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 now, 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.

Yu-Jan Ting April 6, 2022

Variance Reduction Methods for High-Dimensional Optimal Control Problems

by

Yu-Jan Ting

Lars Ruthotto
Adviser

Applied Mathematics & Statistics

Lars Ruthotto

Adviser

Yiran Wang

Committee Member

Jeremy Jacobson

Committee Member

2022

Variance Reduction Methods for High-Dimensional Optimal Control Problems

By

Yu-Jan Ting

Lars Ruthotto

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

Applied Mathematics & Statistics

2022

Abstract

Variance Reduction Methods for High-Dimensional Optimal Control Problems
By Yu-Jan Ting

We evaluate three variance reduction methods for solving high-dimensional optimal control problems: stochastic variance reduced gradient (SVRG), streaming stochastic variance reduced gradient (SSVRG), and stochastic recursive gradient (SARAH) algorithms. SVRG has recently been popular for finite-sum optimization; however, it is unclear whether SVRG can be applied to high-dimensional optimal control problems in real-time settings. We modify SVRG to solve the high-dimensional online optimization problem with an infinite dataset. SSVRG and SARAH are two variants of SVRG. While SSVRG is another streaming version of SVRG, SARAH leverages past stochastic gradient information to minimize variance. On several multi-agent collision avoiding problems, we fine-tune and compare the performance of SVRG, SSVRG, and SARAH with ADAM, a state-of-art algorithm. Our numerical experiments demonstrate the effectiveness of variance reduction methods for non-convex, high-dimensional optimal control problems. In particular, SARAH has the best performance in terms of convergence rate, sampling efficiency, and solution optimality. However, compared with SARAH, SVRG and SSVRG are less stable and relatively less adaptive to hyperparameter changes.

Variance Reduction Methods for High-Dimensional Optimal Control Problems

By

Yu-Jan Ting

Lars Ruthotto

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

Applied Mathematics & Statistics

2022

Acknowledgements

I would like to express my heartfelt gratitude to Dr. Lars Ruthotto, my adviser, for his invaluable support and intellectual guidance throughout the completion of this thesis. Dr. Ruthotto introduced me to the fascinating field of high-dimensional optimization and optimal control, which has greatly influenced my academic path. He has been an excellent supervisor, always providing me with insightful and constructive feedback and advice. His enthusiasm for research and impressive work ethic have inspired others to follow their ideas and dreams, encouraging me to expand my research interests.

I would also like to express my sincere gratitude to my committee members, Dr. Yiran Wang and Dr. Jeremy Jacobson, for their insightful suggestions and valuable feedback. Dr. Wang inspired me to become interested in mathematics, assisted me in developing a solid mathematical reasoning skillset, and taught me mathematical deduction and proof techniques. Dr. Jacobson encouraged me to explore fascinating research topics in machine learning while also revealing the importance of cloud computing, which was crucial in my model training process.

I sincerely thank my parents and friends who have supported my academic interest and self-motivation throughout this journey with love, kindness, encouragement, and understanding.

Contents

1 Introduction

1.1 Motivation 1

1.2 Contributions and Outline 2

2 High-Dimensional Optimization **4**

2.1 High-Dimensional Optimal Control 5

2.1.1 Overview 5

2.1.2 Optimal Control Problem 6

2.1.3 Multi-Agent Control Problem 7

2.1.4 Main Formulation 9

2.1.5 Numerical Implementation 10

2.2 Numerical Optimization Algorithms 11

2.2.1 Stochastic Gradient Descent (SGD) 11

2.2.2 Adaptive Learning Methods (ADAM) 11

2.2.3 Variance-Reduced Methods 12

2.3 Online Optimization 14

3 **Variance Reduction Methods** **15**

3.1 Variance Reduction Basics 15

3.2 Stochastic Variance Reduced Gradient (SVRG) 17

3.3 Streaming Stochastic Variance Reduced Gradient (SSVRG) 20

3.4 Stochastic Recursive Gradient Algorithm (SARAH) 21

3.5 Implementation 23

4 **Numerical Experiments** **24**

4.1 Experiment Set-Up 25

4.2 HyperParameter Optimization 27

4.3 Control Problems 33

4.3.1 Corridor Experiment 34

4.3.2 2-agent Swap Experiment 36

4.3.3 12-agent Swap Experiment 39

5 **Discussion** **42**

6 **Conclusion** **46**

List of Figures

4.1 Slice plot for 2-agent swap problem 32

4.2 Hyperparameter importance for 12-Agent swap problem 32

4.3 Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms in terms of the number of gradient counts and runtime on the corridor problem 34

4.4 Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms in terms of the number of gradient counts and runtime on the 2-agent swap problem 37

4.5 Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms regarding the number of gradient counts and runtime on the 12-agent swap problem 40

List of Tables

4.1 Best tuned hyperparameters for variance-reduced algorithms on different optimal control problems. 33

4.2 Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the corridor problem 35

4.3 Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the 2-agent swap problem . . 38

4.4 Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the 12-agent swap problem . 39

Chapter 1

Introduction

This chapter provides an overview of variance reduction methods and high-dimensional optimal control problems. We examine the significance of variance reduction methods by referring to previous studies where our study stems. After laying out our major contribution, we outline the organization of the thesis.

1.1 Motivation

This thesis is inspired and built upon the recent study [1], which formulates the neural network framework for solving high-dimensional optimal control (OC) problems. [1] investigates a series of multi-agent optimal control problems with state-space dimensions ranging from 4 to 150. The authors employ a grid-free numerical approach of parameterizing the value function with a neural network and a hybrid system of the Pontryagin Maximum Principle (PMP) and Hamilton-Jacobi-Bellman (HJB) methods to effectively mitigate the curse of dimensionality (CoD) for high-dimensional optimal control problems. Due to the high-dimensionality of this approach, an effective optimization algorithm that is both computationally and memory efficient is needed to facilitate the demanding training process for high-dimensional OC problems. The optimization problem in [1] is solved by the ADAM algorithm [2]. While this algorithm is computationally efficient, it can be relatively slow to converge and requires many samples. This thesis proposes to use variance reduction (VR) methods to train the same neural networks on high-dimensional optimal control problems, allowing for higher sampling efficiency with comparable or faster convergence rates than ADAM. We investigate the following variance reduction methods: stochastic variance reduced gradient (SVRG) [3], steaming stochastic variance reduced gradient (SSVRG) [4], and stochastic recursive gradient algorithm (SARAH) [5].

1.2 Contributions and Outline

We modify the SVRG algorithm, which is designed for finite-sum minimization problems, to solve the online optimization problem with an infinite dataset. We also extend SARAH to its streaming version by applying the recursive update rule to SSVRG. We implement three variance reduction optimizers (SVRG, SSVRG, and SARAH) in PyTorch and use them for training various high-dimensional OC problems given in [1]. We fine-tune hyperparameters for each variance reduction algorithm on every high-dimensional problem. Then, we conduct numerical experiments to compare the performance of our VR methods with ADAM, a state-of-the-art optimizer, in terms of the gradient-based computational cost and running time. Our numerical investigations demonstrate that variance reduction methods, in general, are effective in optimizing non-convex, high-dimensional multi-agent optimal control problems. We show that SARAH outperforms all other algorithms in sampling efficiency, convergence rate, and solution optimality. As the dimensionality of an optimization problem increases, the benefits of SARAH become more apparent. SVRG and SSVRG are also effective, but they are not as robust as SARAH, yielding more inconsistent results and requiring more hyperparameter tuning.

This thesis is outlined as follows. Chapter 2 discusses high-dimensional optimal control problems and gives the main mathematical formulation of our stochastic optimization problem. Then, we briefly summarize related optimization algorithms and introduce the variance reduction methods. Chapter 3 describes

SVRG, SSVRG, and SARAH algorithms in detail and analyzes their convergence rates. Chapter 4 conducts numerical experiments to optimize three specific high-dimensional OC problems using SVRG, SSVRG, and SARAH algorithms. We compare the performance of VR methods with the ADAM algorithm. Chapter 5 summarizes and interprets numerical results and discusses the limitations as well as future works. Finally, Chapter 6 draws our conclusions.

Chapter 2

High-Dimensional Optimization

Recently, high-dimensional optimization problems have received more attention due to their wide applications in real life. As a result, many numerical approaches have been developed for solving large-scale, high-dimensional optimal control problems. We first discuss the main mathematical formulation for our stochastic optimization problem with multi-agent collision avoiding systems. Then, we examine some recent optimization algorithms, which include stochastic gradient descent (SGD), adaptive moment estimation (ADAM), and variance reduction (VR) methods.

2.1 High-Dimensional Optimal Control

2.1.1 Overview

Due to its wide range of applications in real-world problems, the optimal control problem has been extensively researched in many fields, such as finance, engineering, physics, and mathematics. In this thesis, we are interested in non-convex, high-dimensional optimal control problems on an infinite dataset. While solving high-dimensional optimal control problems helps in decision-making for complicated dynamic systems, their computational complexity is significantly expensive due to the high dimensionality.

Recently, [1] proposes a grid-free numerical approach for solving high-dimensional optimal control problems. This approach combines the Pontryagin Maximum Principle (PMP) and Hamilton-Jacobi-Bellman (HJB) methods, parametrizing the value function with a neural network (NN). By incorporating the high-dimensional scalability from PMP and the global scope from HJB, this NN approach effectively reduces the curse of dimensionality (CoD) in the high-dimensional optimization problem. While this approach is effective, the training process of neural networks is costly in terms of computational cost and runtime. It also necessitates a significant number of samples to improve the generalizability of the approach. The need for large amounts of data exacerbates the difficulty of training. To improve training efficiency, we propose to use variance reduction methods for training neural networks on high-dimensional OC problems. We use the same optimization framework as [1], including the same problem formulation, data simu-

lation, and the identical NN model.

