_{\theta^*}$ satisfies the minimization in (2.20).

2.2.5 Policy-gradient methods

The objective of reinforcement learning is to converge to an optimal policy. In the previous section, this objective was reached via the convergence of a function approximation to the optimal value function and subsequently choosing the greedy policy. Instead, consider converging upon the policy directly. This idea is a generalization of the method of *policy iteration* described in Section 2.2.2 in which the policy is improved with respect to some evaluated performance measure. Using the tools of function approximation developed thus far, the policy may be parametrized with respect to a set of features. Ensuring differentiability of this construction across the parameter space allows for standard methods of stochastic gradient ascent/descent to be applied.

Formally, consider a smooth parametrized policy π_θ where θ is the parameter. Note that a *stochastic policy* is desired i.e. the policy defines a probability distribution for the actions conditional on the current state and is thus denoted $\pi_\theta(a \mid s)$. A common choice for such a policy is a *softmax* selection (i.e. the *normalized exponential function*) which ensures a differentiable policy that satisfies the requirements for a probability distribution.

A softmax policy is defined as

$$\pi_\theta(a \mid s) = \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}} \tag{2.22}$$

where $h : \mathcal{S} \times \mathcal{A} \times \Theta \to \mathbb{R}$ ascribes a numerical preference for each action. Notably, the function $h(s, a, \theta)$ can make use of a featurizer $\phi(s, a)$ in the same manner as the value function approximations discussed previously. The construction of $h(s, a, \theta)$ may simply be linear in the features i.e.

$$h(s, a, \theta) = \theta^\top \phi(s, a) \tag{2.23}$$

or more complex — like an artificial neural network as described in Section 2.2.3. Of course, softmax selection is only a single example of a possible parameterization.

The performance measure of interest is often the total expected reward of a trajectory. Consider a trajectory $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, \ldots r_T)$ and denote the total reward for the trajectory as $R(\tau)$. The aim is to maximize the performance measure

$$J(\theta) = \mathbb{E}_\tau[R(\tau)] = \int p(\tau|\theta)R(\tau)d\tau \tag{2.24}$$

i.e. the expected total reward over possible trajectories under π_θ.
Gradient ascent with respect to θ requires evaluation of $\nabla_\theta J(\theta)$ as

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \int p(\tau|\theta)R(\tau)d\tau \\ &= \int p(\tau|\theta)\nabla_\theta \log p(\tau|\theta)R(\tau)d\tau \\ &= \mathbb{E}_\tau[\nabla_\theta \log p(\tau|\theta)R(\tau)] \end{aligned} \tag{2.25}$$

where $p(\tau \mid \theta)$ is the probability of a particular trajectory under π_θ. Unrolling this into an expression in terms of the initial state distribution, policy, and transition probability results

$$\begin{aligned} p(\tau|\theta) = \mu(s_0) \cdot \pi_\theta(a_0|s_0) \cdot p(s_1, r_1|s_0, a_0) \cdot \pi_\theta(a_1|s_1) \cdot p(s_2, r_2|s_1, a_1) \\ \ldots \pi_\theta(a_{T-1}|s_{T-1}) \cdot p(s_T, r_T|s_{T-1}, a_{T-1}) \end{aligned} \tag{2.26}$$

which can be substituted into the previous expression to yield

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau) \right] \tag{2.27}$$

Remarkably, the policy gradient expression allows for an unbiased estimate of the performance gradient to be computed without knowledge of the environment dynamics. However, this expression is known to have high variance for longer time horizons [24].

In general, performance gradient estimators for the infinite time horizon have the form

$$g = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{2.28}$$

where $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is called the *score function* [30].
Schulman (2018) [33] provides a list of common choices for Ψ_t

- $\sum_{t=0}^{\infty} r_t$: total trajectory reward
- $\sum_{t=t'}^{\infty} r_{t'}$: reward following a_t
- $\sum_{t'=t}^{\infty} r_{t'} - b(s_{t'})$: baselined version of above
- $q_{\pi_\theta}(s_t, a_t)$: state-action value function
- $A_{\pi_\theta}(s_t, a_t)$: advantage function
- $r_t + v_{\pi_\theta}(s_{t+1}) - v_{\pi_\theta}(s_t)$: one-step TD residual

Advantage functions are of particular interest as a choice to reduce the estimator variance. Broadly, advantage functions approximate the quantity $q_{\pi_\theta}(s_t, a_t) - v_{\pi_\theta}(s_t)$ i.e. the excess expected reward gained by taking action a_t.

Practical applications of policy gradient methods often make the tradeoff between variance and bias. Schulman (2018) [33] provides the *generalized advantage estimate*, the exposition of which will be provided here due to its ubiquitous usage. In particular, Schulman considers the γ-discounted value functions which — although introducing bias into the estimate — greatly reduce the variance by attenuating the effects of rewards distal to (s_t, a_t)

$$\begin{aligned} v^\gamma_{\pi_\theta}(s_t) &= \mathbb{E}_\tau \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \right] \\ q^\gamma_{\pi_\theta}(s_t, a_t) &= \mathbb{E}_\tau \left[\tilde{r}_t + \sum_{k=1}^{\infty} \gamma^k r_{t+k} \right] \\ A^\gamma_{\pi_\theta}(s_t, a_t) &= q^\gamma_{\pi_\theta}(s_t, a_t) - v^\gamma_{\pi_\theta}(s_t) \end{aligned} \tag{2.29}$$

where $\tilde{r}_t$ is the off-trajectory reward from taking action a_t.

With this formulation, the γ-discounted policy gradient is defined as

$$g^\gamma = \mathbb{E}_\tau \left[\sum_{t=0}^{\infty} A^\gamma_{\pi_\theta} \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{2.30}$$

Under GAE, the advantage function $A^\gamma_{\pi_\theta}$ is estimated by the k-step discounted approximation $\hat{A}^{(k)}_{\pi_\theta}$ constructed by first considering the discounted one-step TD residual

$$\delta^\gamma_t = r_t + \gamma v^\gamma_{\pi_\theta}(s_{t+1}) - v^\gamma_{\pi_\theta}(s_t) \tag{2.31}$$

which can be verified to be an unbiased estimator for $A^\gamma_{\pi_\theta}$ as

$$\begin{aligned} \mathbb{E}[\delta^\gamma_t] &= \mathbb{E}[r_t + \gamma v^\gamma_{\pi_\theta}(s_{t+1}) - v^\gamma_{\pi_\theta}(s_t)] \\ &= \mathbb{E}[q^\gamma_{\pi_\theta}(s_t, a_t) - v^\gamma_{\pi_\theta}(s_t)] \\ &= A^\gamma_{\pi_\theta}(s_t, a_t) \end{aligned} \tag{2.32}$$

The k-step discounted approximation is thus defined as the telescoping sum

$$\begin{aligned} \hat{A}^{(1)}_{\pi_\theta} &:= \delta^\gamma_t && = r_t + \gamma v^\gamma_{\pi_\theta}(s_{t+1}) - v^\gamma_{\pi_\theta}(s_t) \\ \hat{A}^{(2)}_{\pi_\theta} &:= \delta^\gamma_t + \gamma \delta^\gamma_{t+1} && = r_t + \gamma r_{t+1} + \gamma^2 v^\gamma_{\pi_\theta}(s_{t+2}) - v^\gamma_{\pi_\theta}(s_t) \\ &\vdots \\ \hat{A}^{(k)}_{\pi_\theta} &:= \sum_{l=0}^{k-1} \gamma^l \delta^\gamma_{t+1} \end{aligned} \tag{2.33}$$

The generalized advantage estimator GAE(γ, λ) is finally given by the exponentially-weighted average of the k-step approximations. The specific details of the derivation

may be found in Schulman (2018) [33].

$$\begin{aligned} \mathrm{GAE}(\gamma,\lambda) &= (1-\lambda)(\hat{A}^{(1)}_{\pi_\theta} + \lambda\hat{A}^{(2)}_{\pi_\theta} + \lambda^2\hat{A}^{(3)}_{\pi_\theta} + \ldots) \\ &\vdots \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta^{\gamma}_{t+l} \end{aligned} \tag{2.34}$$

The policy gradient estimate with generalized advantage estimation is therefore computed for N sampled trajectories as

$$\hat{g}^{\gamma} = \frac{1}{N}\sum_{n=1}^{N}\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a^n_t|s^n_t)\sum_{l=0}^{\infty}(\gamma\lambda)^l\delta^{\gamma}_{t+l} \tag{2.35}$$

Chapter 3

Manifold-optimized RL

This chapter motivates and presents the application of information geometry towards the improvement of the standard reinforcement learning methods presented in the last chapter. This exploration is done under the hypothesis that there exists a latent manifold structure to the various objects studied in RL which may be exploited to improve the training trajectory. This idea is approached from two directions: imposing an explicit manifold restriction on value-function parameters, and introducing second-order policy information as a regularizer.

3.1 Motivation and previous work

The *manifold hypothesis* is a pervasive idea in machine learning which contends that high-dimensional data exists in the vicinity of — or is perhaps sampled from — embedded low-dimensional manifolds. This hypothesis has been used to motivate the development of *manifold learning* techniques which leverage this assumption in the spirit of dimensionality reduction and feature selection. Specifically, as Cayton (2005) [8] describes, such techniques aim to duplicate the behavior of principal component analysis (PCA) except on manifolds instead of on linear subspaces. The literature in this regard has expanded in recent years with attempts to either test this hypothesis,

such as the algorithm proposed in Fefferman (2013) [10], or leverage it through manifold regularization, such as in Saxe (2014) [29], Lee (2015) [23], and Jin (2020) [20].

These efforts build upon a more general theory of statistical manifolds — known as *information geometry* — of which the prolific work of Amari forms the foundational basis. Amari (1998a) [5] introduces the *natural gradient* as an improved descent direction for cost-function defined on a manifold, extending these ideas in Amari (2000) [6] for manifold learning within the parameter space of neural networks. Building directly off of Amari's work, Kakade (2001) [21] introduces the *natural policy gradient* as an improvement over the standard policy gradient method by including information from the manifold of policies. In particular, the work of Amari and Kakade revolve around the use of the Fisher information matrix as a suitable metric on a statistical manifold. Schulman's popular Trust-Region Policy Optimization (TRPO) [31] and Proximal-Policy Optimization [32] methods that are ubiquitous in RL today draw directly from Kakade's natural policy gradient, allowing all three methods to be viewed through an information theoretic lens. Most recently, the work of Chen (2023) [9] has introduced a method for learning a suitable metric tensor which reflects the differential structure of the policy network and loss function, rather than the standard choice of the Fisher information metric.

In general, the motivation behind all of these methods — and indeed this work itself — is to exploit the geometric structure inherent in the objects of interest in reinforcement learning. The potential of manifold learning has already been demonstrated by the precedent, but the application of such techniques to reinforcement learning is a relatively recent development. In this sense, the purpose of this work is to explore the application of geometric techniques to reinforcement learning and to examine the performance of the resulting modified algorithms.

3.2 Manifold-restricted methods

A straightforward approach for examining the effect of an assumed geometric structure is to impose a manifold restriction. By forcing an algorithm's training orbit to lie upon some manifold within the broader parameter space, a simpler case study is constructed allowing for a more interpretable examination. Specifically, the objective of this section is to use such a restriction to develop an understanding of the tangent space of a parameter manifold and develop a method for curvilinear descent upon that manifold. The purpose of this section is demonstrative rather than practical since information geometric considerations are not taken into account i.e. both of the methods presented ignore stochastic effects.

In this section, the manifold restriction chosen is an orthogonality restriction which has precedent for deep neural networks in Huang (2017) [16]. For an application to reinforcement learning, an orthogonality restriction may be a useful proxy for the orthogonal structure of the action space of some environments. Intuitively, some environments may have discrete actions sets — such as $\{\text{up}, \text{down}, \text{left}, \text{right}\}$ or $\{\text{forward}, \text{backward}\}$ or $\{\text{active}, \text{inactive}\}$ — in which it may be desirable for the the action-value function approximation $q(s, a)$ to reflect their inherent orthogonality. The setup for this section is as follows:

Consider a linear approximation to the state-action value function of an RL problem with a discrete, finite action space $\mathcal{A}$ as

$$q_W(s, a) = (We(a))^\top \phi(s) \tag{3.1}$$

where function $e : \mathcal{A} \to \mathbb{R}^{|\mathcal{A}|}$ maps action a to a standard basis vector of $\mathbb{R}^{|\mathcal{A}|}$, featurizer $\phi : \mathcal{S} \to \mathbb{R}^d$ maps a state to a vector of d features, and $W \in \mathbb{R}^{d \times |\mathcal{A}|}$ is the matrix of parameters. As discussed in Section 2.2.3, the objective is to obtain parameter matrix

W such that the Bellman error

$$\delta_W(s,a) = \mathbb{E}[R_t + \gamma \max_{a' \in \mathcal{A}(s)} q_W(S_{t+1}, a') - q_W(s,a) \mid S_t = s, A_t = a] \tag{3.2}$$

for each state-action pair is minimized in squared expectation, i.e.

$$W^* = \underset{W \in \mathbb{R}^{d \times |\mathcal{A}|}}{\arg\min} \ \mathbb{E}_{s \sim \mu, a \sim \pi_W} \left[\frac{1}{2} \delta_W(s,a)^2 \right] \tag{3.3}$$

For the entirety of this section and the subsequent experiments, the algorithm used for solving this minimization is a one-step semi-gradient SARSA (see Section 2.2.4) in which the standard update rule is given by

$$W_{t+1} = W_t + \alpha[r_t + \gamma q_{W_t}(s_{t+1}, a_{t+1}) - q_{W_t}(s_t, a_t)] \nabla q_{W_t}(s_t, a_t) \tag{3.4}$$

where the step-size is denoted α for convenience, but more accurately denoted as α_t which decreases in iteration to satisfy the convergence conditions for stochastic gradient descent (see Section 2.2.4).

