Towards Robust Reinforcement Learning



Supratik Paul
St Catherine's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2020

Acknowledgements

I would like to thank my supervisor, Shimon Whiteson for his guidance throughout my DPhil. His patience in hearing out my ideas no matter how crazy, and guidance on how to proceed in the face of failure has taught me how to critically analyse ideas and the importance of understanding why something works or fails. I am also deeply grateful to Mike Osborne for being there to explore ideas with, and his guidance on the papers we collaborated on, and especially for highlighting the importance of being mathematically precise in my writing. Thanks also to Yarin Gal and Marc Deisenroth for serving as examiners for the thesis - I thoroughly enjoyed our discussion and I am very grateful for all the constructive feedback.

I would also like to thank my collaborators, Konstantinos Chatzilygeroudis, Kamil Ciosek, Jean-Baptiste Mouret. I am especially grateful to Konstantinos for single handedly managing all the experiments with the physical robots on our work on ALOQ. I am deeply appreciative of Kyriacos Shiralis and Feryal Behbahani's support during my time at Latent Logic – the idea for HOOF came during my internship there, and the reason I followed up on it is due in no small part to us mulling it over one evening at the Jam Factory. My time at WhiRL was spent with an amazing set of friends and colleagues who were always eager to talk through ideas and encourage each other. In particular, I would like to thank Vitaly Kurin for all the discussions we had about everything from optimisation techniques to graphnets, and for being an absolutely amazing collaborator. I am also

very grateful to Tabish Rashid for doing much of the heavy lifting on our server related tasks.

My time at Oxford was significantly enriched by my 'acquaintances' at Catz – Ikuya, Jakob, Pri, and Steve. It is said that working on a DPhil is a lonely affair, but thanks to you, it certainly didn't feel like that for me. Your friendship means the world to me. Thank you for putting up with everything, though I am sure my charming and endearing nature made it easy. Thanks also to Chandak, RG, Shrikant, and Sumana, for being my travel buddies, providing me with a base in London, and being the incredibly amazing friends that they are. I couldn't have asked for more. And finally, I would like to thank my parents, and those who always believed in me throughout my rather unconventional journey across academia and industry.

Abstract

While reinforcement learning (RL) algorithms have been successfully applied to a wide range of problems – from improving energy efficiency of data centres (Gamble and Gao, 2018), trading financial products (Srinivasan, 2020), operating self driving cars (Kendall et al., 2018), etc., they have difficulty coping with environments characterised by *significant rare events*, which have a low probability of occurrence, but can have a significant impact in determining the optimal policy. Furthermore, the performance of existing algorithms has been shown to be remarkably sensitive to their hyperparameters. As a result, when faced with a new problem the common practice is to perform a search over these hyperparameters. While this is not a major issue in settings where generating fresh interactions with the environment is cheap (for example, learning policies for playing Atari games, or simulated robotics tasks), this can preclude existing RL algorithms from being applied to problems where interactions with the environment are significantly more expensive.

In this thesis, we focus on the issue of robustness of RL algorithms with regards to the problem of rare events and its own hyperparameters. We present two algorithms – Alternating Optimisation and Quadrature (ALOQ), and Fingerprint Policy Optimisation (FPO), that address the problem of robustness to rare events. Both these algorithms are based on the principle of actively generating experience that takes into account the effect of any rare events, rather than relying on random sampling as is

common to existing methods. While ALOQ is a self-contained method that uses Bayesian optimisation and Bayesian quadrature to perform policy search, FPO is designed to make existing policy gradient methods more robust to these settings. We also present Hyperparameter Optimisation on the Fly, a gradient free algorithm designed to automatically learn the hyperparameters of policy gradient methods while requiring no more interactions than what would be normally collected within one training run of the underlying policy gradient method. We empirically validate our algorithms through experiments across multiple domains.

Contents

1 Introduction **1**

1.1 Robustness to Rare Events 4

1.2 Robustness to Hyperparameters 5

1.3 Contributions and Thesis Overview 7

2 Background **10**

2.1 Markov Decision Process 10

2.2 The Reinforcement Learning Problem 11

2.3 Direct Policy Search 13

2.4 Policy Gradient Methods 13

2.4.1 REINFORCE 14

2.4.2 Variance Reduction Techniques 15

2.4.3 Natural Policy Gradients 16

2.4.4 Trust Region Policy Optimisation 18

2.5 Gaussian Process 18

2.5.1 Learning the Hyperparameters 19

2.5.2 Bayesian Optimisation for Direct Policy Search 20

2.5.3 Bayesian Quadrature 22

3 Alternating Optimisation and Quadrature **24**

3.1 Problem Setting 25

3.2 ALOQ 27

3.3 TALOQ 31
3.3.1 BO with Reality Gap 31
3.3.2 Complete TALOQ Method 33
3.4 Properties of ALOQ and TALOQ 35
3.5 Experimental Results 35
3.5.1 Baseline Methods 35
3.5.2 ALOQ Experiments 37
3.5.2.1 Artificial Test Functions 37
3.5.2.2 Robotic Arm Simulator 41
3.5.2.3 Hexapod Locomotion Task 47
3.5.3 TALOQ Experiments 51
3.5.3.1 Simulated Experiments 52
3.5.3.2 Physical Experiments 53
3.5.4 General Experimental Details 54
3.6 Discussion 55