2.1.2 Optimal Control Problem

We first consider a deterministic optimal control problem over a fixed time horizon $[0, T]$. According to [1], the system's dynamics can be formulated as

$$\partial_s \mathbf{z}_{t,\mathbf{x}}(s) = g\left(s, \mathbf{z}_{t,\mathbf{x}}(s), \mathbf{u}_{t,\mathbf{x}}(s)\right), \quad \mathbf{z}_{t,\mathbf{x}}(t) = \mathbf{x} \quad \text{for} \quad t \leq s \leq T. \tag{2.1}$$

Here, the initial state of the system at time $t \in [0, T]$ is denoted as $\mathbf{x} \in \mathbb{R}^d$. $\mathbf{z}_{t,\mathbf{x}}(s) \in \mathbb{R}^d$ is the state of the system at time $s \in [t, T]$ with initial data $(t, \mathbf{x})$, and $\mathbf{u}_{t,\mathbf{x}}(s) \in U \subset \mathbb{R}^a$ is the control applied at time s. The function $g : [0, T] \times \mathbb{R}^d \times U \to \mathbb{R}^d$ models the change in state $\mathbf{z}_{t,\mathbf{x}} : [t, T] \to \mathbb{R}^d$ as a result of the control $\mathbf{u}_{t,\mathbf{x}} : [t, T] \to U$.

Assume that the control $\mathbf{u}_{t,\mathbf{x}} : [t, T] \to U$ and the trajectory $\mathbf{z}_{t,\mathbf{x}} : [t, T] \to \mathbb{R}^d$ satisfying (2.1) yield a cost $C\left(\mathbf{z}_{t,\mathbf{x}}, \mathbf{u}_{t,\mathbf{x}}\right)$, our goal is to find an optimal control $\mathbf{u}^*_{t,\mathbf{x}}$ that minimizes the cost $C\left(\mathbf{z}_{t,\mathbf{x}}, \mathbf{u}_{t,\mathbf{x}}\right)$, i.e, find $\mathbf{u}^*_{t,\mathbf{x}}$ such that

$$\Phi(t, \mathbf{x}) := C\left(\mathbf{z}^*_{t,\mathbf{x}}, \mathbf{u}^*_{t,\mathbf{x}}\right) = \inf_{\mathbf{u}_{t,\mathbf{x}}} C\left(\mathbf{z}_{t,\mathbf{x}}, \mathbf{u}_{t,\mathbf{x}}\right), \tag{2.2}$$

where Φ is the value function, and $\mathbf{u}^*_{t,\mathbf{x}}$ is an optimal control with the corresponding optimal trajectory $\mathbf{z}^*_{t,\mathbf{x}}$.

Additionally, the Hamiltonian of the system is defined by

$$\begin{aligned} H(t, \boldsymbol{z}, \boldsymbol{p}) &= \sup_{\boldsymbol{u} \in U} \{-\boldsymbol{p} \cdot g(t, \boldsymbol{z}, \boldsymbol{u}) - L(t, \boldsymbol{z}, \boldsymbol{u})\} \\ &= \sup_{\boldsymbol{u} \in U} \mathcal{H}(t, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{u}), \end{aligned} \tag{2.3}$$

where $\boldsymbol{p} \in \mathbb{R}^d$ is the adjoint state. The Hamiltonian is critical to the main formulation that will be discussed in 2.1.4.

2.1.3 Multi-Agent Control Problem

The multi-agent collision-avoidance problem is representative of high-dimensional OC problems whose dimensionality grows as the number of agents increases. According to [1], we extend the general mathematical OC framework described in 2.1.2 to multi-agent control problems. In a dynamic system of n agents, the initial joint-state of the system can be expressed as

$$\boldsymbol{x} = \left(x^{(1)}, x^{(2)}, \ldots, x^{(n)}\right) \in \mathbb{R}^d, \tag{2.4}$$

where $x^{(i)} \in \mathbb{R}^q$ is the initial state of the i th agent, and the dimension of the initial joint-state is $d = q \cdot n$. Similarly, the joint-state at time s of the system can be represented as

$$\boldsymbol{z}_{t,\boldsymbol{x}}(s) = \left(z^{(1)}_{t,\boldsymbol{x}}(s), z^{(2)}_{t,\boldsymbol{x}}(s), \ldots, z^{(n)}_{t,\boldsymbol{x}}(s)\right). \tag{2.5}$$

The control of the system can be denoted as

$$\boldsymbol{u}_{t,\boldsymbol{x}}(s) = \left(u^{(1)}_{t,\boldsymbol{x}}(s), u^{(2)}_{t,\boldsymbol{x}}(s), \ldots, u^{(n)}_{t,\boldsymbol{x}}(s)\right). \tag{2.6}$$

Given the system's dynamics (2.1), the objective function for multi-agent collision-avoidance problems is defined as

$$C\left(\mathbf{z}_{t,\mathbf{x}}, \mathbf{u}_{t,\mathbf{x}}\right) := \int_t^T L\left(s, \mathbf{z}_{t,\mathbf{x}}(s), \mathbf{u}_{t,\mathbf{x}}(s)\right) \mathrm{d}s + G\left(\mathbf{z}_{t,\mathbf{x}}(T)\right), \tag{2.7}$$

where $G : \mathbb{R}^d \to \mathbb{R}$ is the terminal cost, and $L : [0, T] \times \mathbb{R}^d \times U \to \mathbb{R}$ is the running cost or the Lagrangian. The terminal cost G measures the distance between the agents' final positions $\boldsymbol{z}_{t,\boldsymbol{x}}$ and their target states $\boldsymbol{y} \in \mathbb{R}^d$, i.e,

$$G\left(\boldsymbol{z}_{t,\boldsymbol{x}}(T)\right) = \frac{\alpha_1}{2} \left\|\boldsymbol{z}_{t,\boldsymbol{x}}(T) - \boldsymbol{y}\right\|^2. \tag{2.8}$$

The running cost L is defined as

$$L(s, \boldsymbol{z}, \boldsymbol{u}) = E(\boldsymbol{u}) + \alpha_2 Q(\boldsymbol{z}) + \alpha_3 W(\boldsymbol{z}). \tag{2.9}$$

Here, E is the energy term that measures how far the agents move along the trajectories. Q is the terrain function that models obstacles, such as smooth hills, by agents' spatial preferences. W models interactions among the individual agents to avoid collisions. $\alpha_1, \alpha_2, \alpha_3$ in (2.8) and (2.9) are scalar parameters inherent to different types of OC problems.

2.1.4 Main Formulation

Given the system dynamics defined in 2.1.2 and the multi-agent framework in 2.1.3, [1] blends the Pontryagin Maximum Principle (PMP) with Hamilton-Jacobi-Bellman (HJB) methods, parametrizing the value function with a Neural Network (NN), which gives the main formulation of our optimization problem. In this thesis, we study the following stochastic optimization problem.

$$\min_{\boldsymbol{w}} f(\boldsymbol{w}), \tag{2.10}$$

where

$$\begin{aligned} f(\boldsymbol{w}) \overset{\text{def}}{=} \mathbb{E}_{\boldsymbol{x}\sim\rho} \{ &\ell_{\boldsymbol{x}}(T) + G(\boldsymbol{z}_{0,\boldsymbol{x}}(T)) + \beta_1 c_{\text{HJt},\boldsymbol{x}}(T) \\ &+\beta_2 c_{\text{HJfin},\boldsymbol{x}} + \beta_3 c_{\text{HJgrad},\boldsymbol{x}} \} \end{aligned} \tag{2.11}$$

subject to

$$\partial_s \begin{pmatrix} \boldsymbol{z}_{0,\boldsymbol{x}}(s) \\ \ell_{\boldsymbol{x}}(s) \\ c_{\text{HJt},\boldsymbol{x}}(s) \end{pmatrix} = \begin{pmatrix} -\nabla_{\boldsymbol{p}} H\left(s, \boldsymbol{z}_{0,\boldsymbol{x}}(s), \nabla_{\boldsymbol{z}} \Phi\left(s, \boldsymbol{z}_{0,\boldsymbol{x}}(s); \boldsymbol{w}\right)\right) \\ L_{\boldsymbol{x}}(s) \\ P_{\text{HJt},\boldsymbol{x}}(s) \end{pmatrix}. \tag{2.12}$$

Our goal is to minimize the non-convex loss function $f(\boldsymbol{w})$ (2.11) subject to constraints (2.12) for initial states $\boldsymbol{x}$ independently sampled from the distribution with density ρ. Note that our stochastic optimization problem has an infinite dataset. Here, $\boldsymbol{w}$ is the NN parameter that holds the trainable weights used to approximate the value function Φ.

In the objective function (2.11), ℓ accumulates the Lagrangian cost, also called the running cost (L) given in (2.9), along the trajectories. Note that $\ell_{\boldsymbol{x}}(0) = 0$. G is the terminal cost defined in (2.8). Terms $c_{\mathrm{HJt},\boldsymbol{x}}, c_{\mathrm{HJfin},\boldsymbol{x}}, c_{\mathrm{HJgrad},\boldsymbol{x}}$, aim to penalize violations of the HJB equations. Note that $c_{\mathrm{HJt},\boldsymbol{x}}(0) = 0$ and $s \in [0, T]$. $\beta_1, \beta_2, \beta_3 > 0$ are scalar weights for penalty terms and unchanged for different OC problems. In constraints (2.12), H is the Hamiltonian of the system given in (2.3). $P_{\mathrm{HJt},\boldsymbol{x}}$ is also a penalty term that penalizes violations of the HJB equations, and it is accumulated along the trajectories.