The orthogonality condition to be imposed on this algorithm requires that $W^\top W = I$ at all iterations. In particular, the set $\{X \in \mathbb{R}^{n \times p} \mid X^\top X = I_p\}$ is a smooth manifold embedded in $\mathbb{R}^{n \times p}$ referred to as the *Stiefel Manifold*, and is normally denoted as $\mathcal{V}_p(\mathbb{R}^n)$ to describe a p-tuple of orthonormal vectors (a p-frame) in $\mathbb{R}^n$. Verification that $\mathcal{V}_p(\mathbb{R}^n)$ is a smooth manifold follows from the Preimage Theorem [28].

Lemma 3.2.1. *The space $\mathcal{V}_p(\mathbb{R}^n)$ of p-frames in $\mathbb{R}^n$ is a compact smooth manifold of dimension $nk + \frac{k(k-1)}{2}$.*

Proof. (Compactness) Consider $\mathcal{V}_p(\mathbb{R}^n)$ as a subspace of all orthonormal p-tuples of

$$(S^{n-1})^p = S^{n-1} \times S^{n-1} \times \underset{p \text{ times}}{\cdots} \times S^{n-1} \times S^{n-1} \tag{3.5}$$

where S^n is the n-sphere. Since n-spheres are compact then their product is compact. $\mathcal{V}_p(\mathbb{R}^n)$ takes the subspace topology of this product and since

$$\mathcal{V}_p(\mathbb{R}^n) = \{(v_1, \ldots, v_p) \in (S^{n-1})^p \mid \langle v_i, v_j \rangle = 0 \text{ for } i \neq j \text{ and } \langle v_i, v_j \rangle = 1 \text{ for } i = j\} \tag{3.6}$$

every convergent sequence $V_k \in \mathcal{V}_p(\mathbb{R}^n)$ converges to an element of $\mathcal{V}_p(\mathbb{R}^n)$, thus it is a closed subspace. Since $\mathcal{V}_p(\mathbb{R}^n)$ is a closed subspace of compact space, it is compact. (Manifold) Define the map

$$f : \mathbb{R}^{n\times p} \to \mathbb{R}^{p\times p} \tag{3.7}$$

where $f(X) = X^\top Y$ and observe that $\mathcal{V}_p(\mathbb{R}^n) = f^{-1}(I_p)$. It suffices to show that I_p is a regular value of f i.e. that the map between tangent spaces

$$df_X : \mathcal{T}_X(\mathbb{R}^{n\times p}) \to \mathcal{T}_{f(X)}(\mathbb{R}^{p\times p}) \tag{3.8}$$

is surjective for all $X \in \mathcal{V}_p(\mathbb{R}^n)$. Calculate $df_X(Y)$ as

$$\begin{aligned} df_X(Y) &= \lim_{h\to 0} \frac{f(X+hY) - f(X)}{h} \\ &= \lim_{h\to 0} \frac{(X+hY)^\top (X+hY) - X^\top X}{h} \\ &= \lim_{h\to 0} X^\top Y + Y^\top X + hY^\top Y \\ &= X^\top Y + Y^\top X \end{aligned} \tag{3.9}$$

where for any $Z \in \mathbb{R}^{p\times p}$, set $Y = \frac{1}{2}XZ$ and obtain

$$df_X(Y) = X^\top(\frac{1}{2}XZ) + (\frac{1}{2}XZ)^\top X = Z \tag{3.10}$$

thus the map between tangent spaces is surjective and I_p is a regular value of f, therefore $\mathcal{V}_p(\mathbb{R}^n)$ is a smooth manifold by the Preimage Theorem. Specifically, $\mathcal{V}_p(\mathbb{R}^n)$

is of dimension

$$\dim \mathcal{V}_p(\mathbb{R}^n) = \dim \mathbb{R}^{n\times p} - \dim \mathbb{R}^{p\times p} = np - \frac{p(p+1)}{2} \tag{3.11}$$

□

The geometric structure of the Stiefel manifold will be explored further in Section 3.2.2 with curvilinear descent. In order to examine the effects of an orthogonal restriction on the action-value function approximation parameters, the most direct method is to simply re-orthogonalize the parameter matrix at each step. Such a procedure is equivalent to projecting the next iterate onto the Stiefel manifold at the closest point with respect to some metric, as will be shown in the next section.

3.2.1 Orthogonal Procrustes

The simplest procedure for implementing an orthogonal restriction is to produce the matrix W via the standard update and then compute the matrix R with orthogonal columns (i.e. where $R^\top R = I$) which is closest to W with respect to the Frobenius inner product $\langle A, B\rangle = \mathrm{Tr}(A^\top B)$ and its induced norm. This problem is equivalent to the *Orthogonal Procrustes Problem* well-known in the literature whose general solution is given by Schönemann (1966) [34]. An additional solution for reduced-rank matrices is given by Zou (2006) [17].

Theorem 3.2.2. *Let $R, W \in \mathbb{R}^{n\times p}$ be two matrices and consider the minimization*

$$\hat{R} = \arg\min_R \|R - W\|_F^2 \quad s.t \quad R^\top R = I \tag{3.12}$$

Suppose the singular value decomposition $W = U\Sigma V^\top$, then $\hat{R} = UV^\top$

Proof. The proof follows from a straightforward substitution of the decomposition

$$\begin{aligned}
\hat{R} &= \arg\min_R \|R - W\|^2 \\
&= \arg\min_R \langle R - W, R - W \rangle \\
&= \arg\min_R \|R\|^2 + \|W\|^2 - 2\langle R, W \rangle \\
&= \arg\max_R \langle R, W \rangle \\
&= \arg\max_R \langle R, U\Sigma V^\top \rangle \\
&= \arg\max_R \langle U^\top R V, \Sigma \rangle \\
&= UV^\top \qquad\qquad \text{since } \Sigma \text{ non-negative}
\end{aligned} \tag{3.13}$$

Specifically, the last statement results since $U^\top \hat{R} V = I$ is required for the maximization.

□

A simple procedure to project W onto the Stiefel manifold at the point closest with respect to the Frobenius norm can now be constructed by first producing W_{t+1} via the standard one-step semi-gradient SARSA update, then computing the singular value decomposition $W_{t+1} = U_{t+1}\Sigma_{t+1}V_{t+1}^\top$, and finally producing

$$\tilde{W}_{t+1} = U_{t+1}V_{t+1}^\top \tag{3.14}$$

3.2.2 Curvilinear descent on Stiefel manifold

Imposing an orthogonality restriction on the parameter matrix can also be enforced by replacing the standard stochastic gradient descent in the parameter space with a curvilinear descent along the embedded Stiefel manifold. In particular, such a method requires a characterization of the tangent space of Stiefel manifolds in order to construct a parameterized curve along which to compute the update. The idea in

this section is thus to compute the standard update direction, project this direction onto the tangent plane of the Stiefel manifold, and then use a parameterization of a curve along the manifold to produce the next iterate.

Recall the Stiefel manifold as

$$\mathcal{V}_p(\mathbb{R}^n) = \{X \in \mathbb{R}^{n\times p} \mid X^\top X = I_p\} \tag{3.15}$$

Consider a particular $X \in \mathcal{V}_p(\mathbb{R}^n)$ and extend X to an orthonormal basis of $\mathbb{R}^n$, letting $X_\perp$ be the $n \times (n-p)$ matrix composed of the additional vectors such that the concatenation $[XX_\perp]$ is an $n \times n$ orthonormal matrix. Since it is orthonormal, it is an invertible linear operator and thus an automorphism on $\mathbb{R}^{n\times p}$. Therefore, any element $U \in \mathbb{R}^{n\times p}$ can be written as $U = [XX_\perp]C$ for some $C \in \mathbb{R}^{n\times p}$. Splitting $C = \begin{bmatrix} A \\ B \end{bmatrix}$ where A and B are $p \times p$ and $(n-p) \times p$ respectively allows for the representation

$$U = XA + X_\perp B \tag{3.16}$$

Denote the tangent space of the Stiefel manifold at X as $\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$. Tagare (2011) [41] provides the following characterization

Lemma 3.2.3. *For any $Z \in \mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$*

$$Z^\top X + X^\top Z = 0 \tag{3.17}$$

holds i.e. Z is a skew-symmetric $p \times p$ matrix

Proof. Let $Y(t)$ be a smooth curve on $\mathcal{V}_p(\mathbb{R}^n)$ where $Y(0) = X$ and $Y'(0) = Z \in \mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$. Since $Y(t)$ lies on the Stiefel manifold, $Y(t)^\top Y(t) = I_p$ holds and differen-

tiation with respect to t results

$$Y'(t)^\top Y(t) + Y(t)^\top Y'(t) = 0 \tag{3.18}$$

where evaluation at $t = 0$ gives the result. □

The tangent space can be further represented by in terms of the orthonormal basis $[XX_\perp]$ as follows

Lemma 3.2.4. *A matrix $Z = XA + X_\perp B$ is in the tangent space $\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$ if and only if A is skew-symmetric*

Proof. ($\Rightarrow$) Let $Z \in \mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$ represented as $Z = XA + X_\perp B$. The condition from Lemma 3.2.3 requires $Z^\top X + X^\top Z = 0$ which upon substitution of Z results $A^\top + A = 0$ i.e. A is skew-symmetric.
($\Leftarrow$) Consider the subspace S of $\mathbb{R}^{n\times p}$ of all matrices $Z = XA + X_\perp B$ with A skew-symmetric. From the forward direction, it has been shown that $\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$ is a subspace of S. Furthermore,

$$\dim(S) = np - \frac{p(p+1)}{2} = \dim(\mathcal{V}_p(\mathbb{R}^n)) = \dim(\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)) \tag{3.19}$$

thus $S = \mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$ □

The canonical inner product used on the tangent space of Stiefel manifolds used predominantly in the literature aims to weigh each coordinate A and B equally. It is given (without proof) as

$$\langle Z_1, Z_2\rangle = \mathrm{Tr}(Z_1^\top (I - \frac{1}{2}XX^\top)Z_2) \tag{3.20}$$

For some point $X \in \mathcal{V}_p(\mathbb{R}^n)$ and a function $F : \mathbb{R}^{n\times p} \to \mathbb{R}$, let $G = \left[\dfrac{\partial F}{\partial X_{i,j}}\right] \in \mathbb{R}^{n\times p}$

and define the directional derivative of F at X towards some Z as

$$DF_X(Z) = \mathrm{Tr}(G^\top Z) \tag{3.21}$$

The aim is to find a matrix form AX that represents the action of DF_X on the tangent plane with respect to the canonical inner product. Following Tagare (2011) [41]:

Lemma 3.2.5. *The vector AX with $A = (GX^\top - XG^\top)$ represents the action of DF_X on the tangent space $\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$*

Proof. Consider the differential $G = \left[\dfrac{\partial F}{\partial X_{i,j}}\right] \in \mathbb{R}^{n\times p}$ as well as some vector $Z \in \mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$. Each can be represented in the coordinate form at X as $G = XG_A + X_\perp G_B$ and $Z = XZ_A + Z_\perp Z_B$ with the restriction that Z_A is skew symmetric. The directional derivative $DF_X(Z)$ is thus given by the substitution as

$$\begin{aligned} DF_X(Z) &= \mathrm{Tr}(G^\top Z) \\ &= \mathrm{Tr}(G_A^\top Z_A) + \mathrm{Tr}(G_B^\top Z_B) \end{aligned} \tag{3.22}$$

Perform the decomposition $G_A = \mathrm{sym}(G_A) + \mathrm{skew}(G_A)$ where upon substitution,

$$DF_X(Z) = \mathrm{Tr}(\mathrm{skew}(G_A)^\top Z_A) + \mathrm{Tr}(G_B^\top Z_B) \tag{3.23}$$

since Z_A is skew-symmetric (and thus $\mathrm{Tr}(\mathrm{sym}(G_A)^\top Z_A) = 0$)

Consider some matrix $U = XA + X_\perp B$ that represents the action of DF_X on the tangent space $\mathcal{T}_X\mathcal{V}_p(\mathbb{R}^n)$ with respect to the canonical inner product. Specifically, $\langle U, Z\rangle = \mathrm{Tr}(\mathrm{skew}(G_A)^\top Z_A) + \mathrm{Tr}(G_B^\top Z_B)$ is required.