2.1.5 Numerical Implementation

We use the same implementation framework as [1] to formulate our main stochastic optimization problem given in 2.1.4. We first discrete the optimization problem using a Runge-Kutta 4 integrator, which helps eliminate the constraints (2.12) and compute the objective function (2.11). Then, we employ the backpropagation (BP) to compute the gradients of the objective function (2.11) with respect to the weights of neural network parameters $\boldsymbol{w}$. In this thesis, we use variance reduction methods to update the NN parameters $\boldsymbol{w}$ and optimize the loss function. In Chapter 3, we go through the variance-reduced algorithms used in our numerical experiments in detail.

2.2 Numerical Optimization Algorithms

2.2.1 Stochastic Gradient Descent (SGD)

The classic gradient descent algorithms have been widely used in solving convex OC problems, and stochastic gradient descent (SGD) is one of the most prevailing methods. SGD iteratively acquires the gradient of the objective function at a randomly chosen data point and updates the parameters until converging. Compared with the gradient descent (GD) method, SGD has lower computational complexity and performs better in practice. However, SGD has a high variance due to the noise introduced by random sampling of individual data points. Therefore, the convergence of SDG is slow and unstable [6]. Moreover, SGD is likely to be trapped by saddle points in the objective function, thus failing to guarantee the global convergence of optimization [6]. The step decay in learning rate helps SGD to perform better in complicated OC problems. When we slowly decrease the learning rate of SGD, SGD may achieve the same convergence rate as the batch gradient descent.

2.2.2 Adaptive Learning Methods (ADAM)

ADAM [2], as one of the adaptive learning-method algorithms, is generally the state-of-the-art algorithm for solving high-dimensional optimization problems. ADAM leverages the exponential moving average of the gradient as well as the squared gradient to accelerate the convergence rate and adjust the learning rate

during the optimization [6]. Therefore, ADAM is an efficient approach with strong robustness and a satisfactory convergence rate. Our stochastic optimization problem, given by (2.11) and (2.12), is solved initially using ADAM in [1]. Although this approach is usually effective, it has several drawbacks when addressing our stochastic optimization problem. For example, ADAM requires too many samples to converge, particularly as the problem dimension grows. When problem dimension d is exceptionally high, we need more samples to capture the patterns behind the noisy non-convex objective function (2.11) to attain optimality. Moreover, to promote the model's generalizability and ensure the robustness of the NN approach, a large number of samples are required. Accordingly, finding a sampling-efficient algorithm is critical for training our stochastic optimization problem.

2.2.3 Variance-Reduced Methods

Variance-reduced methods have been developed to overcome certain limitations of SGD and ADAM when solving the high-dimensional optimal control problem. For example, SGD does not converge on its own with a constant stepsize and may fluctuate around the solution without converging to the optimum [7]. The core idea of variance reduction (VR) methods is to shift each gradient by some approximations of the stochastic gradient at the optimal point [7]. This way, VR methods update an estimate of the full gradient and overcome the not-converge nature of stochastic gradients, and naturally terminate when they reach the optimum. Furthermore, for adaptive learning-method algorithms such as ADAM, the

moving average of the gradient and the squared gradient give greater importance to recently sampled gradients. Hence, the weighted sum fails to converge to the full gradient [7]. In this regard, VR methods provide a more precise gradient estimation by using the plain average and are more likely to converge than ADAM [7].

The stochastic average gradient (SAG) [8] is one of the first variance reduction techniques that achieve linear convergence under smoothness, strong convexity conditions [7]. However, SAG has a hefty memory requirement because it keeps track of the gradient for every data, which is infeasible as state-space dimensionality grows large. SAGA [9] improves on SAG by eliminating a bias in each update and reduces the variance using covariates [7]. The stochastic variance-reduced gradient (SVRG) [3] is one of the most well-known and widely used variance reduction methods. We are interested in SVRG, its streaming variant SSVRG [4], and its recursive variant SARAH [5]. SVRG overcomes the high memory needs of SAG and SAGA while maintaining the same linear convergence rate on strongly convex and smooth objective functions [7]. As a result, SVRG and its variants are better suited for high-dimensional applications than other VR methods because of their lower memory requirements. While our learning problem generally is non-convex, we may relax the assumptions and generalize SVRG to problems that are not necessarily perfectly convex and smooth.

2.3 Online Optimization

So far, we have focused on finite-sum minimization problems. In other words, we have discussed the offline learning problem, which requires a finite dataset beforehand. Unlike offline optimization, the online optimization problem has an infinite dataset. Even though we observe some distribution information, such as the distribution density, we still do not have an expression for the distribution. In this thesis, we are more interested in online optimization, where we initialize the dynamic system with particular initial states and optimize the parameters based on streaming samples. The stochastic optimization problem given by (2.11) and (2.12) is an online problem, and we keep resampling data from the distribution. However, SAG and SAGA are unsuitable for online optimization since their high memory demand is impossible to fulfill with an infinite dataset. SVRG is adapted to its online variant streaming stochastic variance-reduced gradient (SSVRG), overcoming the difficulty of sampling from an infinite dataset. SSVRG is an algorithm for handling streaming data arrivals and processing them in a single pass-through. The Monte Carlo method is used to estimate the control state more accurately with each pass of the sampling data, as we keep drawing samples from the distribution. Therefore, we want more sampling-efficient algorithms to give better performance in optimization.

Chapter 3

Variance Reduction Methods

This chapter goes through the fundamental ideas of variance-reduced optimization algorithms. Then, we discuss SVRG, SSVRG, and SARAH algorithms in detail. We explain how they work, their update procedures, and provide a code implementation framework for these VR techniques. In the next chapter, we compare the performance of these optimizers with our baseline model ADAM in solving three different high-dimensional optimal control problems.

3.1 Variance Reduction Basics

As stated in 2.2, SGD is difficult to converge with a constant stepsize because stochastic gradients, either single gradients or batch gradients, will not converge towards zero as the full gradient (GD) method does when reaching the optimum. In other words, variance inherent in full gradient estimation eventually prevents

SGD from further converging. Variance reduction (VR) methods seek to eliminate the noise and make the convergence faster theoretically. Instead of using stochastic gradients to approximate the full gradient as SGD does directly, VR methods collect information from stochastic gradients and use it to update the full gradient estimate [7]. Consider the objective function f given in (2.11) for our stochastic optimization problem, we denote the full gradient as $\nabla f\left(\boldsymbol{w}_k\right)$, the stochastic gradient on data point i at parameter $\boldsymbol{w}_k$ as $\nabla f_i\left(\boldsymbol{w}_k\right)$, and the full gradient estimate as $g_k \in \mathbb{R}^d$. Here, k represents the kth update of the parameter $\boldsymbol{w}$, and i stands for the ith data point used for computing the gradient. Then, in $k+1$th iteration, parameters $\boldsymbol{w}$ can be updated with learning rate η as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k - \eta g_k, \tag{3.1}$$

The full gradient estimate g_k is derived from shifting the stochastic gradients at the kth stage $f_i\left(\boldsymbol{w}_k\right)$ by the stochastic gradient at the terminal stage $\nabla f_i\left(\boldsymbol{w}_\star\right)$ when reaching the optimum point $\boldsymbol{w}_\star$ [7], i.e,

$$g_k = f_i\left(\boldsymbol{w}_k\right) - \nabla f_i\left(\boldsymbol{w}_\star\right). \tag{3.2}$$

We can think of the shifted term $\nabla f_i\left(\boldsymbol{w}_\star\right)$ as a bias correction term that reduces the bias in the stochastic gradient $f_i\left(\boldsymbol{w}_k\right)$ in each iteration, allowing g_k to converge to an unbiased estimate of the full gradient $\nabla f\left(\boldsymbol{w}_k\right)$ [10]. However, it is unlikely that we know $\nabla f_i\left(\boldsymbol{w}_\star\right)$ beforehand. Instead of computing the true $\nabla f_i\left(\boldsymbol{w}_\star\right)$, VR methods approximate it using some gradient aggregation tech-

niques [10, 7]. These techniques include incorporating the weighted past gradient information into the current gradient to update the full gradient estimate [10]. This is the core idea behind VR methods, and different VR methods approximate $\nabla f_i(\boldsymbol{w}_\star)$ (and therefore g_k) differently. We will see how this works in the next section by approximating the bias correction term $\nabla f_i(\boldsymbol{w}_\star)$ and full gradient estimate g_k in SVRG, SSVRG, and SARAH.

3.2 Stochastic Variance Reduced Gradient (SVRG)

The stochastic variance reduced gradient (SVRG) algorithm [3], as outlined in **Algorithm 1**, employs a nested loop to approximate the bias correction term $\nabla f_i(\boldsymbol{w}_\star)$, which is defined in (3.2). The original SVRG only works for offline learning problems with a finite dataset. We modify this method in this thesis to solve the online high-dimensional optimal control problem over an infinite dataset.