$$\begin{aligned} \langle U, Z\rangle &= \mathrm{Tr}(U^\top (I - \frac{1}{2}XX^\top)Z_2) \\ &= \frac{1}{2}\mathrm{Tr}(A^\top Z_A) + \mathrm{Tr}(B^\top Z_B) \end{aligned} \tag{3.24}$$

where the requirement results in $A = 2\mathrm{skew}(G_A)$ and $B = G_B$. To simplify this form

further, note that $\text{skew}(G_A) = \frac{1}{2}(G_A - G_A^\top) = \frac{1}{2}(X^\top G - G^\top X)$ allowing for

$$\begin{aligned} U &= 2\text{skew}(G_A) + X_\perp G_B \\ &= X(X^\top G - G^\top X) + X_\perp G_B \\ &= XG_A + X_\perp G_B - XG^\top X \\ &= G - XG^\top X \\ &= (GX^\top - XG^\top)X \end{aligned} \tag{3.25}$$

i.e. if $A = (GX^\top - XG^\top)$, then AX is the action of DF_X on the tangent plane. □

In the case of the semi-gradient SARSA algorithm, the function of interest is $q_W(s,a) = (We(a))^\top \phi(s)$ where at each time step, the gradient $\nabla q_W(s,a) = \left[\frac{\partial q_W}{\partial X_{i,j}}\right]$ is computed as per the standard update rule. Let $n = d = |\phi(\mathcal{S})|$ and $p = |\mathcal{A}|$. Assuming $W \in \mathbb{R}^{n\times p}$ enters an update with orthonormal columns, the action of Dq_W on the tangent space $\mathcal{T}_W\mathcal{V}_p(\mathbb{R}^n)$ is given by

$$AW = (\nabla q_W W^\top - W\nabla q_W^\top)W \tag{3.26}$$

Since A is skew-symmetric, the action of A on W can be mapped to an orthogonal action using the *Cayley transformation*, allowing a parameterized curve on the Stiefel manifold to be constructed as:

$$Y(t) = \left(I + \frac{t}{2}A\right)^{-1}\left(I - \frac{t}{2}A\right)W \tag{3.27}$$

This transformation and choice of parameterization ensures that

$$\begin{aligned} Y(t)^\top Y(t) &= W^\top \left[\left(I + \frac{t}{2}A \right)^{-1} \left(I - \frac{t}{2}A \right) \right]^\top \left(I + \frac{t}{2}A \right)^{-1} \left(I - \frac{t}{2}A \right) W \\ &= W^\top \left(I + \frac{t}{2}A \right) \left(I - \frac{t}{2}A \right)^{-1} \left(I + \frac{t}{2}A \right)^{-1} \left(I - \frac{t}{2}A \right) W \\ &= W^\top \left(I - \frac{t}{2}A \right)^{-1} \left(I + \frac{t}{2}A \right) \left(I + \frac{t}{2}A \right)^{-1} \left(I - \frac{t}{2}A \right) W \\ &= W^\top W \\ &= I \end{aligned} \tag{3.28}$$

i.e. $Y(t)$ remains on the Stiefel manifold for all t, and also

$$\begin{aligned} Y'(t) = &- \left(I + \frac{t}{2}A \right)^{-1} \left(\frac{1}{2}A \right) \left(I + \frac{t}{2}A \right)^{-1} \left(I - \frac{t}{2}A \right) W \\ &+ \left(I + \frac{t}{2}A \right)^{-1} \left(-\frac{1}{2}A \right) W \\ Y'(0) = &-AW \end{aligned} \tag{3.29}$$

i.e. the tangent vector at $t = 0$ is $-AW$, the opposite direction of the gradient. Therefore, $Y(t)$ is a descent curve for $q_W(s, a)$ on the Stiefel manifold.

The two implementations of this descent curve used in the following experimental section are to either solve the $n \times n$ system of equations for a given t

$$\left(I + \frac{t}{2}A \right) Y(t) = \left(I - \frac{t}{2}A \right) W \tag{3.30}$$

or to further simplify the form using Sherman-Morrison-Woodbury formula. Specifically, perform the decomposition $A = UV^\top$ where $U = [\nabla q_W, W]$ and $V = [W, -\nabla q_W]$ and apply the formula to result

$$\left(I + \frac{t}{2}A \right)^{-1} = \left(I + \frac{t}{2}UV^\top \right)^{-1} = I - \frac{t}{2}U \left(I + \frac{t}{2}V^\top U \right)^{-1} V^\top \tag{3.31}$$

where substitution of this form into $Y(t)$ and simplification results

$$Y(t) = W - t \cdot U \left(I + \frac{t}{2} V^\top U \right)^{-1} V^\top W \tag{3.32}$$

which requires inversion of a $2p \times 2p$ matrix. Recall that n is the number of features and p is the cardinality of the action space. Both implementations have their drawbacks and thus both are tested in the following experimentation.

3.2.3 Experiments and Results

The following experiments were conducted on a custom implementation of one-step semi-gradient SARSA built on top of the GNFlow package by Dr. Elizabeth Newman, named *ClassicControl on a Manifold* (see Appendix for details). This implementation utilizes an ε-greedy policy and linear polynomial featurizer of the state-action value function $q_w(s, a)$ with the following parameters

polynomial featurizer order = 8

discount $\gamma = 0.9$

number of episodes = 400

maximum steps = 9000

$\varepsilon_{\text{min}} =$ (varied)

$\varepsilon_{\text{max}} = 1$

initial step-size $\alpha_0 =$ (varied)

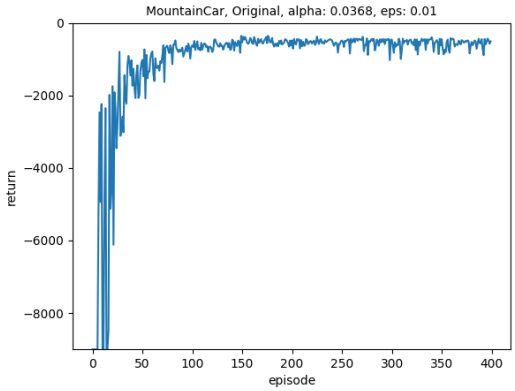
with a step-size regime of $\alpha_{t+1} = \frac{\alpha_t}{1+\sqrt{t}}$ and exploration regime taken logarithmically from ε_{min} to ε_{max} across the number of episodes.

The test environments for the following experiments are the MountainCar and Trajectory environments from the OpenAI Gym package [1], chosen due to the intuitively

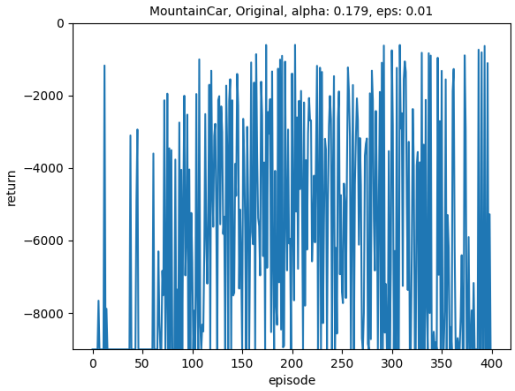
orthogonal nature of their action spaces. As indicated in the parameter list, experiments were performed over a log log sweep of values for $\varepsilon_{\min}$ and α_0 between 0.005 and 0.2, with default values of 0.01 for both paramters. The results denoted `CurvilinearNoInverse` refer to the curve parameterization provided in Eq. (3.30). The `CurvilinearInverse` of Eq. (3.32) method is present in the implementation, but the results are omitted due to them being identical. Lastly the results denoted `Procrustes` refer to the method given in Eq. (3.14)

Three experiments are selected for each method (`Original`, `Procrustes`, and `CurvilinearNoInverse`) across the parameter sweep that represent the best, worst, and an interesting case.

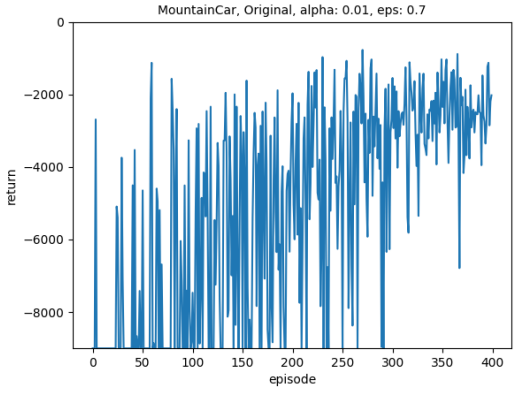
For the MountainCar environment, the `Original` method yields



(a) $\alpha_0 = 0.037$, $\varepsilon = 0.01$



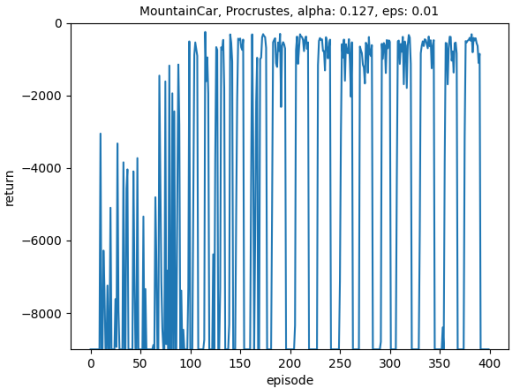
(b) $\alpha_0 = 0.179$, $\varepsilon = 0.01$



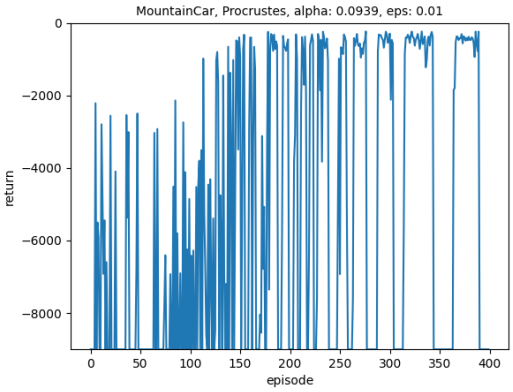
(c) $\alpha_0 = 0.01$, $\varepsilon = 0.07$

Figure 3.1: MountainCar with standard one-step semi-gradient SARSA

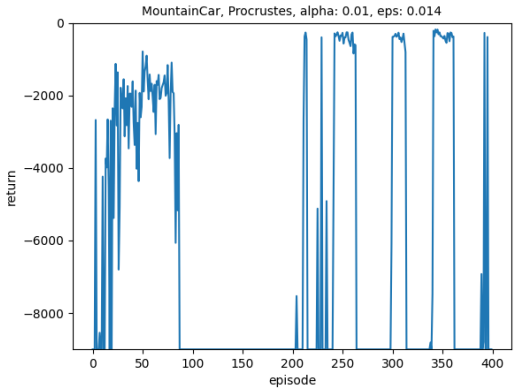
As shown by the examples provided in Figure 3.4, the standard one step semi-gradient SARSA method can achieve rapid convergence for a select choice of parameters — thus verifying the implementation — but is extremely sensitive to both α_0 and $\varepsilon_{\min}$. The `Procrustes` method yields:



(a) $\alpha_0 = 0.127$, $\varepsilon = 0.01$



(b) $\alpha_0 = 0.094$, $\varepsilon = 0.01$

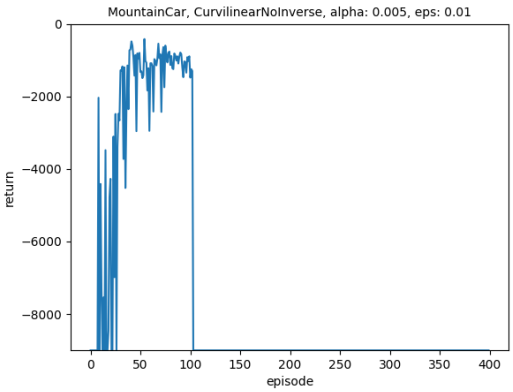


(c) $\alpha_0 = 0.01$, $\varepsilon = 0.014$

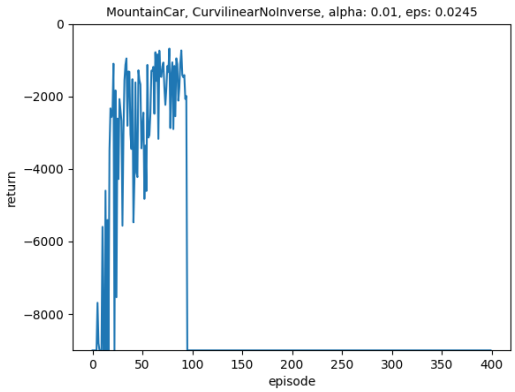
Figure 3.2: MountainCar with orthogonal restriction via Eq. (3.14)

in which an apparent convergence is interspersed with multiple catastrophic stretches of episodes.