For each outer loop, representing stage s, the original SVRG [3] computes the full gradient for the optimization parameters $\boldsymbol{w}$ over the entire dataset. However, since our stochastic optimization problem has an infinite dataset, we modify SVRG in **Algorithm 1** to compute the average gradient for a batch $\tilde{B} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{bs}\}$ drawn from the distribution ρ, i.e, $\nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)$. The initial point $\boldsymbol{w}_0$ for the inner loop is initialized as $\tilde{\boldsymbol{w}}_s$. In each inner loop iteration, representing stage t, SVRG samples a mini-batch $B = \{x_1, x_2, \ldots, x_{bs_1}\}$ from the batch $\tilde{B}$ drawn from the outer loop. Then, we have the following update rule

Algorithm 1: Stochastic Variance Reduced Gradient (SVRG) [3]

Input Initial point $\tilde{\boldsymbol{w}}_0$, outer-loop batch sizes bs, update frequency m, learning rate η, inner-loop batch size bs_1

for *each stage* $s = 0, 1, 2 \ldots$ **do**
  Sample $\tilde{B} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{bs}\}$ of size bs from ρ, then compute and store
  $$\nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)$$
  $\boldsymbol{w}_0 \leftarrow \tilde{\boldsymbol{w}}_s$
  for $t = 0, 1, 2 \ldots, m-1$ **do**
    Sample $B = \{x_1, x_2, \ldots, x_{bs_1}\}$ of size bs_1 from $\tilde{B}$ and set
    $g_t \leftarrow \nabla f_B(\boldsymbol{w}_t) - (\nabla f_B(\tilde{\boldsymbol{w}}_s) - \nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s))$
    $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta g_t$
  end
  $\tilde{\boldsymbol{w}}_{s+1} \leftarrow \boldsymbol{w}_m$
end

$$g_t \leftarrow \nabla f_B(\boldsymbol{w}_t) - (\nabla f_B(\tilde{\boldsymbol{w}}_s) - \nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)) . \tag{3.3}$$

Here, g_t is the gradient estimate given in (3.2), and the bias approximation term in this case is

$$\nabla f_i(\boldsymbol{w}_\star) \approx \nabla f_B(\tilde{\boldsymbol{w}}_s) - \nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s) . \tag{3.4}$$

In (3.3), the first term, $\nabla f_B(\boldsymbol{w}_t)$, is the stochastic gradient on batch B at $\boldsymbol{w}$ in inner-loop stage t; the second term, $\nabla f_B(\tilde{\boldsymbol{w}}_s)$, is the stochastic gradient on the same batch B at $\tilde{\boldsymbol{w}}$ in outer-loop stage s; and the third term, $\nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)$, is the average gradient computed in the outer-loop stage s. $\nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)$ is indeed the expectation of $\nabla f_B(\tilde{\boldsymbol{w}}_s)$, and $\nabla f_B(\tilde{\boldsymbol{w}}_s) - \nabla f_{\tilde{B}}(\tilde{\boldsymbol{w}}_s)$ is the bias of $\nabla f_B(\tilde{\boldsymbol{w}}_s)$. Therefore,

we may use this bias stated in (3.4) to approximate the bias of $\nabla f_B(\boldsymbol{w}_t)$. Accordingly, the variance in the gradient estimate g_t is reduced every time $\nabla f_B(\boldsymbol{w}_t)$ is corrected by the past gradient information and the approximated bias [10]. Then, the neural network parameter $\boldsymbol{w}$ is updated as

$$\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta g_t. \tag{3.5}$$

As the inner loop terminates, parameters tracked in the outer loop $\tilde{\boldsymbol{w}}_{s+1}$ are assigned by the value of $\boldsymbol{w}_m$, and the outer loop continues. The process is repeated until the loss function no longer decreases.

Convergence

With a finite dataset, SVRG converges linearly under smoothness and strong convexity assumptions for the objective function [3]. When compared to SGD, which has a sub-linear convergence rate with a decaying learning rate, VR methods converge faster under the same assumptions because the learning rate may stay rather large without decaying as the inherent variance is reduced during the optimization [3].

The SVRG method can solve non-convex problems and is a broad technique for high-dimensional optimization problems with a complex (usually non-convex) loss function. SVRG can be applied to non-convex problems immediately after the SDG method runs certain epochs and approaches to the optimum. As a result, SVRG can accelerate the convergence obtained by SGD [3].

Furthermore, in non-convex settings, the convergence of SVRG is demonstrated to be faster than SGD [11]. [11] also emphasizes the effectiveness of mini-batching in non-convex settings. Mini-batching improves the performance of SVRG in minimizing non-convex loss functions while also increasing the parallelism rate to speed up the training time.

3.3 Streaming Stochastic Variance Reduced Gradient (SSVRG)

The streaming stochastic variance reduced gradient (SSVRG) algorithm [4], described in **Algorithm 2**, is a streaming variant of SVRG that addresses the online optimization problem with an infinite dataset. The intuition and broad algorithmic framework of SSVRG are similar to those of SVRG. However, two major differences are occurring within the inner loop:

1). Instead of sampling from $\tilde{B}$, SSVRG samples directly from ρ in each inner loop.

2). Rather than using the fixed update frequency m, SSVRG begins the inner loop by uniformly selecting the batch size $\tilde{m}$ from $\{1, 2, \ldots, m\}$.

Convergence

SSVRG may produce linear or super-polynomial convergence results, depending on assumptions for strong convexity, smoothness, and self-concordance. How-

Algorithm 2: Streaming Stochastic Variance Reduced Gradient (SSVRG) [4]

Input Initial point $\tilde{\boldsymbol{w}}_0$, outer-loop batch size bs, update frequency m, learning rate η, inner-loop batch size bs_1

for *each stage* $s = 0, 1, 2 \ldots$ **do**

  Sample $\tilde{B} = \{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_{bs}\}$ of size bs from ρ and compute the estimate

$$\nabla f_{\tilde{B}}\left(\tilde{\boldsymbol{w}}_s\right)$$

  $\boldsymbol{w}_0 \leftarrow \tilde{\boldsymbol{w}}_s$

  Sample $\tilde{m}$ uniformly at random from $\{1, 2, \ldots, m\}$.

  for $t = 0, 1, 2 \ldots, \tilde{m} - 1$ **do**

    Sample $B = \{x_1, x_2, \ldots, x_{bs_1}\}$ of size bs_1 from ρ and set

    $g_t \leftarrow \nabla f_B(\boldsymbol{w}_t) - (\nabla f_B(\tilde{\boldsymbol{w}}_s) - \nabla f_{\tilde{B}}\left(\tilde{\boldsymbol{w}}_s\right))$

    $\boldsymbol{w}_{t+1} \leftarrow \boldsymbol{w}_t - \eta g_t$

  end

  $\tilde{\boldsymbol{w}}_{s+1} \leftarrow \boldsymbol{w}_{\tilde{m}}$

end

ever, these convergence results are not guaranteed in our completely non-convex optimization scenario. Similar strategies in Section 3.2 that have been adapted for SVRG to alleviate the non-convexity may also be beneficial to SSVRG in order to minimize the non-convex loss function. Similar to SVRG, SSVRG algorithm is efficient in storage with space complexity linear to the sample size [4].

3.4 Stochastic Recursive Gradient Algorithm (SARAH)

More recently, [5] proposes a variant of SVRG, the stochastic recursive gradient algorithm (SARAH), to address the non-convex optimization problem effectively by using the recursive update of the stochastic gradient estimate. The update rule

for SVRG given in (3.3) is modified as the following recursive version [5]

$$g_t \leftarrow \nabla f_B(\boldsymbol{w}_t) - \nabla f_B(\tilde{\boldsymbol{w}}_s) + g_{t-1}. \tag{3.6}$$

Here, the gradient estimate g_t is updated recursively with its previous value g_{t-1} instead of the fixed average gradient $g_0 = \nabla f_{\tilde{B}}\left(\tilde{\boldsymbol{w}}_s\right)$.

The SARAH algorithm proposed by [5] is used for solving the offline finite-sum minimization problems. In this thesis, we are interested in the streaming version of SARAH to solve problems with an infinite dataset. Therefore, we modify SSVRG by replacing its update rule with the recursive update rule presented in (3.6). We use the resulting streaming SARAH algorithm to solve the high-dimensional optimal control problems over an infinite dataset.

Convergence

Similar to SVRG and SSVRG, SARAH achieves a linear convergence rate under the strong convexity assumption [5]. Additionally, SARAH converges linearly for inner loops under strong convexity, while SVRG only converges linearly for outer loops [5]. In other words, SARAH generates more stable and robust convergence patterns during the training. For non-convex problems, SARAH attains a sublunar convergence rate [5]. The streaming variant of SARAH, which we use in our numerical experiments for solving optimal control problems with an infinite dataset, exhibits similar convergence properties.

3.5 Implementation

We implement the SVRG, SSVRG, and SARAH optimizers in PyTorch. PyTorch provides us *torch.optim.Optimizer*, which handles all general optimization machinery and is the base class for all optimizers in PyTorch. To implement these optimizers, we inherit and modify two methods *__init__()* and *step()* in *torch.optim.Optimizer*. In addition to default parameters given by the base class, we add extra hyperparameter fields called *mean* and *new_param_groups* in *__init__()*. We add hidden functions *get_param_groups()*, *get_mean()*, *set_mean()*, *set_grad()*, and *set_mean_grad()* to calculate the gradient terms $\nabla f_B(w_t)$, $\nabla f_B(\tilde{w}_s)$, and $\nabla f_{\tilde{B}}(\tilde{w}_s)$ for update rules given in (3.3) and (3.6). We override the *step()* method to perform the parameter update once we calculate all required gradient terms. For higher efficiency, we optimize VR optimizers by using Pytorch Tensor operations to do gradient-based computations.

For each VR algorithm, we write the driver functions *train()* and *validate()* to fulfill the training and validation procedures. The implementation of the optimizers follows the standards of PyTorch and includes basic exception handling.

Chapter 4

Numerical Experiments

This section demonstrates the potential of variance-reduced algorithms (SVRG, SSVRG, and SARAH) for training the two-layer residual neural network (ResNet). We prepare several high-dimensional optimal control problems to compare the effectiveness of VR methods and the baseline algorithm ADAM in terms of the sampling efficiency and convergence rate. These high-dimensional multi-agent collision-avoiding systems include the corridor problem, 2-agent swap problem, and 12-agent swap problem.

Experiments show that VR methods are efficient when solving non-convex, high-dimensional optimal control problems, with SARAH having the best performance regarding sampling efficiency, solution optimality, and runtime.

4.1 Experiment Set-Up

Data

Our input data are initial points drawn from the distribution with density ρ. For each problem, we first initialize the initial joint-state $\boldsymbol{x}_0$ and the respective target joint-state $\boldsymbol{y}$ with pre-defined values. Then, we form the training set $\boldsymbol{X}$ by randomly sampling initial points from the Gaussian distribution, which is centered at $\boldsymbol{x}_0$ with an identity covariance [1]. This training set $\boldsymbol{X}$ helps to promote the model generalizability. We independently sample one batch of initial states from the distribution and train the NN on that batch for each outer or inner loop iteration. We resample a new batch for the next iteration and continue the process described in algorithms given in Chapter 3.

Evaluation Metrics

We use the validated loss value $f(\boldsymbol{w})$ given in (2.11) as an evaluation criterion for NN training. We assume the training converges when the change over the validation loss value is less than a pre-defined threshold δ. Besides the validated loss, we also look at the validated L value, running cost over the trajectory, and the validated G value, terminal cost that measures how far the final state is from the goal. Analyzing the L and G helps us validate the benefits of VR methods more directly. A lower L value indicates a shorter trajectory with fewer collisions to the other agents or the hills. A lower G value indicates that the final state is

closer to the goal, leading to a shorter trajectory.

We measure the validated loss, L, and G in terms of the number of gradient counts and runtime for VR methods (SVRG, SSVRG, and SARAH) and the baseline algorithm ADAM. Typically, the number of epochs or iterations is frequently used to assess the effectiveness of optimization algorithms. However, since we are dealing with the online learning problem, the notion of the epoch is infeasible in our case. Moreover, because we are using several mini-batching strategies within complicated nested loops, the number of iterations would not be comparable across different algorithms. Alternatively, gradient counts are accumulated during the training, and thus the number of gradient counts can be considered a fair indicator to measure the effectiveness. The number of gradient counts reflects the number of times the gradient is computed as parameter updating proceeds. More importantly, gradient computation may be the most intensive operation during optimization [3]. Therefore, the number of gradient counts is a meaningful indicator of the computational cost, unaffected by the training environment and other system variables. The running time, also called runtime, measures the time elapsed from the beginning to the current training time. It is the most immediate measure of how long a method takes to converge.

Training Environment

We perform the experiment on a Macbook running MacOS 10.15.4 with a 2.3 GHz CPU and 16 GB of RAM for training and validation. In addition, to facilitate and speed up the training process, we fine-tune some hyperparameters in Colab.

Logging and Visualization

To tune the hyperparameters and store the configuration as well as training results, we use Python package **Wandb**, which is an online dashboard that keeps track and visualizes all results. **Wandb** provides an easy way to understand our data distribution, save the model configuration, and visualize training performance. We generate reports on experiments and plot interactive graphs using specific trails to analyze training performance throughout the experiments.

4.2 HyperParameter Optimization

Optuna

We use the **Optuna** library [12] to fine-tune hyperparameters. The **Optuna** library [12] is a powerful optimization toolkit followed the *define-by-run* principle for Bayesian optimization and black-box optimization. It can handle high-dimensional problems and has rich features, such as automatic model selection and parallel search. **Optuna** offers a state-of-the-art hyperparameter sampling method and efficiently prunes unpromising trials. We employ it to locate the feasible range of optimal hyperparameters within which we fine-tune the hyperparameters manually.

Tuning Methods

There are two stages in the tuning process:

Phase 1: Locate the feasible range of hyperparameters for the model. We employ the **Optuna** package to explore the parameter space automatically. Each trial runs for about 15 minutes. The time duration of 15 minutes may not be enough for complete training, but it is more than enough to observe the performance of the optimization algorithms. For example, for the corridor and 2-agent swap problems, the loss value drops dramatically in the first 200 seconds, while the loss value drops dramatically in the first 500 seconds for a 12-agent swap problem. The remaining duration is enough for observing the convergence trends.

Phase 2 We manually fine-tune hyperparameters within the range found in phase 1 to get the most optimal hyperparameters. In this stage, we run each trial for a longer duration until the model converges.

HyperParameters

Our experiments only consider and tune hyperparameters associated with our optimization algorithms (SVRG, SSVRG, and SARAH). Hyperparameters inherent to the NN model are not tuned in our case. Instead, we use default settings for these hyperparameters given in [1]. The following are the hyperparameters we consider for tuning:

Learning rate (lr): We tune the learning rate (denoted as η in **Algorithm 1, 2**) because it can significantly affect the convergence of optimization algorithms. A too-large learning rate might cause the algorithm to diverge, while a too-small learning rate will slow the training process. We search learning rate in the range of $[1\text{e}{-1}, 1\text{e}{-2}, \ldots, 1\text{e}{-7}]$.

Batch size: We also tune the batch size because it can affect both the training time and the accuracy of models. Larger batch size will lead to faster training but might not lead to better results, while a smaller batch size will take more time to train but might lead to better results. We will tune two batch size parameters:

- $batch_size$: Outer-loop batch size (denoted as bs in **Algorithm 1, 2**),
- $mini_bs$: Inner-loop batch size (denoted as bs_1 in **Algorithm 1, 2**).

Both parameters are sampled from the range of $[2^1, 2^2, \ldots, 2^{11}, 2^{12}]$. We normally choose values of batch size in factor of 2 due to the efficiency of matrix-matrix multiplications and parallelisms [13].

Update frequency (m_1): We also tune the update frequency, which determines how often the model parameters are updated during training. A higher value for m will cause the model to be updated more frequently, but this might not always lead to better results. Note that this hyperparameter is only used by SSVRG and SARAH algorithms but not SVRG. We search update frequency m_1 in the range of $[2^1, 2^2, \ldots, 2^{11}, 2^{12}]$.

Learning rate decreasing frequency (lr_freq): Another hyperparameter we consider is the learning rate decreasing frequency. This parameter determines how often the learning rate is reduced during training. Reducing the learning rate can help the model converge faster and lead to better results. We search lr_freq uniformly in the range of $[30, 200]$.

Tuning Process

In phase 1, we use package **Optuna** to perform the automatic tuning, and we create a database to store the tuning results. Each row in our database corresponds to one trial, consisting of hyperparameters and corresponding performance metrics such as loss value and training time. To ensure the efficiency of the searching, we set a pruner to early stop the unnecessary trials. We apply the following stopping rules:

- If the current trial has run for more than 15 minutes, we stop the training. We choose this value because it is long enough to observe the performance trends of algorithms in terms of loss value and training time.
- If the trial's best intermediate result is worse than the median of intermediate results of previous trials at the same step, we prune the trial. We disabled the pruning until the first 5 trials are finished.
- If the validation loss value becomes nan, we stop the trial.

After long trial-and-error training, we retrieve the searching results from the database to prepare for fine-tuning in Phase 2. We first draw the slice plot **Figure 4.1**, which shows the correlation between the loss value and different hyperparameters. Based on the plot, we have several observations.

First, most low loss values appear around the $batch_size$ of 128 or 512. [13] suggests that the utility curve of choosing the batch size is a U shape. A trade-off of choosing batch sizes exists: a larger batch size means we can take advantage of

efficient matrix multiplication or parallelism to make computation faster, while a smaller batch size means more numbers of updating per computation. Therefore, we decide to manually search the optimal batch size in the range of $[128, 512]$ to maximize the utilization.

Second, we can see that lr_freq is not a deterministic hyperparameter that affects the loss value, while lr is relatively deterministic. A good guess about learning rate and frequency is 0.001 and 180, respectively. A good strategy proposed by [13] is that if the initial learning rate diverges, we should try with a new learning rate three times smaller than the old learning rate until no divergence is observed.

Third, the values of m_1 are scattered around 128, 512, and 1024. Note that m_1 determines the number of iterations; thus, the parameters will be more accurate if the parameters have been updated more frequently with larger m_1. However, each iteration comes at the cost of increasing gradient computation. Therefore we start with a smaller value of m_1 (e.g., 128) and gradually increase m_1 if the models do not converge.

We also run ANOVA tests to examine the importance of hyperparameters on loss value (Fig 4.2a) and the training duration (Fig 4.2b). First, the right choice of lr produces good loss values because lr controls how much we direct along the gradients in each step. The second important is m_1, which is intrinsically deterministic. Thus, m_1 is the hyperparameter we need to tune carefully. Finally, it can be seen from **Figure 4.2 (b)** that $mini_bs$ is the most crucial factor determining the duration. The larger $mini_bs$, the more iterations the algorithm takes.