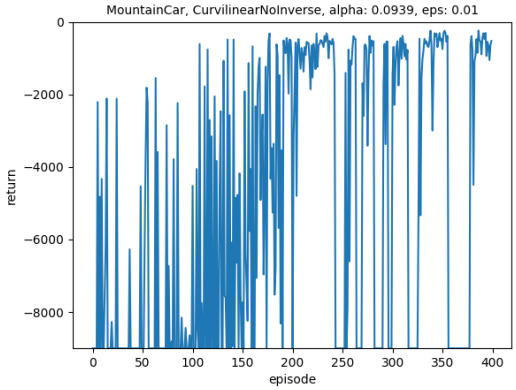
The `CurvilinearNoInverse` method yields a similar result:



(a) $\alpha_0 = 0.005$, $\varepsilon = 0.01$



(b) $\alpha_0 = 0.01$, $\varepsilon = 0.025$



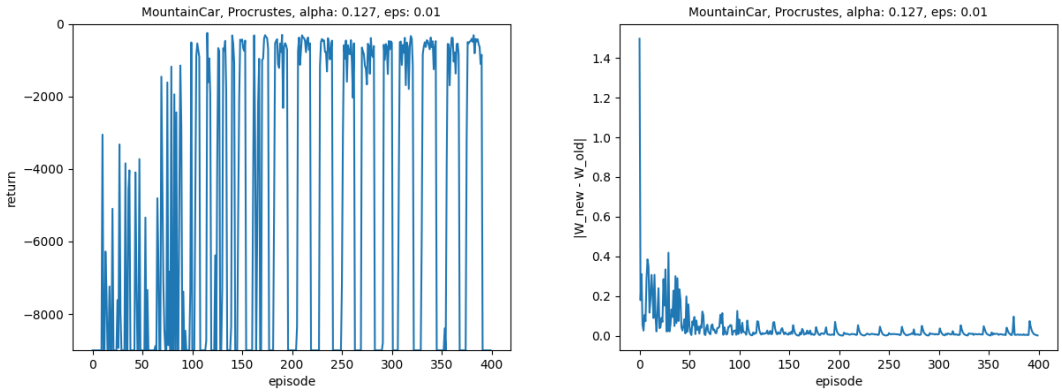
(c) $\alpha_0 = 0.094$, $\varepsilon = 0.01$

Figure 3.3: MountainCar with orthogonal restriction via Eq. (3.30)

in which for some parameter choices the same periodic nature is observed, and for others the catastrophic collapse in training is not recoverable.

The implementation of the orthogonal methods was carefully checked, and indeed the value $\|W^\top W - I\|$ remained less than 10^{-8} for all episodes in both orthogonal methods, indicating that the manifold restriction held with respect to single precision throughout the training. However, neither methods were able to display consistent convergence to the optimal policy without catastrophic collapse. The reason behind this collapse — and particularly the periodic nature observed in some experiments — remains unclear. A potential source of this anomaly may be related to the curvature of the Stiefel manifold at certain points. In areas with extremely high curvature, the

distance between the new iterate generated by the standard update and that generated by the manifold-restricted update may be extremely large. Examining $\|W_{t+1} - W_t\|$ across the episodes lends credence to this hypothesis



(a) $\alpha_0 = 0.127$, $\varepsilon = 0.01$ (b) $\alpha_0 = 0.127$, $\varepsilon = 0.01$

Figure 3.4: MountainCar with Procrustes, return (left) and $\|W_{t+1} - W_t\|$ (right)

in which the catastrophic collapses correspond exactly to a relative spike in $\|W_{t+1} - W_t\|$. The conclusion of this observation is that the Stiefel manifold restricted methods as formulated are — unfortunately — not viable except with additional consideration for the step-size taken, which may require knowledge of the curvature at each iteration.

3.3 Manifold-regularized/preconditioned methods

The purpose of this section is to explore the motivation, development, and current state of second-order optimization in its application to policy-gradient methods in reinforcement learning. In this regard, the theoretical foundations of manifold learning come from the field of information geometry i.e. the study of statistical manifolds. A ubiquitous feature within this field is the construction of an approximations for metric tensor on the manifold of parameters for the purpose of deriving an improved descent iteration as an alternative to stochastic gradient descent. Such efforts are motivated in a similar fashion as the Newton method in contrast to gradient descent

i.e. via a local approximation that captures curvature information. The majority of this section focuses on the use of the *Fisher information matrix* as a metric tensor on the statistical manifold of policy parameters, leading to the method of *natural gradient descent*.

The considerations that motivate the natural gradient lead to the TRPO (Trust-region policy optimization) and PPO (Proximal policy optimization) methods that are considered state-of-the-art in reinforcement learning today. In regard to the natural gradient, much of the difficulty in its application to reinforcement learning comes from the construction of a suitable approximation to the true Fisher information matrix, which otherwise requires taking an expectation over the entire model distribution. In reinforcement learning, however, the information available for sampling is often only obtained through the collection of trajectories. In most cases, this deficiency has been addressed via computation of the *empirical Fisher matrix* as a proxy for the true Fisher matrix, but recent work has shown clear issues in the motivation, properties, and results of this approximation.

3.3.1 Motivation

The motivation behind all second-order optimization methods is to compute a descent direction that yields the greatest decrease in the objective function with respect to a local quadratic model. Consider an iteration on the model parameters as θ_k and a loss function denoted $L(\theta_k)$. The objective is to compute a descent direction δ_k such that the local quadratic model of $L(\theta_k + \delta_k)$ centered at θ_k is minimized, resulting in the iteration $\theta_{k+1} = \theta_k + \alpha_k \delta_k^*$ for some step-size α_k. Specifically, this quadratic model is constructed as

$$M_k(\delta) = L(\theta_k) + \nabla L(\theta_k)^\top \delta + \frac{1}{2}\delta^\top B_k \delta \tag{3.33}$$

where $B_k \in \mathbb{R}^{n \times n}$ is a symmetric matrix that captures the curvature of the loss function at θ_k. The resulting objective is to find

$$\delta_k^* = \arg\min_{\delta} M_k(\delta) \tag{3.34}$$

which can be found as $\delta_k^* = -B_k^{-1} \nabla L(\theta_k)$ when B_k is non-singular and positive definite. In this construction, the choice of $B_k = I$ yields the method of steepest descent and $B_k = \nabla^2 L(\theta_k)$ — where $\nabla^2 L$ denotes the Hessian — yields the Newton method. The choice of $\delta_k^* = \nabla L(\theta_k)$ indeed provides the direction of steepest descent in the case of an orthonormal coordinate system used in Euclidean space. However, this choice does not generalize to the context of a Riemannian manifold in which local distances are influenced by the geometry.

Consider the n-dimensional parameter space of θ to be a generalized Riemannian manifold $(\mathbb{R}^n, g_{ij})$ where g_{ij} is the $(0,2)$-type metric tensor defined on $\mathbb{R}^n$ and denote the tangent space at θ as $\mathcal{T}_\theta$. Specifically, the following properties hold for all $\theta \in \mathbb{R}^n$ (using abstract index notation)

(a) For all $u, v \in \mathcal{T}_\theta$, $g_{ij} u^i v^j = g_{ji} u^i v^j$ i.e. g_{ij} is symmetric

(b) Assume there exists u^i such that $g_{ij} u^i v^j$ for all $v^j \in \mathcal{T}_\theta$, then $u^i = 0$ i.e. g_{ij} is non-degenerate

Amari (2016) [18] provides a construction of the steepest descent direction on a Riemannian manifold. The squared local distance between two points θ and $\theta + d\theta$ on the manifold is given by the quadratic form

$$ds^2 = g_{ij} d\theta^i d\theta^j \tag{3.35}$$

The objective thus becomes to compute the direction $d\theta$ yielding the largest change in the loss function $L(\theta)$. The curvature of the manifold requires restricting the step-size

of $d\theta$ to be of equal magnitude in all directions i.e. for some small constant $\varepsilon > 0$, require

$$g_{ij} d\theta^i d\theta^j = \varepsilon^2 \tag{3.36}$$

and denote $d\theta = \varepsilon a$ where $\|a\|^2 = g_{ij} a^i a^j = 1$. The problem of finding the direction of greatest change thus becomes to maximize

$$L(\theta + d\theta) - L(\theta) = \nabla L(\theta) \cdot \varepsilon a \tag{3.37}$$

subject to $\|a\|^2 = 1$, which can be written as the constrained quadratic problem

$$a^* = \arg\max_a \nabla L(\theta) \cdot a - \lambda g_{ij} a^i a^j \tag{3.38}$$

Denoting $G = [g_{ij}]$, the solution is as $a^* = \frac{1}{2\lambda} G^{-1} \nabla L(\theta)$ and the *natural gradient* of L at θ is denoted $\tilde{\nabla} L(\theta) = G^{-1}(\theta) \nabla L(\theta)$ i.e. $\delta = -\tilde{\nabla} L(\theta)$ is the steepest descent direction.

As mentioned previously, the replacement of $G(\theta)$ with $H(\theta) = \nabla^2 L(\theta)$ yields the Newton method, and indeed choosing the Hessian as an estimate of the metric produces the most accurate local model of the curvature in certain scenarios (specifically when θ lies on a *Hessian manifold* i.e. $G(\theta) = H(\theta)$). The issues with naive implementations of Newton's method are well-known. The most apparent of these drawbacks in any optimization context is the assumption that $H(\theta)$ is positive-definite which is required for the local quadratic model to have a minimum. In the context of neural networks with n parameters (where n is typically quite large), the calculation and storage of the n^2 elements of the Hessian — not to mention the cost of solving $H(\theta)\delta = -\nabla L(\theta)$ at each iteration — result in an immensely expensive method. In addition to these drawbacks, Martens (2020) [25] describes a more conceptual issue that may arise: that the local optimality of the Hessian-based approximation can generalize poorly to

the broader space. He describes a worst-case scenario in which the quadratic model $M(\delta)$ is an extremely poor local approximation of $L(\theta)$ but nonetheless has the correct minimizer at θ^*. In this regard, the choice of approximations made can affect the near/far-sightedness of the method.

3.3.2 Geometry of statistical manifolds

Consider the parameterized conditional probability distribution $\pi_\theta(a|s)$ wholly specified by θ — purposely notated as such since the following analysis will later be applied to stochastic policies in reinforcement learning. Denote S the space of all distributions of this type (e.g. all neural networks of a given structure across the space of θ) and consider θ to be the local coordinate system of S. More accurately, the space under consideration is the quotient space $\tilde{S} = S \backslash R$ where R is defined by the equivalence relation $\theta \sim \theta'$ when

$$\pi_\theta(a|s) = \pi_{\theta'}(a|s) \tag{3.39}$$

i.e. the two parameters specify identical conditional distributions. However, for ease of notation, the space under consideration will simply be denoted S. The divergence between two points $\theta, \theta' \in S$ is defined by the Kullback-Leibler (KL) divergence between the two distributions they specify as

$$D_{KL}(\pi_\theta : \pi'_\theta) = \int \pi_\theta(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} \, ds \, da \tag{3.40}$$

Note that the KL-divergence is not symmetric and thus cannot be considered a metric. Choosing D_{KL} as a notion of distance, however, allows for the specification of the Riemannian structure of S. Consider some loss function $L(\theta)$. With respect to D_{KL} as the divergence between two coordinates, the steepest descent (natural gradient)

direction upon the manifold is $F(\theta)^{-1}\nabla L(\theta)$ where

$$F(\theta) = \nabla^2_{\theta'} D_{KL}(\pi_\theta : \pi_{\theta'})|_{\theta=\theta'} \tag{3.41}$$

i.e. the Hessian of the KL-divergence at θ. Specifically, the matrix $F(\theta)$ is called the *Fisher information matrix* and takes the form

$$F(\theta) = \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}[\nabla \log \pi_\theta(a|s) \nabla \log \pi_\theta(a|s)^\top] \tag{3.42}$$

pre-emptively using the RL notation $s \sim \mu$ to denote the sampling of s from the stationary distribution $\mu(s)$. Under certain regularity conditions (for exchanging integration and differntiation), this form of the Fisher matrix can be shown to be equivalent to the Hessian of D_{KL} as follows:

$$\begin{aligned}
F(\theta) &= \nabla^2_{\theta'} D_{KL}(\pi_\theta : \pi_{\theta'})|_{\theta=\theta'} \\
&= \nabla^2_{\theta'} \int \pi_\theta(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)} \, ds \, da \, \Big|_{\theta=\theta'} \\
&= -\int \pi_\theta(a|s) \, \nabla^2_\theta \log \pi_\theta(a|s) \, ds \, da \qquad (*) \\
&= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}[-\nabla^2_\theta \log \pi_\theta(a|s)] \\
&= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[-\frac{1}{\pi_\theta(a|s)} \nabla^2_\theta \pi_\theta(a|s)\right] + \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[\frac{1}{\pi_\theta(a|s)^2} \nabla \pi_\theta(a|s) \nabla \pi_\theta(a|s)^\top\right]
\end{aligned} \tag{3.43}$$

where upon taking expectation and exchanging differentiation/integration (*) the first term reduces to $\nabla^2_\theta \, 1 = 0$ while the second term is rearranged as

$$\begin{aligned}
F(\theta) &= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[\frac{1}{\pi_\theta(a|s)^2} \nabla \pi_\theta(a|s) \nabla \pi_\theta(a|s)^\top\right] \\
&= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[\frac{1}{\pi_\theta(a|s)} \nabla \pi_\theta(a|s) \frac{1}{\pi_\theta(a|s)} \nabla \pi_\theta(a|s)^\top\right] \\
&= \mathbb{E}_{s\sim\mu, a\sim\pi_\theta}\left[\nabla \log \pi_\theta(a|s) \nabla \log \pi_\theta(a|s)^\top\right]
\end{aligned} \tag{3.44}$$

The Fisher information matrix in this form has the property that for two points θ and $\theta + d\theta$ on the statistical manifold, the local divergence between them is

$$D_{KL}(\pi_\theta : \pi_{\theta+d\theta}) = \frac{1}{2} d\theta^\top F(\theta) d\theta \tag{3.45}$$

This property allows the Fisher information matrix to be used as a Riemannian metric. In fact, Amari (1987) [3] proves that the Fisher metric is the unique invariant metric on the manifold characterized by KL-divergence.

3.3.3 Natural policy-gradient

The choice of notation for the parameterized conditional probability distribution $\pi_\theta(a|s)$ in the previous subsection foreshadows the application of the natural gradient to reinforcement learning. Amari (1998b) [2] makes the case for the natural gradient as *Fisher efficient* in the sense that with step-size choice $\alpha_k = \frac{1}{k}$, the parameter estimates are unbiased and their covariance approaches the Cramer-Rao lower bound (more accurately: specifically in the case of a maximum-likelihood cost function). As described in Section 2.2.5, one of the primary motivations for the development of new policy-gradient methods is the desire to decrease the variance in parameter estimates. The use of such tools from information geometry serve not only as the theoretical basis for naive implementations of the natural gradient in RL, but also as motivation for the development of methods such as Trust-region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) as methods that, at their basis, aim to bound the KL-divergence of the new iterate θ_{k+1} i.e. constrain $D_{KL}(\pi_{\theta_{k+1}} : \pi_{\theta_k}) < \delta$ for some radius δ.

Kakade (2001) [21] introduces the natural policy gradient in a specific context for actor-critic methods with — as he describes — a compatible function approximation. However, the method is generalizable to more complex function approximations. The

construction is as follows:

Let $\psi_\theta(s,a) = \nabla_\theta \log \pi_\theta(s,a)$ and $f_w(s,a) = w^\top \psi_\theta(s,a)$ where f is a linear function approximation of $q_\pi(s,a)$ parameterized by w. The function approximation f is compatible with the policy in the sense described by Sutton (1999) [39] which allows for an unbiased approximation of the true policy-gradient with respect to maximizing the undiscounted reward. Denote this reward $J(\theta)$ and consider the standard policy gradient

$$\nabla_\theta J(\theta) = \sum_{s,a} \mu(s) \nabla \pi_\theta(s,a) q_\pi(s,a) \tag{3.46}$$

where $\mu(s)$ is the stationary distribution. Let w^* minimize the squared error $\epsilon(w,\pi)$ i.e.

$$w^* = \arg\min_w \epsilon(w,\pi) = \arg\min_w \sum_{s,a} \mu(s)\pi_\theta(s,a)(f_w(s,a) - q_\pi(s,a))^2 \tag{3.47}$$

implying that $\nabla_w \epsilon(w^*,\pi) = 0$ i.e.

$$\sum_{s,a} \mu(s)\pi_\theta(s,a)\psi_\theta(s,a)(\psi_\theta(s,a)^\top w^* - q_\pi(s,a)) = 0 \tag{3.48}$$

or equivalently

$$\left(\sum_{s,a} \mu(s)\pi_\theta(s,a)\psi_\theta(s,a)\psi_\theta(s,a)^\top \right) w^* = \sum_{s,a} \mu(s)\pi_\theta(s,a)\psi_\theta(s,a) q_\pi(s,a) \tag{3.49}$$

where substitution with the previous definitions results $F(\theta)w^* = \nabla_\theta J(\theta)$

Furthermore, let w^* minimize the approximation error $\epsilon(w,\pi)$ and construct the iteration on policy parameters $\theta' = \theta + \alpha F(\theta)^{-1} \nabla_\theta J(\theta)$. Then $\Delta\theta = \alpha w^*$ by compatibility

of the function approximator and to first order,

$$\begin{aligned}\pi_{\theta'}(s,a) &= \pi_\theta(s,a) + \nabla_\theta \pi_\theta(s,a)\Delta\theta + \mathcal{O}(\alpha^2) \\ &= \pi_\theta(s,a)(1 + \alpha\psi_\theta(s,a)^\top w^*) + \mathcal{O}(\alpha^2) \\ &= \pi_\theta(s,a)(1 + \alpha f_{w^*}(s,a) + \mathcal{O}(\alpha^2)\end{aligned} \tag{3.50}$$

i.e. the iteration converges to the greedy action with respect to the the local linear approximation $f_{w^*}(s,a)$.

While the method outlined in Kakade (2001) [21] is specific to the compatibile function approximation, the same natural gradient iteration on policy parameters can be applied to methods with more advanced approximations such as those employing neural networks.

As mentioned previously, the basic motivation behind state-of-the-art policy gradient methods such as TRPO and PPO is to impose a constraint on the KL-divergence of the newly computed step. Consider the current iterate θ_k and the selection of new iterate θ. Following Section 3.3.2, the KL-divergence between the two iterates can be locally approximated as

$$D_{KL}(\pi_{\theta_k} : \pi_\theta) \approx \frac{1}{2}(\theta - \theta_k)^\top F(\theta_k, s)(\theta - \theta_k) \tag{3.51}$$

Denote $\bar{F}(\theta) = \mathbb{E}_{\tau\sim\pi_{\theta_k}}[F(\theta_k, s)]$ the *average Fisher information*. Constraining the new iterate to lie within a trust-region of radius δ is to require

$$\frac{1}{2}(\theta - \theta_k)^\top \bar{F}(\theta)(\theta - \theta_k) \leq \delta \tag{3.52}$$

which produces the iteration $\theta = \theta_k + \alpha_k \bar{F}(\theta)^{-1}\nabla J(\theta_k)$ where $\nabla J(\theta_k)$ is the standard

policy gradient. Furthermore, the constraint on the KL-divergence requires that

$$\alpha_k = \frac{\delta}{\sqrt{\nabla J(\theta_k)^\top \bar{F}(\theta)^{-1} \nabla J(\theta_k)}} \tag{3.53}$$

With the addition of conditions for expanding or contracting the trust-region radius δ, the TRPO method as introduced by Schulman (2015) [31] is complete. In this setup, the natural policy gradient can be viewed as the same method but without adherence to the trust-region constraint.

Introduced by Schulman (2017) [32], PPO does not use the Fisher matrix directly but still aims for a similar goal. Specifically, denote the ratio between policies at the current θ_k and new θ as

$$r_\theta(s,a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} \tag{3.54}$$

At each iteration, PPO aims to maximize

$$\mathbb{E}_{\tau\sim\pi_{\theta_k}}[r_\theta(s,a) A_{\theta_k}(s,a)] \tag{3.55}$$

where A_{θ_k} is a chosen advantage function. There are two versions of PPO in current use: PPO-Clip and PPO-Penalty.

The objective function used in PPO-Clip for some choice of ε

$$L(s,a,\theta,\theta_k) = \mathbb{E}_{\tau\sim\pi_{\theta_k}}\big[\min\{r_\theta(s,a)A_{\theta_k}(s,a),\ \text{clip}(\varepsilon, A_{\theta_k}(s,a))\}\big] \tag{3.56}$$

where

$$\text{clip}(\varepsilon, A_{\theta_k}(s,a)) = \begin{cases} (1+\varepsilon)A & \text{if } A \geq 0 \\ (1-\varepsilon)A & \text{if } A < 0 \end{cases} \tag{3.57}$$

The objective function used in PPO-Penalty for some choice of β

$$L(s,a,\theta,\theta_k) = \mathbb{E}_{\tau\sim\pi_{\theta_k}}\big[r_\theta(s,a)A_{\theta_k}(s,a) - \beta D_{KL}(\pi_{\theta_k} : \pi_\theta)\big] \tag{3.58}$$

as is clearly apparent for PPO-Penalty — and slightly disguised in PPO-Clip — each method is motivated by a desire to constrain the KL-divergence of the next iterate while also producing a suitably large improvement in the objective.

3.3.4 Empirical Fisher information matrix

Recall the natural policy gradient method outlined by Kakade (2001) [21] in which the iteration on policy parameters is defined as

$$\theta' = \theta + \alpha F(\theta)^{-1} \nabla_\theta J(\theta) \tag{3.59}$$

In this method, the Fisher information matrix is estimated in an online fashion as the weighted sum of outer products

$$F_{t+1} = F_t + \nabla \log \pi_{\theta_t}(a_t|s_t) \nabla \log \pi_{\theta_t}(a_t|s_t)^\top \tag{3.60}$$

where for a trajectory of length T, Kakade presents the estimate $\frac{F}{T}$ as a consistent estimate of F. However, recent work in the study of Fisher information has challenged this notion and drawn distinctions in the properties of the true Fisher information matrix and the *empirical Fisher information matrix*. Specifically, the true Fisher matrix (in the notation of RL policies) is given as

$$F(\theta) = \mathbb{E}_{a \sim \pi_\theta, s \sim \mu(s)}[\nabla \log \pi_{\theta_t}(a_t|s_t) \nabla \log \pi_{\theta_t}(a_t|s_t)^\top] \tag{3.61}$$

Notably, the expectation of the outer product of gradients is taken with respect to both the conditional action distribution specified by the policy, as well as the stationary distribution $\mu(s)$. Indeed only when the expectation is taken over the entire domain of both distributions does the Fisher matrix hold the desired properties that produce a suitable metric for the statistical manifold. In Amari (1998) [4], the analysis and

examples provided consider an explicit formulation of the Fisher based on a stationary distribution $\mu(s) \sim N(0, 1)$ for a basic perceptron model with no hidden layers. In the majority of real world use-cases — especially in reinforcement learning with deep neural networks — the total information of the distributions required to explicitly compute the Fisher matrix is simply unavailable. In lieu of the true Fisher, a common choice is to instead to compute the empirical Fisher as an estimate via

$$\bar{F} = \frac{1}{T} \sum_{(a,s)\in\tau} \nabla \log \pi_\theta(a|s) \nabla \log \pi_\theta(a|s)^\top \tag{3.62}$$

Recent work, however, has challenged the empirical Fisher as a suitable estimate and has demonstrated its shortcomings in application to the natural gradient where, in many cases, it underperforms significantly in comparison to natural gradient ascent/descent with the true Fisher or even standard gradient ascent/descent [22]. Specifically, the critical property of the true Fisher matrix as converging to the Hessian of the loss function in expectation does not hold for the empirical Fisher in general. Similarly, the true Fisher can be shown as equivalent to the commonly used Generalized Gauss-Newton (GGN) matrix under certain conditions, but this equivalency does not hold for the empirical Fisher — a detailed analysis of which given in Kunstner (2020) [22] — and natural gradient experiments have displayed significant underperformance in comparison to the GGN.

Martens (2020) provides a simple example to demonstrate these shortcomings. Consider the case with $n = 1$ parameters, predictive distribution $\pi_\theta \sim N(\theta, 1)$, state-action space $S = \{(0, 0)\}$, and loss function $L(\theta) = \frac{1}{2}\theta^2$. In this example, loss gradient $\nabla L(\theta) = \theta$, empirical Fisher $\bar{F}(\theta) = \theta^2$, and true Fisher $F(\theta) = 1$. The iteration

utilizing the empirical Fisher for some exponent $\frac{1}{2} \leq \xi \leq 1$ given as

$$\begin{aligned} \theta_{k+1} &= \theta_k - \alpha_k(\bar{F}(\theta_k)^{\xi})^{-1}\nabla L(\theta_k) \\ &= \theta_k - \alpha_k(\theta_k^2)^{-\xi}\theta_k \\ &= (1 - \alpha_k|\theta_k|^{-2\xi})\theta_k \end{aligned} \tag{3.63}$$

fails to converge to the minimizer $\theta = 0$ unless $\xi < 1$ and $a_k \to 0$ sufficiently fast. In the case of convergence, the rate is limited to the rate of convergence of $\alpha_k \to 0$. In contrast, the iteration utilizing the true Fisher given as

$$\begin{aligned} \theta_{k+1} &= \theta_k - \alpha_k F^{-1}\nabla L(\theta_k) \\ &= \theta_k - \alpha_k\theta_k \\ &= (1 - \alpha_k)\theta_k \end{aligned} \tag{3.64}$$

experiences linear convergence with rate $|1 - \alpha|$ for any fixed step-size $0 < \alpha_k = \alpha < 1$

Solutions to the shortcomings of the empirical Fisher suggested in recent literature center around building truly unbiased approximations to the true Fisher. Kunstner (2020) [22] suggests a Monte-Carlo approximation to the true Fisher obtained by re-sampling actions a from $\pi_\theta(a|s)$ and computing the estimate as

$$\hat{F}(\theta) = \sum_n \nabla \log \pi_\theta(a_n|s)\nabla \log \pi_\theta(a_n|s)^\top \tag{3.65}$$

which, while noisy, produces an unbiased estimate of the true Fisher matrix.

The most significant advancement in the construction of the true Fisher matrix in the context of neural networks comes from Martens (2015) [26] which introduces the Kroenecker-factored Approximate Curvature (KFAC). The details of its construction are fairly involved and will thus be summarized only briefly:

Consider the output distribution $p(y|x)$ of a neural network of ℓ layers with weights

$W \in \mathbb{R}^{C_{in} \times C_{out}}$ where C_{in} and C_{out} denote the number of input and output neurons connected to the layer respectively. Denote the input activation vector $a \in \mathbb{R}^{C_{in}}$ and the pre-activation vector for the next layer as $s = Wa$. Construct the loss function as log-likelihood i.e. $L = \log p(x, y)$ where the gradient with respect to the weights is given as $\nabla_W L = (\nabla_s L)a^\top$. Denote the true Fisher corresponding to layer ℓ as F_ℓ and construct the approximation $\hat{F}_\ell$ as

$$\begin{aligned} F_\ell &= \mathbb{E}[\text{vec}\{\nabla_W L\}\text{vec}\{\nabla_W L\}^\top] \\ &= \mathbb{E}[aa^\top \otimes \nabla_s L(\nabla_s L)^\top] \\ &\approx \mathbb{E}[aa^\top] \otimes \mathbb{E}[\nabla_s L(\nabla_s L)^\top] \\ &= \hat{F}_\ell \end{aligned} \tag{3.66}$$

where critically, an assumption of independence between the gradients and the second-order information of the activations is being made. Since the natural gradient requires $\hat{F}_\ell^{-1}$, the properties of the Kroenecker product i.e. $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$ are utilized to construct — in the case of the original implementation in Martens (2020) [25] — block diagonal or block tri-diagonal approximations of the true F_ℓ^{-1}. The advantage of KFAC thus lies in the parallelization facilitated by the block matrix structure, as well as the fact that storage and computation is required only on matrices of size W. George (2018) [12] improves KFAC by introducing Eigenvalue-corrected KFAC (EKFAC) which — as a cursory summary — computes eigendecompositions of the two terms of the Kroenecker product and rescales the diagonal matrix to better match the true second moments of the policy gradient.