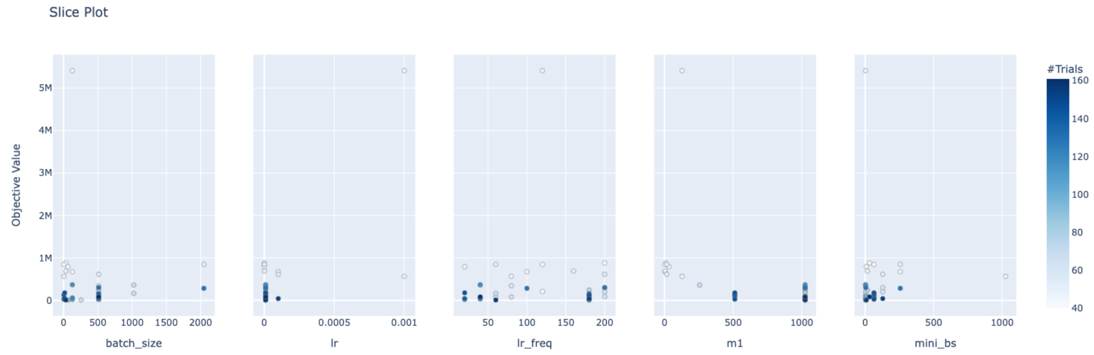
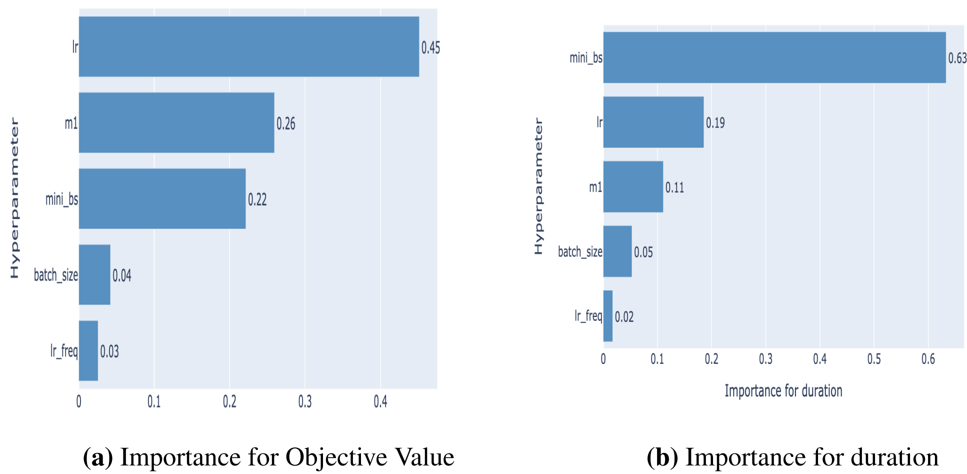


Figure 4.1: Correlations between hyperparameters and validated loss values for the 2-agent swap problem. We can see that lr_freq and $batch_size$ are less deterministic to the loss value, while lr, m_1, and $mini_bs$ are relatively deterministic.



(a) Importance for Objective Value **(b)** Importance for duration

Figure 4.2: Hyperparameter importance by running ANOVA tests between hyperparameters and loss value (left) or runtime (right) for the 12-agent swap problem.

Tuning Results

The best sets of hyperparameters for each VR algorithm on each optimal control problem are selected based on the lowest validation loss. The results are summa-

rized in **Table 4.1**.

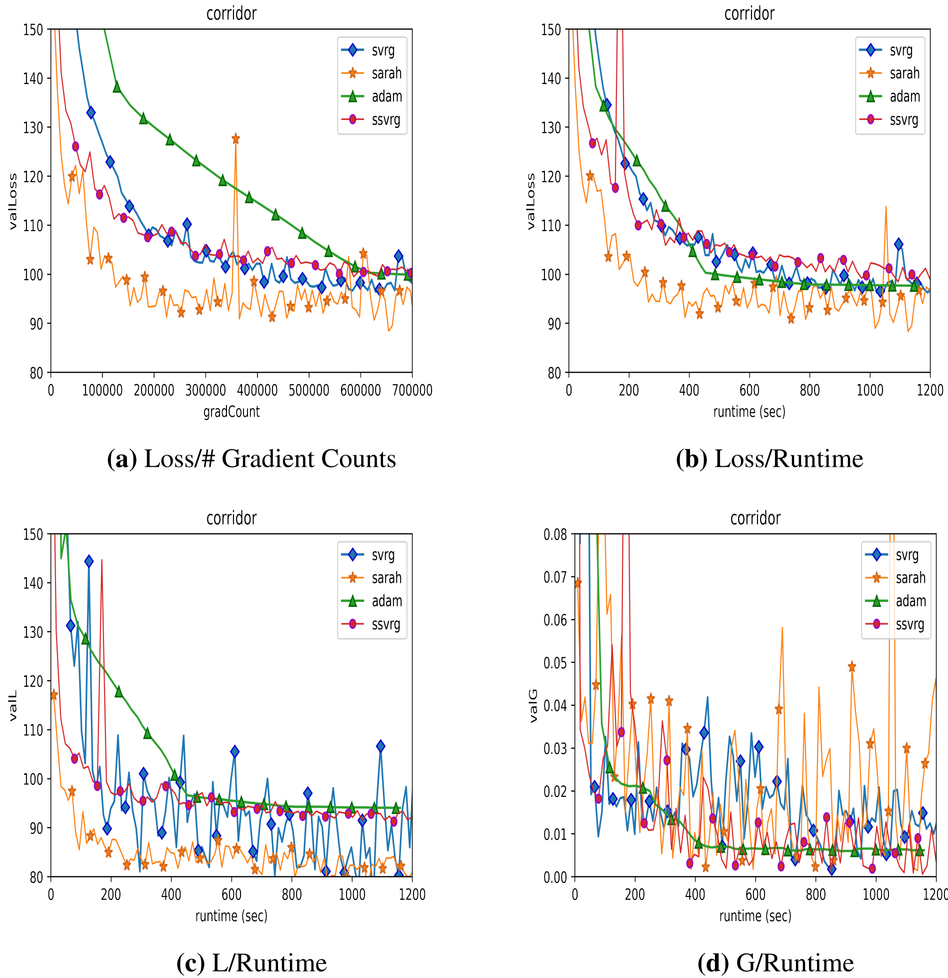
	lr	batch_size	mini_bs	m1	lr_freq
		Corridor			
SVRG	0.001	1024	32	-	45
SSVRG	0.001	128	32	64	70
SARAH	0.01	128	32	128	80
		2-Agent Swap			
SVRG	0.001	128	32	-	180
SSVRG	0.001	128	32	128	180
SARAH	0.001	128	32	128	180
		12-Agent Swap			
SVRG	0.00001	256	4	-	180
SSVRG	0.001	256	32	128	180
SARAH	0.00001	256	4	1024	180

Table 4.1: Best tuned hyperparameters for variance-reduced algorithms on different optimal control problems.

4.3 Control Problems

We compare numerical results of SVRG, SSVRG, and SARAH with the state-of-the-art adaptive method ADAM on three optimal control problems by employing the optimized hyperparameters summarized in **Table 4.1**. ADAM is trained with the default hyperparameters given in [1]. We measure the validated loss value ($valLoss$), the validated running cost ($valL$), and the validated terminal cost($valG$) in terms of the number of gradient counts and the runtime for each method on each problem. The following is a description of each scenario and the corresponding numerical findings.

4.3.1 Corridor Experiment



(a) Loss/# Gradient Counts

(b) Loss/Runtime

(c) L/Runtime

(d) G/Runtime

Figure 4.3: Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms in terms of the number of gradient counts and runtime on the corridor problem. It can be seen that SARAH outperforms all other methods regarding the number of gradients counts and runtime, resulting in a more optimal loss. VR methods are generally more sampling efficient than ADAM, although ADAM is more stable.

The corridor experiment was designed to be a $d = 4$ dimensional optimal control problem. In this experiment, two agents attempt to cross the narrow corridor

	# Gradient Counts			Runtime (min)		
Optimizers	200,000	500,000	600,000	5.75	7.38	13.59
ADAM	129.61	106.53	100.31*	112.22	100.31	100.31*
SVRG	107.76	98.20*	99.23	108.59	98.20*	98.20
SSVRG	107.07	101.87*	102.12	92.37	101.87*	101.21
SARAH	92.37*	92.56	98.76	92.37*	99.72	97.14

Table 4.2: Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the corridor problem. * denotes the optimal loss value obtained when the algorithm starts to converge at the specific # of gradient counts/runtime.

between two hills to their predetermined destination with the shortest paths while avoiding collisions. The corridor between the hills is kept as narrow as feasible to allow just one agent to pass through. An agent must wait before the other agent passes through the corridor. We use the same problem formulated in [1], with more details available in [1].

Figure 4.3 shows the numerical results of validated loss over the number of gradient counts (a) and validated loss (b), validated L (c) as well as validated G (d) over runtime, respectively. **Table 4.2** summarizes the specific validated loss values at different levels of gradient counts (left) and runtime (right), where specific algorithm starts to converge.

As we can see from **Figure 4.3 (a)**, SVRG and SSVRG algorithms perform similarly on this problem in terms of gradient counts. Both converge faster than ADAM while using the same number of gradient counts. SARAH algorithm outperforms ADAM, SVRG, and SSVRG algorithms in terms of gradient counts. Accordingly, we found that VR methods are more sampling efficient, requiring fewer

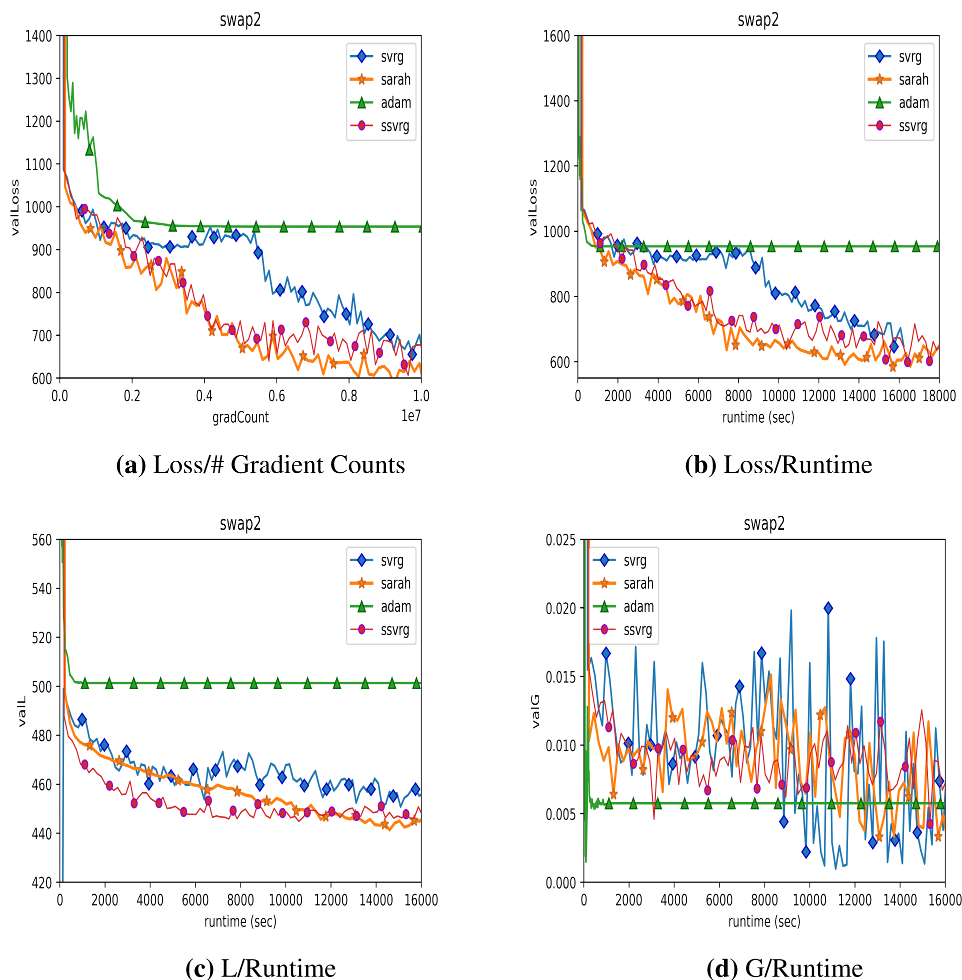
samples than ADAM. SARAH is the most sampling efficient among the three VR methods as SARAH requires only $200,000$ gradient counts to converge, which is 67% less than ADAM (Table 4.2). We can also observe that SVRG and SSVRG have comparable runtime durations, whereas ADAM outperforms them by converging faster in terms of runtime (Fig 4.3b). The quickest-running algorithm is SARAH, which spends 5.75 minutes to converge (Table 4.2). More importantly, SARAH finds slightly more optimal solutions, achieving the lowest loss value of 92.37, which is 7.9% more optimal than ADAM (Table 4.2). Based on **Figures 4.3** (c) and (d), we discovered that ADAM has lower G values, whereas SARAH has lower L values. This implies that SARAH is better at reducing running costs, whereas ADAM decreases terminal expenses.

In summary, for corridor problems, VR methods are more sampling efficient than ADAM with comparable runtime. Still, SARAH outperforms all other methods in terms of gradient counts and runtime to reach more optimal loss values. However, ADAM is the most stable algorithm, while VR methods oscillate more frequently.

4.3.2 2-agent Swap Experiment

Inspired by the corridor experiment, we conduct a 2-agent swap experiment. Instead of directly passing through the narrow corridor between two hills, the two agents begin at opposite ends of the corridor and aim to exchange their positions. Details for the problem formulation can be found in [1].

Numerical results are summarized in **Figure 4.4** and **Table 4.3**. **Figure 4.4**



(a) Loss/# Gradient Counts

(b) Loss/Runtime

(c) L/Runtime

(d) G/Runtime

Figure 4.4: Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms in terms of the number of gradient counts and runtime on the 2-agent swap problem. VR methods converge to lower loss values than ADAM. Moreover, VR methods are more sampling efficient, but they take longer runtime to converge. SARAH is the most sampling efficient and achieves the lowest loss value.

(a) shows that VR methods outperform ADAM in terms of the gradient counts because all VR methods achieve lower loss values than ADAM when using the same number of gradient counts. For example, when ADAM converges to the

	# Gradient Counts				Runtime (min)			
Optimizers	2,000k	8,000k	9,000k	10,000k	7.35	205.4	240.9	266.31
ADAM	967.21*	953.66	953.66	953.66	967.21*	953.66	953.66	953.66
SVRG	929.63	722.19	668.63	715.98*	1032.56	744.027	673.73	715.98*
SSVRG	905.49	658.55	649.08*	646.45	1016.06	696.97	649.08*	640.42
SARAH	844.56	603.01*	641.40	603.08	1055.55	603.01*	624.5	680.04

Table 4.3: Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the 2-agent swap problem. * denotes the optimal loss value obtained when the algorithm starts to converge at the specific # of gradient counts/runtime.

loss of 967.21 with two million gradient counts, SARAH reduces the loss to 844.56 (12.7% more optimal than ADAM), SVRG to 929.63 (3.9% more optimal), and SSVRG 905.49 (6.8% more optimal) using the same number of gradient counts (Table 4.3). However, ADAM converges slightly faster than VR methods in runtime, but their convergence rates regarding runtime are still of a comparable magnitude (Fig 4.4b). When ADAM spends 7.35 minutes to converge, SARAH reaches the loss of 1055.55 (9.1% less optimal than ADAM), SVRG to 1032.56 (6.8% less optimal), and SSVRG to 1016.06 (5.1% less optimal) when spending the same runtime (Table 4.3). After ADAM converges and stops reducing the loss, VR methods still improve the solution by dropping the loss. Eventually, SARAH converges to the loss of 603.01 (37.7% more optimal than ADAM), SVRG to 715.98 (26.0% more optimal), and SSVRG to 649.08 (32.9% more optimal). The runtime of VR solutions, on the other hand, rises significantly. SARAH takes 28 times more time than ADAM, SVRG 36 times more, and SSVRG 33 times more to reach the optimal points. Based on **Figures 4.4 (c)** and **(d)**, we found

that ADAM has lower G values, while VR methods have lower L values. This indicates that VR methods are focused on reducing running costs, but ADAM is focused on minimizing terminal costs.

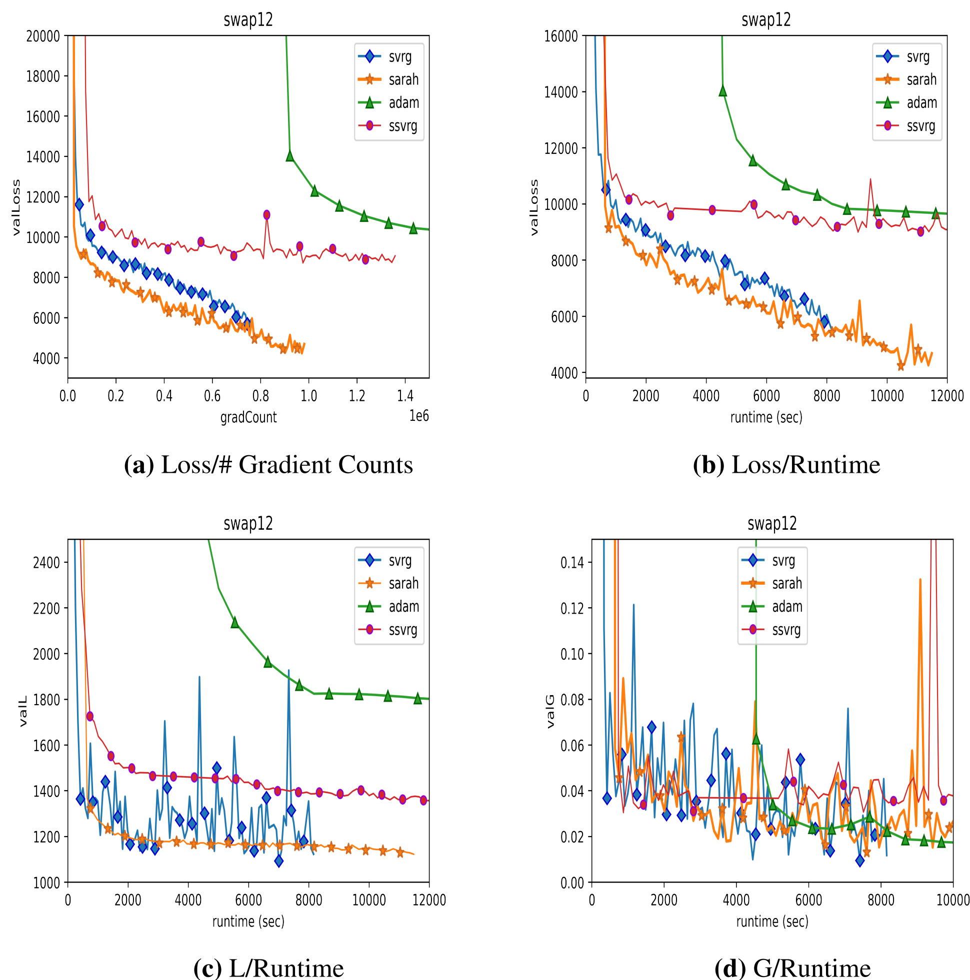
In conclusion, all VR methods are more effective than ADAM in terms of gradient counts and also achieve lower loss values. Like the corridor problem, SARAH performs best in gradient counts and reaches the lowest loss values. Regarding the runtime, VR methods converge slightly slower than ADAM but still achieve comparable convergence rates at the beginning. However, after ADAM converges, the time VR methods spent on training is much greater than the rate at which the value of loss decreases.