3.3.5 Approximate-Fisher natural policy-gradient

This subsection will briefly summarize current literature on the application of Fisher approximations to the natural policy-gradient. Following the development of Kroenecker-

factored Approximate Curvature by Martens (2015) [26], the first appearance of KFAC applied to reinforcement learning appears in Wu (2017) [43] with the introduction of Actor-Critic Kroenecker-factored Trust Region (ACKTR), which indicated significant performance increase over Advantage Actor-Critic (A2C) and TRPO implementations with similar parameters. In the OpenAI Gym benchmark environments, ACKTR displayed increases in sample efficiency of up to 10 times over the other methods, as well as modest improvements in wall-clock time. The original ACKTR as implemented performs the natural gradient update only upon the actor, but further experiments in natural gradient descent with respect to the critic network indicate slight improvements, with better performance in regard to variance arising from using the Generalized Gauss-Newton (GGN) instead. In Wu (2018) [36], natural gradient ascent using KFAC is applied to PPO yielding modest improvements over standard PPO (i.e. with an SGD update) and underperforming in comparison to ACKTR in all benchmarks.

In Gebotys (2022) [11], implementation of natural policy gradient using

1. Diagonal approx. (store and estimate only block diagonals of F)
2. Hessian-free (HF) approx. (e.g. TRPO)
3. Kroenecker-factor approx. (i.e. KFAC/EKFAC)
4. Woodbury approx. (TENGraD, $m \times m$ block diagonal where m is batch-size)

are compared across seven MuJoCo (OpenAI Gym) environments. In all experiments, the diagonal approximation performed the worst, Woodbury approximation (TENGraD) performed the best, and EKFAC consistently outperformed KFAC. However, the computational cost of TENGraD is prohibitively expensive for batches of appreciable size.

The current literature thus demonstrates the utility of information geometric techniques to reinforcement learning, verifying the construction of the policy parameter

space as a generalized Riemannian manifold with the Fisher information matrix as the metric tensor. With the theoretical basis for the natural gradient from Amari (1998) [4] requiring the true Fisher matrix, the most significant advancements in natural policy gradient methods has been the improvement of Fisher approximations. In this sense, the curvature of the policy parameter space has proven to be an essential consideration for developing robust, efficient RL methods.

3.3.6 Experiments and Results

The following experiments were conducted on a custom implementation of Proximal Policy Optimization (both Penalty and Clip varieties) with Generalized Advantage Estimation built using the TorchRL package, named *Geometric Control* (see Appendix for details). The implementation utilizes a size $[64, 64]$ multi-layer perceptron with Tanh activations for both the policy and value function approximation. The policy MLP trains the parameters `loc` and `scale` for a probabilistic actor with a normalized Tanh distribution i.e. $\pi_\theta \sim \text{TanhNormal}(\texttt{loc}, \texttt{scale})$, while the value MLP estimates the advantage. Both MLPs are orthonormal initialized. The advantage used is the Generalized Advantage Estimate (GAE) presented in section 2.2.5

The purpose of these experiments are to examine the difference between an empirical Fisher distribution and one obtained via Monte-Carlo methods (i.e. sampling the policy). Two algorithms are implemented — PPO-Penalty and PPO-Clip — as well as two test environments from the OpenAI Gym - MuJoCo package: *HalfCheetah-v4* and *InvertedPendulum-v4* [1]. The following parameters are used

frames per batch $= 2000$

total frames $= 200,000$

mini-batch size $= 100$

loss optimization epochs $= 10$

learning rate $\lambda = 3 \times 10^{-4}$

discount $\gamma = 0.99$

GAE learning rate $\lambda_{GAE} = 0.95$

PPO-Clip $\varepsilon = 0.2$

PPO-Penalty KL-target $d = 0.01$

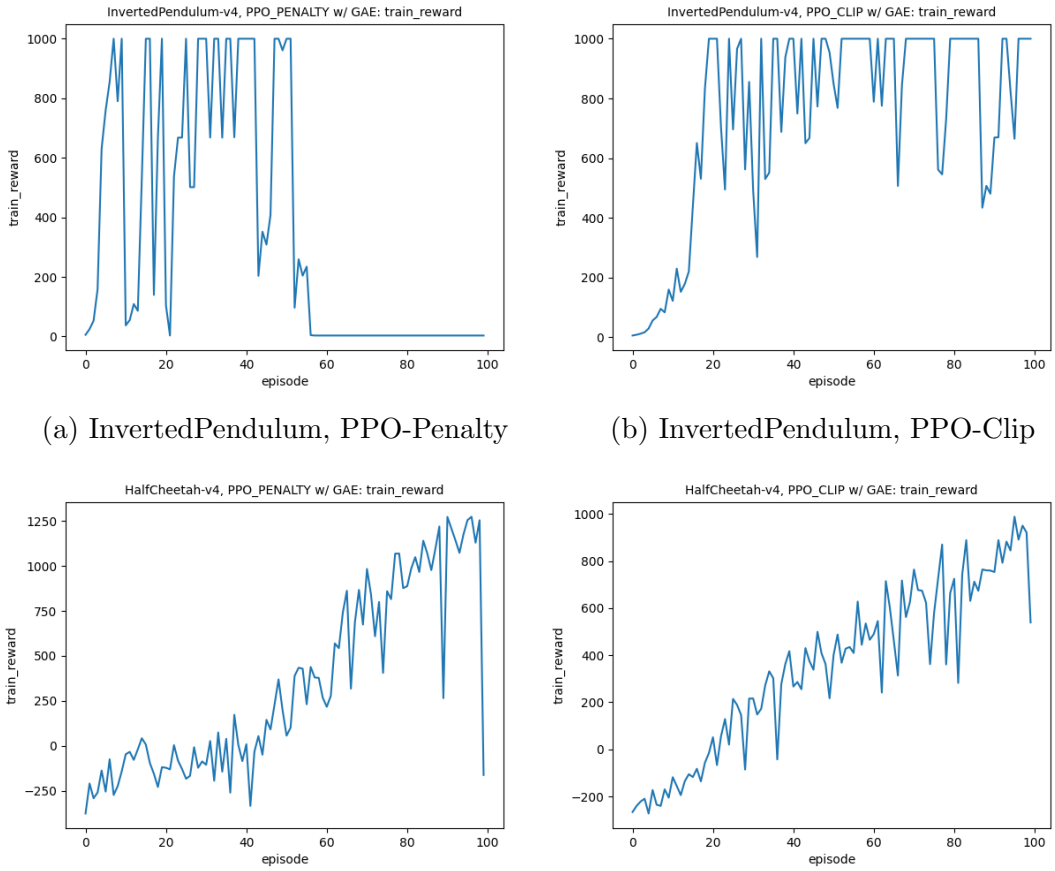
PPO-Penalty $\beta = 1$

Monte-Carlo Fisher sample size = 20

The Monte-Carlo and empirical Fisher approximations are computed as outlined by Eq. (3.65) and Eq. (3.62) respectively. Specifically, the estimate is only collected for every mini-batch on the last optimization epoch (i.e. $2000/100 = 20$ estimates per batch. For the Monte-Carlo approximation, during each mini-batch in the last epoch a sample of 20 actions is collected from the policy distribution π_θ and $\nabla \log \pi_\theta$ is computed for the sample, the sum of which results in the Fisher approximation for the current batch (as per Kunstner (2020) [22]) For the empirical Fisher approximation, during each mini-batch in the last epoch the current log policy gradient is collected and the outer products are averaged over the batch.

It is important to note that since the goal of these experiments is to compare the two Fisher approximations, a full training (e.g. with 1,000,000 total frames) was not conducted due to both time and computational cost restrictions, as well as the fact that the training rewards are not the object of study (the PPO implementations and test environments used are standard).

The observed reward from each algorithm-environment pair is



(a) InvertedPendulum, PPO-Penalty

(b) InvertedPendulum, PPO-Clip

(c) HalfCheetah, PPO-Penalty

(d) HalfCheetah, PPO-Clip

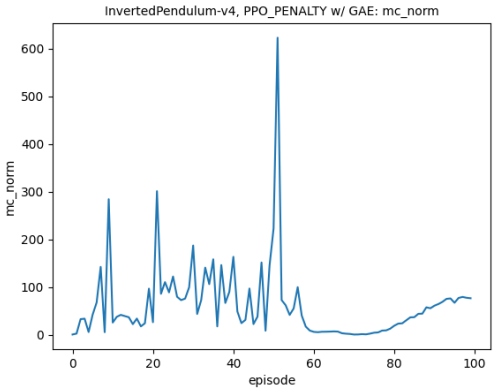
Figure 3.5: PPO training rewards per batch

Although the failure of PPO-Penalty on InvertedPendulum is anomalous, the rest of the experiments display the expected early-training behavior and verify the implementation.

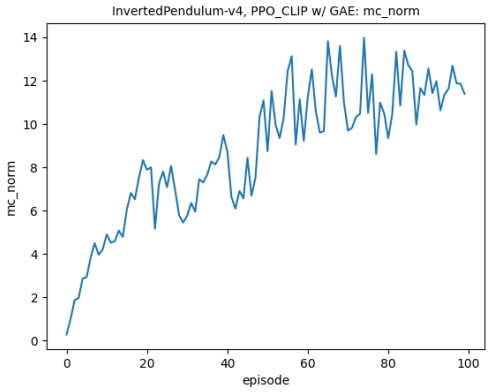
Two quantities were tracked for each Fisher approximation in these experiments: the 2-norm and the trace. Since the eigenvalues of the Hessian provide the principal curvatures — and the true Fisher is equivalent to the Hessian in expectation via Eq. (3.43)— the 2-norm is tracked as a measure of the largest principal curvature. There are two reasons for tracking the trace: First, the trace of the Hessian of a function defined on a Riemannian manifold provides the divergence of the gradient field. Secondly, from the theory of optimal experimental design, the trace of the

Fisher information matrix is used as a proxy for the average variance of estimates — specifically, maximizing the Fisher trace results in minimizing the average variance (called *A-optimality*), thus the trace is tracked.

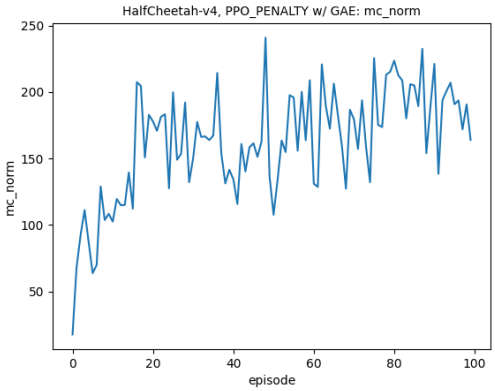
The norm of the Monte-Carlo Fisher approximation for each batch:



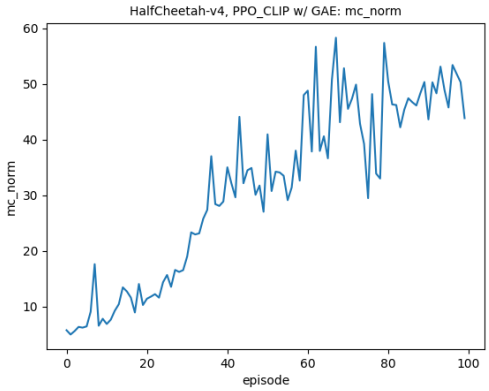
(a) InvertedPendulum, PPO-Penalty



(b) InvertedPendulum, PPO-Clip



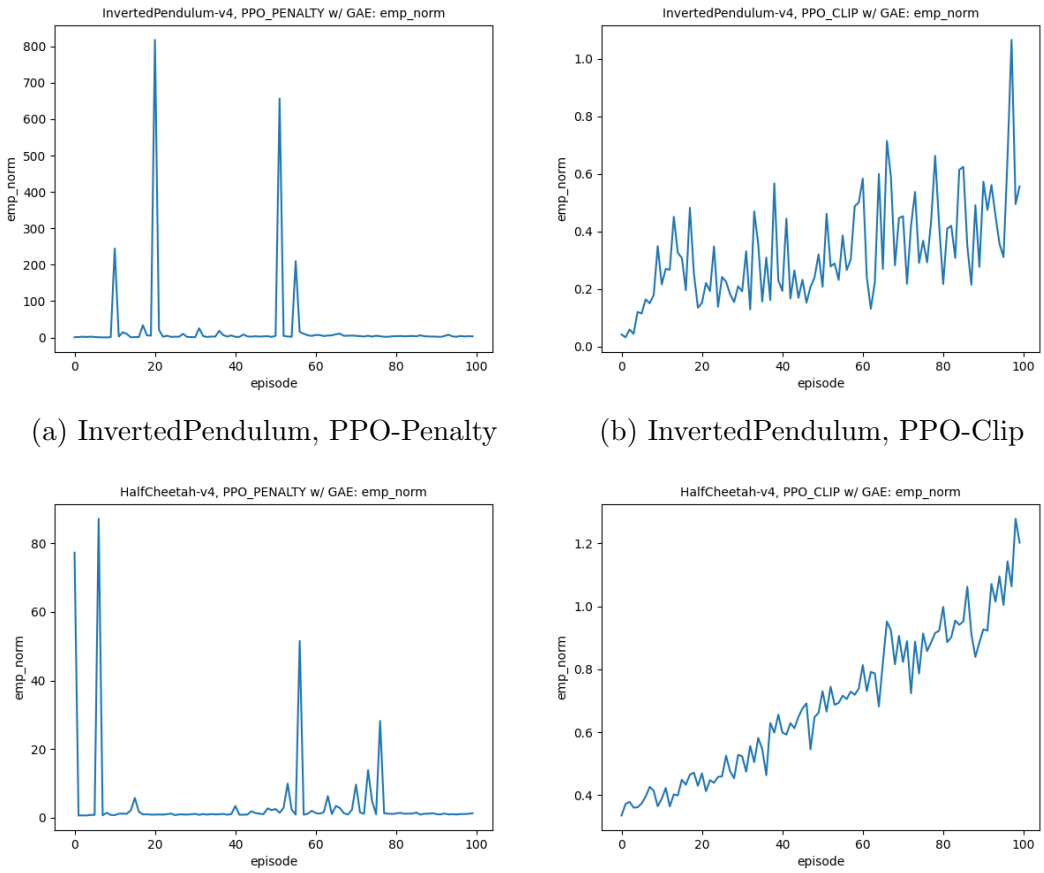
(c) HalfCheetah, PPO-Penalty



(d) HalfCheetah, PPO-Clip

Figure 3.6: 2-norm of Monte-Carlo Fisher approximation

The norm of the empirical Fisher approximation for each batch:



(a) InvertedPendulum, PPO-Penalty

(b) InvertedPendulum, PPO-Clip

(c) HalfCheetah, PPO-Penalty

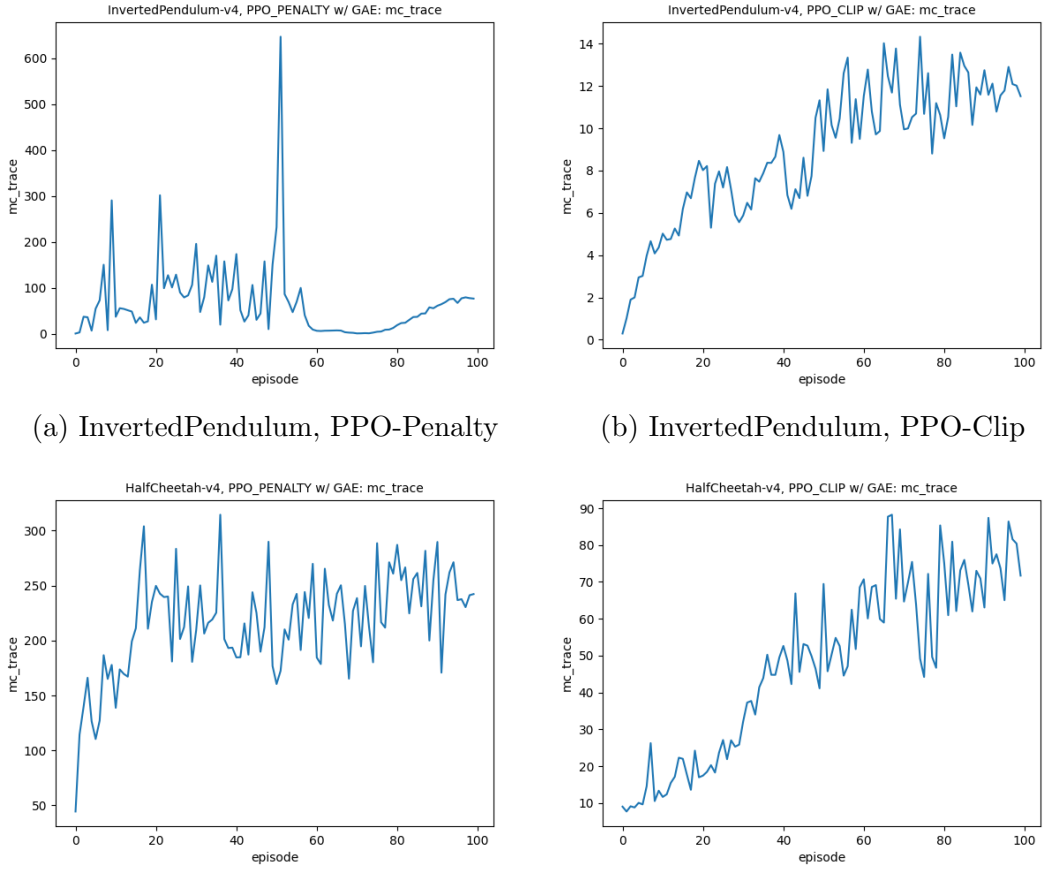
(d) HalfCheetah, PPO-Clip

Figure 3.7: 2-norm of empirical Fisher approximation

Comparison between the norms of each approximation, particularly when paying close attention to the scales, displays stark differences. For the PPO-Penalty experiments on both environments, the norm of the empirical Fisher [Fig. 3.7 (a) (c)] remains near-zero throughout a majority of the training arc but interspersed with spikes many orders of magnitude large, indicating poor conditioning. However — and ignoring the anomalous PPO-Penalty on InvertedPendulum results — the norm of the Monte-Carlo Fisher [Fig. 3.6 (a) (c)] remains within an order of magnitude for a majority of the training. For the PPO-Clip experiments on both environments, the results are starkly different than PPO-Penalty. Both the Monte-Carlo and empirical

Fisher norms exhibit a consistent, stable increase as the training proceeds, without any of the large spikes seen with PPO-Penalty. While sharing a similar trend, however, the Monte-Carlo and empirical Fisher norms differ by an order of magnitude as the empirical Fisher norm barely exceeds a magnitude of 1.

The trace of the Monte-Carlo Fisher approximation for each batch:



(a) InvertedPendulum, PPO-Penalty

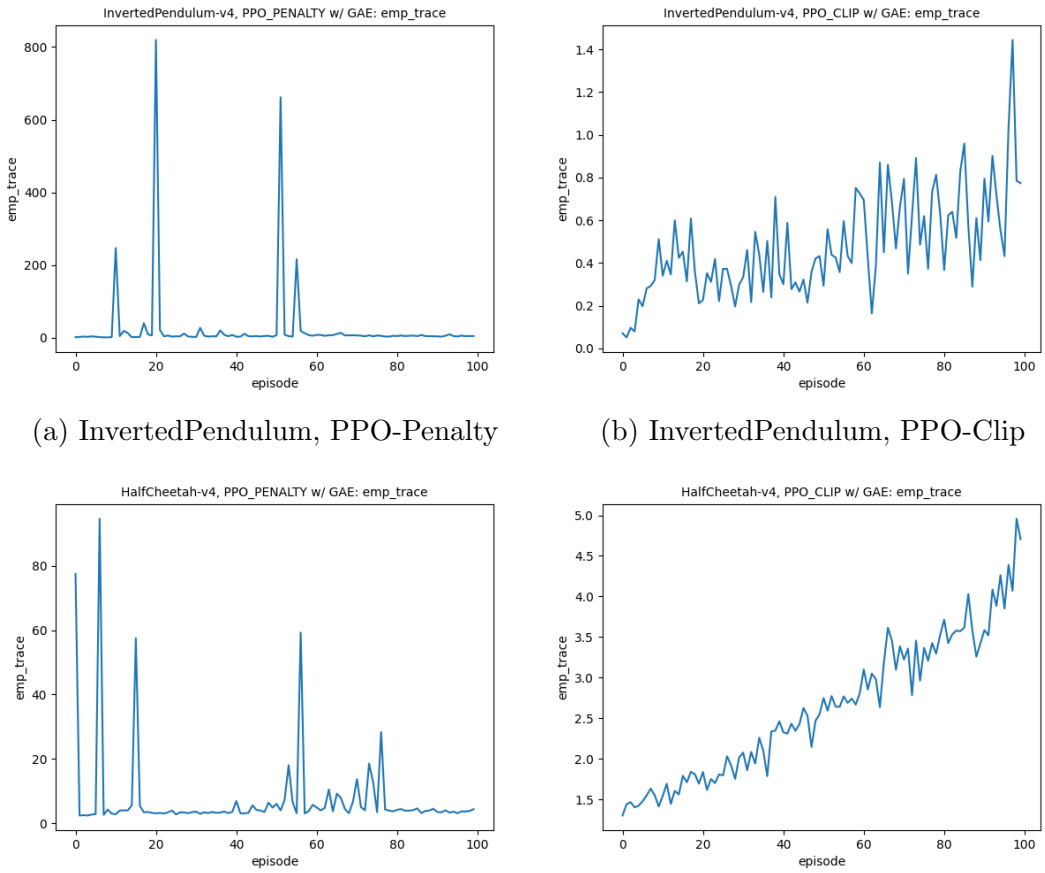
(b) InvertedPendulum, PPO-Clip

(c) HalfCheetah, PPO-Penalty

(d) HalfCheetah, PPO-Clip

Figure 3.8: Trace of Monte-Carlo Fisher approximation

The trace of the empirical Fisher approximation per batch:



(a) InvertedPendulum, PPO-Penalty

(b) InvertedPendulum, PPO-Clip

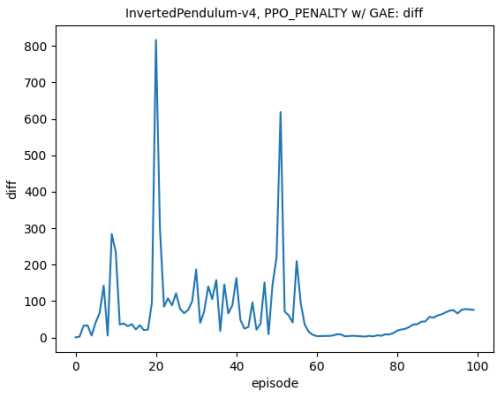
(c) HalfCheetah, PPO-Penalty

(d) HalfCheetah, PPO-Clip

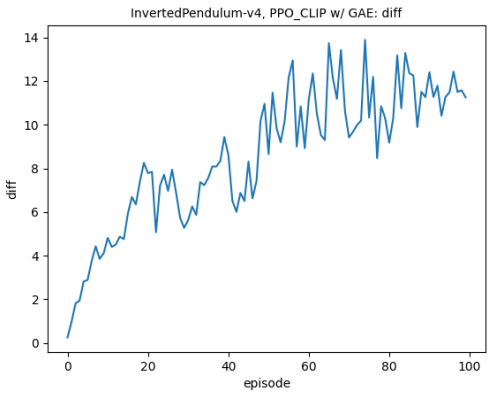
Figure 3.9: Trace of empirical Fisher approximation

Comparisons between the traces of the two Fisher approximations yield the same conclusions as the norm comparisons. Surprisingly, the results of the experiments indicated that the norm and trace of each approximation tracked extremely closely in regards to trend/change (not necessarily in actual value).

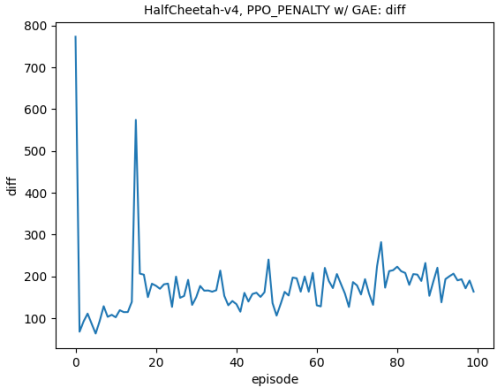
Finally, the 2-norm difference between the Monte-Carlo and empirical Fisher



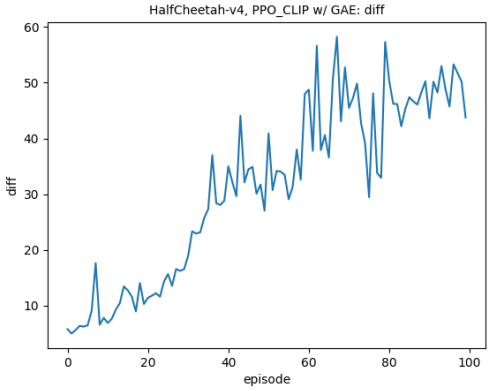
(a) InvertedPendulum, PPO-Penalty



(b) InvertedPendulum, PPO-Clip



(c) HalfCheetah, PPO-Penalty



(d) HalfCheetah, PPO-Clip

Figure 3.10: Normed difference between MC and empirical Fisher approximations

Examining the 2-normed difference between the two approximations across the training arc indicates a constant difference for PPO-Penalty [Fig 3.10 (a) (c)] and divergence for PPO-Clip [Fig 3.10 (b) (d)]. The normed difference does not appear to converge to zero in any case indicating that the two approximations do not appear to estimate the same quantity. Specifically, if Kunstner (2020) [22] is correct in that the Monte-Carlo approximation is unbiased with respect to the true Fisher, the experiment validates the claim that the empirical Fisher is biased.