4.3.3 12-agent Swap Experiment

	# Gradient Counts				Runtime (min)			
Optimizers	300,000	600,000	700,000	2,000,000	45.67	106.48	139.52	165.01
ADAM	470589	85811	244819	9827.56*	8253.13	7803.86	6398.18	9827.56*
SVRG	8265.41	6529.64*	5915.58	6529.64**	8430.76	6529.64*	6520.64	5620.07
SSVRG	9727.17*	9567.25	9294.09	9727.17**	9727.17*	9345.56	9173.66	9370.37
SARAH	7204.48	5953.54	5764.09*	5764.09**	7437.45	5770.99	5764.09*	6503.17

Table 4.4: Comparison of validated loss for different algorithms at different levels of gradient counts (left) and runtime (right) on the 12-agent swap problem. * denotes the optimal loss value obtained when the algorithm starts to converge at the specific # of gradient counts/runtime. ** indicates that we did not run long enough to achieve the same number of gradient counts but used the values obtained at the point of convergence.

In the 12-agent swap experiment mentioned in [14], six pairs of agents swap their position without any collisions. More details for the problem set-up can be



(a) Loss/# Gradient Counts

(b) Loss/Runtime

(c) L/Runtime

(d) G/Runtime

Figure 4.5: Comparison of validated loss, running cost (L), and terminal cost (G) for different algorithms regarding the number of gradient counts and runtime on the 12-agent swap problem. It can be seen that SARAH is the most efficient at sampling and takes the shortest time to converge among all of the methods. SARAH also delivers the most optimal solutions. SVRG and SSVRG perform well in terms of gradient computation cost, runtime, and solution quality.

found in [1].

The numerical outcomes of the 12-agent swap experiment are shown in **Fig-**

ure 4.5 and **Table 4.4**. From **Figure 4.5** (a) and (b), we see that all VR methods (SVRG, SSVRG, and SARAH) outperform ADAM in terms of gradient computational cost and runtime. This implies that VR methods have higher sampling efficiency than ADAM and can obtain more optimal solutions with better convergence rates. In particular, SARAH performs the best among all three VR methods. Compared to ADAM, SARAH converges to a 41.3% more optimal solution while taking 15% less runtime and spending 65% less gradient computation cost. SVRG performs slightly worse than SARAH but much better than SSVRG in terms of gradient counts, runtime, and solution quality. **Figures 4.5** (c) and (d) suggest that VR methods have lower L values, and all four algorithms have similar G values. This indicates that VR methods outperform ADAM when reducing running costs.

In conclusion, SARAH is the most sampling efficient and takes the shortest runtime to converge among all the techniques we have evaluated. SARAH also yields the lowest loss value. Furthermore, SVRG and SSVRG perform well on 12-agent swap problems.

Chapter 5

Discussion

Our experiments demonstrate the effectiveness of VR methods in solving high-dimensional optimal control problems, especially for multi-agent collision avoidance systems in non-convex settings. In particular, we find that VR methods have higher sampling efficiency than ADAM, requiring less gradient-based computations while maintaining a similar or faster convergence rate. A sampling-efficient method requires less computational cost to achieve a certain level of accuracy and therefore improves applicability [15]. VR methods use the data from each sample to its maximum potential because they minimize the variance in each NN parameter updating, improving sampling efficiency. High-dimensional problems can suffer from a lack of or poor-quality samples in real life. Given high sampling efficiency, VR methods may perform well in practice since they offer improved generalization errors.

In the experiments, we also observe that as the dimensionality of the optimal

control problem increases, the convergence rate for VR methods becomes faster compared to ADAM. For example, when the dimensionality is small, as in the corridor and 2-agent swap problems where $d = 4$, VR methods generally yield a comparable convergence rate as ADAM. When the dimensionality of the problem becomes higher, as in the 12-swap problem where $d = 24$, the advantage of the VR methods in terms of convergence speed becomes obvious. Accordingly, VR methods are well suited for optimizing high-dimensional OC problems.

Furthermore, VR methods generally converge to more optimal solutions with lower loss values than ADAM. For example, SARAH obtains 7.9% more optimal solutions in the corridor experiment, 37.7% more optimal solutions in the 2-agent swap experiment, and 41.3% more optimal solutions in the 12-agent swap experiment.

Based on the numerical results, we found that SARAH has the best performance among all methods we have evaluated. SARAH has the highest sampling efficiency, quickest convergence rate, and lowest loss values. SARAH's outstanding performance in non-convex high-dimensional optimization problems is due to the fact that it utilizes accumulated past stochastic gradient information and continually decreases the variance in the inner loop, whereas SVRG and SSVRG only reduce variance as the number of outer iterations increases [15]. SARAH is, therefore, more stable and will converge to the optimum at a faster convergence rate. SVRG and SSVRG are also very effective, making them viable alternatives. However, SVRG and SSVRG are not particularly robust as SARAH and are vulnerable to changes in hyperparameters. They require a well-tuned learning rate

and other hyperparameters to reach their peak performance. In contrast, SARAH is more robust to hyperparameter changes. Therefore, SVRG and SSVRG are relatively sensitive to different types of high-dimensional optimal control problems and may produce inconsistent results for each trial. Compared to ADAM, VR methods are less stable since they apply mini-batching strategies for better performance in non-convex settings.

We also discovered that VR methods have lower L values on average, while ADAM has lower G values. Note that L values and the total loss have a similar magnitude, while G is virtually zero in all cases. This implies that the agents can get to the small circle centered on the targets most of the time. On the contrary, the running cost (L) is relatively difficult to minimize, and VR methods outperform ADAM in reducing the running costs. Further study is needed to analyze the underlying reasons.

Despite these promising results, this study has several limitations that future work could address. First, we only tune the least set of hyperparameters inherent to the algorithms. Future work should consider more hyperparameters such as the weight decay and the dropout rate to optimize the performance. Second, we only evaluate the performance of VR methods on three specific OC problems. Future work should apply VR methods to a wider variety of problems with higher dimensionality to see if the successful results can be generalized. Third, each inner loop iteration of VR methods has to compute the loss function twice to get the first and second gradient terms given in (3.3). According to the code profiling, the backward propagation is responsible for 39.79% of the training time,

and 48.19% of the runtime is spent computing the loss function. As a result, avoiding recomputing the loss function in each inner loop iteration may help speed up VR methods' convergence rate. For example, using a weighted moving average to estimate the second gradient term given in (3.3) may be beneficial. However, there may be other drawbacks, such as the bias introduced by the estimation. We leave this for future work.

Despite these limitations, our study provides empirical evidence that VR methods (particularly SARAH) are more effective at solving non-convex high-dimensional optimal control problems than ADAM.

Chapter 6

Conclusion

We modify, implement, and evaluate the performance of three variance reduction algorithms (SVRG, SSVRG, and SARAH) on high-dimensional optimal control problems. On three high-dimensional optimal control problems, we fine-tune and compare the effectiveness of SVRG, SSVRG, and SARAH with the baseline algorithm ADAM. The numerical experiments show that variance reduction methods are effective for non-convex, high-dimensional optimal control problems and generally outperform ADAM. SARAH has the best performance in terms of convergence rate, sampling efficiency, and solution optimality because it continually utilizes past gradient information to decrease variance. Compared with SARAH, SVRG and SSVRG are less stable and less adaptive to hyperparameter changes. Future investigations will focus on more challenging high-dimensional optimal control problems with higher dimensionality.

Bibliography

[1] Derek Onken, Levon Nurbekyan, Xingjian Li, Samy Wu Fung, Stanley Osher, and Lars Ruthotto. A neural network approach for real-time high-dimensional optimal control, October 2021. arXiv:2104.03270.

[2] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. arXiv:1412.69800.

[3] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[4] Roy Frostig, Rong Ge, Sham M. Kakade, and Aaron Sidford. Competing with the empirical risk minimizer in a single pass, December 2014. arXiv:1412.6606.

[5] Lam M. Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. June 2017. arXiv:1703.00102.

[6] Sebastian Ruder. An overview of gradient descent optimization algorithms, September 2017. arXiv:2017.00892.

[7] Robert M. Gower, Mark Schmidt, Francis Bach, and Peter Richtarik. Variance-reduced methods for machine learning, October 2020. arXiv:2010.00892.

[8] Mark Schmidt Nicolas Le Roux and Francis Bach. A stochastic gradient method with an exponential convergence rate for finite training sets, March 2013. arXiv:1202.6258.

[9] Francis Bach Aaron Defazio and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives, December 2014. arXiv:1407.0202.

[10] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning, June 2016. arXiv:1606.04838.

[11] Sashank J. Reddi, Ahmed Hefny, Suvrit Sra, Barnabas Poczos, and Alex Smola. Stochastic variance reduction for nonconvex optimization, March 2016. arXiv:1603.06160.

[12] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, July 2019. arXiv:1907.10902.

[13] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures, June 2012. arXiv:1206.5533.

[14] Thulasi Mylvaganam, Mario Sassano, and Alessandro Astolfi. A differential game approach to multi-agent collision avoidance. *IEEE Transactions on Automatic Control*, 62(8):4229–4235, 2017.

[15] Hans Janssen. Monte-carlo based uncertainty analysis: Sampling efficiency and sampling convergence. *Reliability Engineering System Safety*, 109:123–132, 2013.