The comparisons made in these experiments are for the purpose of both comparing the two Fisher approximations, as well as examining basic information about the

curvature of the policy parameters space across the training arcs for PPO-Penalty and PPO-Clip. From a qualitative perspective, it is reasonable to conclude from these results that the Monte-Carlo approximation enjoys much greater numerical stability than its empirical counterpart. Similarly, PPO-Clip appears to enjoy greater stability (with respect to Fisher approximations) than PPO-Penalty. Using the PPO-Clip results as a reference point, it appears that the general trend is an increase in the Fisher norm (i.e. greater curvature) as well as an increase in the Fisher trace (i.e. lower variance of estimates). These qualitative observations line up with intuition about the expected behavior near the optimal θ^*.

The time constraints of this project inhibited the completion of further experiments, which would have included natural gradient implementations with both Fisher approximations and with KFAC. Nonetheless, the utility and importance of second-order information, as well as the validity of the information geometrical perspective for policy distributions has been demonstrated in both the literature and experimentation. Future work in the field of RL with manifold optimization continues beyond the natural gradient with works such as Chen (2023) [9], which abandon the Fisher metric and instead iteratively learn a new metric that minimizes the Hessian trace and allows for a curvilinear ascent via the Levi-Civita connection on the policy parameter manifold. Improvements over the methods explored in this work will only progress as the fields of reinforcement learning and information geometry continue to collide.

Appendix A

Appendix

The experiments in this work were conducted on two programs written by the author: *ClassicControl* [github] built on Dr. Elizabeth Newman's GNFlow3 package for the SARSA experiments in Chapter 3.2, which was subsequently rewritten as the more general *GeomControl* [github] built on the newly-released TorchRL package for the policy-gradient experiments in Chapter 3.3.

ClassicControl: *Batch Multi-processed RL Runner for GNFlow3*

```
usage: Classic Control Experiment Scheduler [-h] [-m] [-v] [-t] [-z] [filename]

Performs experiments specified by an input CSV file with columns:

{job_name, problem, algorithm, featurizer, submethod, alpha, constant_step,
gamma, num_episodes, max_steps, max_eps, min_eps, order, seed, cvlen}

Jobs are performed sequentially by default, with an option for
multiprocessing for long or numerous experiments.

If -v (verbose) is not set, output of each job is stored in a file named
[DATE]_[TIME]_[problem]_[job_name]_[algorithm]_[submethod].[ext]
where [ext] is CSV (default) of TXT (enabled by -t flag)If -v is set, output is printed in th
e console (not compatible with multiprocessing).

-------Implemented-------
Problems:
   MountainCar
   CartPole
   Trajectory
Algorithms->Submethods:
   SemiGradientSARSAOrtho
      -> Original, Procrustes, CurvilinearInverse, CurvilinearNoInverse, QR
Step Control
    Armijo
Featurizers:
   Linear

--To be implemented:--

positional arguments:
  filename       filename/path of parameter CSV

optional arguments:
  -h, --help     show this help message and exit
  -m, --multi    enable multiprocessing
  -v, --verbose  print output to console, forces single-processing!
  -t, --txt      send output to TXT instead of CSV
  -z, --visual   visualize policy (only for Trajectory CSV mode)
```

GeomControl: *Batch Multi-processed RL Runner for TorchRL*

```
usage: GeomControl: Jafer's Deep RL Experiment Scheduler/Runner [-h] [-v] [-m] [filename]

Performs RL experiments in parallel specified by an input CSV file

positional arguments:
  filename       filename/path of parameter CSV

optional arguments:
  -h, --help     show this help message and exit
  -v, --verbose  print output to console
  -m, --multi    enable multiprocessing
```

Both programs ingest CSV files of jobs with specified parameters.

For example, an excerpt of one of the job files for the Section 3.2.3 experiments using *ClassicControl* looks like

job_name	problem	algorithm	featurizer	submethoc	alpha	constant_s	gamma	num_episc	max_steps	max_eps	min_eps	order	seed	cvlen
alpha1	Trajectory	SemiGradi	Linear	Original	5.00E-03	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha2	Trajectory	SemiGradi	Linear	Original	1.40E-02	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha3	Trajectory	SemiGradi	Linear	Original	2.45E-02	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha4	Trajectory	SemiGradi	Linear	Original	3.68E-02	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha5	Trajectory	SemiGradi	Linear	Original	5.16E-02	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha6	Trajectory	SemiGradi	Linear	Original	7.00E-01	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha7	Trajectory	SemiGradi	Linear	Original	9.39E-02	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha8	Trajectory	SemiGradi	Linear	Original	1.27E-01	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha9	Trajectory	SemiGradi	Linear	Original	1.79E-01	FALSE	0.9	400	9000	1	0.01	8	42	alpha
alpha10	Trajectory	SemiGradi	Linear	Original	2.00E-01	FALSE	0.9	400	9000	1	0.01	8	42	alpha

and the job file for the Section 3.3.6 experiments using *GeomControl* was

job_name	algorithm	loss	advant	env_name	frames	total_fran	test_inter	num_te	lrate	weight_d	anneal_l	gamma	mini_ba	loss_epi	gae_lan	clip_e	anneal_cl	critic_cc	entrop	loss_critic_type
IPp_FINALI	PPO_PENA	clip	GAE	InvertedPe	2000	200000	200000	5	3.00E-04	0	TRUE	0.99	100	10	0.95	0.2	FALSE	0.25	0	l2
IPc_FINALI	PPO_CLIP	clip	GAE	InvertedPe	2000	200000	200000	5	3.00E-04	0	TRUE	0.99	100	10	0.95	0.2	FALSE	0.25	0	l2
HCp_FINA	PPO_PENA	clip	GAE	HalfCheeta	2000	200000	200000	5	3.00E-04	0	TRUE	0.99	100	10	0.95	0.2	FALSE	0.25	0	l2
HCc_FINA	PPO_CLIP	clip	GAE	HalfCheeta	2000	200000	200000	5	3.00E-04	0	TRUE	0.99	100	10	0.95	0.2	FALSE	0.25	0	l2

Bibliography

[1] OpenAI Gym Documentation. https://www.gymlibrary.dev/index.html. Accessed: 2023-09-30.

[2] S. Amari and S.C. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, volume 2, pages 1213–1216 vol.2, 1998b. doi: 10.1109/ICASSP.1998.675489.

[3] S.-I. Amari, O. E. Barndorff-Nielsen, R. E. Kass, S. L. Lauritzen, and C. R. Rao. Differential Geometry in Statistical Inference. *Lecture Notes-Monograph Series*, 10:i–240, 1987. ISSN 07492170. URL http://www.jstor.org/stable/4355557.

[4] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 02 1998. ISSN 0899-7667. doi: 10.1162/089976698300017746. URL https://doi.org/10.1162/089976698300017746.

[5] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, 1998a. doi: 10.1162/089976698300017746.

[6] Shun-ichi Amari. Information Geometry of Neural Networks. volume E75-A, page 2, 08 2000. ISBN 978-3-540-67925-7. doi: 10.1007/3-540-44533-1_2.

[7] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[8] Lawrence Cayton. Algorithms for Manifold Learning. 2005. URL https://api.semanticscholar.org/CorpusID:408872.

[9] Gang Chen and Victoria Huang. Deep Metric Tensor Regularized Policy Gradient, 2023. URL https://arxiv.org/abs/2305.11017.

[10] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the Manifold Hypothesis. *J. Amer. Math. Soc.*, 29:983–1049, 2016. URL https://www.ams.org/journals/jams/2016-29-04/S0894-0347-2016-00852-4/.

[11] Brennan Gebotys, Alexander Wong, and David A. Clausi. Bag of Tricks for Natural Policy Gradient Reinforcement Learning. *CoRR*, abs/2201.09104, 2022. URL https://arxiv.org/abs/2201.09104.

[12] Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast Approximate Natural Gradient Descent in a Kronecker-factored Eigenbasis. *CoRR*, abs/1806.03884, 2018. URL http://arxiv.org/abs/1806.03884.

[13] Nessrine Hammami and Kim Khoa Nguyen. On-Policy vs. Off-Policy Deep Reinforcement Learning for Resource Allocation in Open Radio Access Network. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1461–1466, 2022. doi: 10.1109/WCNC51071.2022.9771605.

[14] Ronald A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

[15] Ronald A. Howard. Comments on the Origin and Application of Markov Decision Processes. *Operations Research*, 50(1):100–102, 2002. doi: 10.1287/opre.50.1.100.17788. URL https://doi.org/10.1287/opre.50.1.100.17788.

[16] Lei Huang, Xianglong Liu, Bo Lang, Adams Wei Yu, Yongliang Wang, and Bo Li. Orthogonal Weight Normalization: Solution to Optimization over Multiple Dependent Stiefel Manifolds in Deep Neural Networks, 2017. URL https://arxiv.org/abs/1709.06079.

[17] Robert Tibshirani Hui Zou, Trevor Hastie. Sparce Principle Component Analysis. *Journal of Computational and Graphical Statistics*, 15(2):271, 2006. doi: 10.1198/106186006X113430. URL https://hastie.su.domains/Papers/spc_jcgs.pdf.

[18] Shun ichi Amari. *Information Geometry and Its Applications*. Springer Tokyo, 2016.

[19] Michael Janner. Model-Based Reinforcement Learning: Theory and Practice. https://bair.berkeley.edu/blog/2019/12/12/mbpo/, 2019. Accessed 2024-01-16.

[20] Charles Jin and Martin Rinard. Manifold Regularization for Locally Stable Deep Neural Networks, 2020. URL https://arxiv.org/abs/2003.04286.

[21] Sham M Kakade. A natural policy gradient. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/4b86abe48d358ecf194c56c69108433e-Paper.pdf.

[22] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the Empirical Fisher Approximation. *CoRR*, abs/1905.12558, 2019. URL http://arxiv.org/abs/1905.12558.

[23] Taehoon Lee, Minsuk Choi, and Sungroh Yoon. Manifold Regularized Deep Neural Networks using Adversarial Examples. *CoRR*, abs/1511.06381, 2015. URL http://arxiv.org/abs/1511.06381.

[24] Yanli Liu, Kaiqing Zhang, Tamer Basar, and Wotao Yin. An Improved Analysis of (Variance-Reduced) Policy Gradient and Natural Policy Gradient Methods. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7624–7636. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/56577889b3c1cd083b6d7b32d32f99d5-Paper.pdf.

[25] James Martens. New Insights and Perspectives on the Natural Gradient Method. *Journal of Machine Learning Research*, 21(146):1–76, 2020. URL http://jmlr.org/papers/v21/17-678.html.

[26] James Martens and Roger B. Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *CoRR*, abs/1503.05671, 2015. URL http://arxiv.org/abs/1503.05671.

[27] Donald Michie. Experiments on the Mechanization of Game-Learning Part I. Characterization of the Model and its parameters. *The Computer Journal*, 6 (3):232–236, 11 1963. ISSN 0010-4620. doi: 10.1093/comjnl/6.3.232. URL https://doi.org/10.1093/comjnl/6.3.232.

[28] Gereon Quick. TMA 4192 Differential Topology: Lecture 16, Stiefel Manifolds and Grassmanians, March 2022. URL https://folk.ntnu.no/gereonq/TMA4192V2022/TMA4192_Lecture16.pdf.

[29] Andrew Saxe, James McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *Proceedings of the International Conference on Learning Represenatations 2014*, 2014.

[30] John Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, University of California, Berkeley, 2016.

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL `https://proceedings.mlr.press/v37/schulman15.html`.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

[33] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation, 2018.

[34] Peter H. Scönemann. A Generalized Solution of the Orthogonal Procrustes Problem. *Psychometrika*, 31(1):1–3, 1966. doi: 10.1007/BF02289451. URL `https://web.stanford.edu/class/cs273/refs/procrustes.pdf`.

[35] Satinder Singh, Tommi Jaakkola, Michael Littman, and Csaba Szepesvári. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, 38:287–308, 03 2000. doi: 10.1023/A:1007678930559.

[36] Jiaming Song and Yuhuai Wu. An Empirical Analysis of Proximal Policy Optimization with Kronecker-factored Natural Gradients. *CoRR*, abs/1801.05566, 2018. URL `http://arxiv.org/abs/1801.05566`.

[37] Richard S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988. doi: 10.1007/BF00115009. URL `http://incompleteideas.net/papers/sutton-88.pdf`.

[38] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Intro-*

duction. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[39] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[40] Csaba Szepesvári. *Algorithms for Reinforcement Learning*, volume 4. 01 2009. doi: 10.2200/S00268ED1V01Y201005AIM009.

[41] Hemant D. Tagare. Notes on Optimization on Steifel Manifolds. URL: https://noodle.med.yale.edu/hdtag/notes/steifel_notes.pdf, 2011.

[42] Christopher Watkins. *Learning from Delayed Rewards*. PhD thesis, Royal Holloway, University of London, 1989.

[43] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017. URL http://arxiv.org/abs/1708.05144.