4 Fingerprint Policy Optimisation **57**
4.1 Problem Setting 58
4.2 The Algorithm 60
4.2.1 Selecting ψ_n 61
4.2.2 Estimating $J(\pi_n)$ 62
4.2.3 Policy Fingerprints 63
4.2.4 Covariance Function 64
4.3 Experiments 64
4.3.1 Cliff Walker 65
4.3.2 Half Cheetah 67
4.3.3 Ant 69
4.3.4 Further Examination of the Learnt Policies 72

4.3.5 Performance in settings without SREs . . . 73
4.3.6 Hyperparameter Settings . . . 74
4.4 Discussion . . . 74

5 Hyperparameter Optimisation on the Fly **76**
5.1 Method . . . 78
5.1.1 First Order HOOF . . . 79
5.1.2 $(\boldsymbol{\gamma}, \boldsymbol{\lambda})$ Conditioned Value Function . . . 80
5.1.3 Robustness to HOOF Hyperparameters and Computational Costs 81
5.1.4 Choice of Optimiser . . . 81
5.2 Experiments . . . 82
5.2.1 Illustration of the Ordering Effect of WIS . . . 82
5.2.2 HOOF with A2C . . . 83
5.2.2.1 Comparison to Meta-Gradients . . . 84
5.2.2.2 Robustness to the Choice of KL Constraint . . . 87
5.2.2.3 Robustness to the Choice of Optimiser . . . 87
5.2.2.4 Performance of HOOF without a KL Constraint . . . 87
5.2.2.5 Sample Efficiency vs. Grid Search . . . 88
5.2.2.6 Implementation Details . . . 88
5.2.3 HOOF with TNPG . . . 89
5.2.3.1 Importance of Learning (γ, λ) Conditional Value Functions . . . 90
5.2.4 Implementation Details . . . 90
5.3 Discussion . . . 91

6 Related Work **96**
6.1 Optimising in the presence of environment variables . . . 96
6.2 Learning Policies Robust to SREs . . . 97

6.3 Learning policies for physical systems 99
6.4 Hyperparameter Optimisation for Reinforcement Learning 101

7 Conclusion **105**

7.1 Alternative Formulation for Settings with SREs 106
7.2 Expanding the Scope of Hyperparameter Optimisation Algorithms . . 107

Bibliography **108**

List of Figures

2.1 The reinforcement learning problem: agent - environment interaction 12

3.1 Brief explanation of ALOQ 27
3.2 Effect of beta warping 30
3.3 An illustrative example of the effect of lengthscale 33
3.4 Comparison of performance on test functions 39
3.5 Comparison of runtimes on test functions 39
3.6 Contour plot, performance, and runtimes on F-SRE1 41
3.7 Contour plot, performance, and runtimes on F-SRE2 42
3.8 Performance and learned configurations on the robotic arm collision avoidance task. 44
3.9 Performance and learned configurations on the robotic arm joint breakage task. 45
3.10 Performance of REINFORCE and TRPO on the robotic arm simulator tasks. 46
3.11 Hexapod locomotion problem. 49
3.12 Brief explanation of TALOQ 52
3.13 Performance of TALOQ on the simulated experiments. 53

4.1 Brief explanation of FPO 59
4.2 FPO results for the Cliff Walker environment 66
4.3 FPO results for the Half Cheetah environment 69

4.4 FPO results for the Ant environment 70
4.5 Comparison of FPO-UCB(S) against the Fixed baselines 72
4.6 Histogram of the velocity profile of the learnt policies 72
4.7 FPO results for the Cliff Walker environment without any SRE . . . 74

5.1 Illustration of the Ordering Effect of WIS 83
5.2 Performance of HOOF with A2C 84
5.3 Schedule for the learning rates learnt by HOOF 85
5.4 Performance of HOOF showing robustness to the choice of optimiser . 93
5.5 Performance of HOOF without any KL constraint 94
5.6 Performance of HOOF-TNPG vs TRPO baselines. 94
5.7 Hyperparameters learnt by HOOF-TNPG 95
5.8 Performance of HOOF without (γ, λ) conditioned value functions . . 95

Chapter 1

Introduction

Reinforcement learning (RL) is concerned with sequential decision making problems. In such settings, an agent operates in an environment and takes a sequence of actions, with the overall goal of maximising some performance metric. At each timestep, the agent receives some observation, takes an action, and receives a reward, together with the next observation. The agent's goal is to learn a *policy* that determines which action to take given an observation, such that it maximises the cumulative reward. For example, we might be interested in having a robot learn to navigate around a room with different obstacles, to reach some specified destination as fast as possible. At every timestep, the sensors on the robot provides some observations to it. The robot's policy then decides which action to take based on these observations. Since we want the robot to learn to get to its goal as fast as possible, we might give it a small negative reward at each timestep, while giving it a large positive reward once it reaches the goal.

Since very little prior knowledge about the environment is available to the agent, it needs to take some exploratory actions to find policies that can lead to high performance. At the same time, it needs to exploit the knowledge gained through exploration by focusing on policies that led to high rewards. A central goal of RL is to develop methods that are *sample efficient*, i.e. require as few interactions with the environment as possible to learn high performing policies. To be able to achieve

this, the agent needs to balance exploration vs exploitation: too much of the former would lead to wasted interactions with the environment that explore regions of low rewards, while too much of the latter can lock the agent into a suboptimal policy as it may completely miss a region with high performance simply because it didn't explore enough. This requires the agent to take potentially exploration actions throughout learning, and this is normally achieved in different ways. For example, some algorithms rely on taking random exploratory actions with a certain probability, while others assume that all states have a potentially high returns, or increasing the uncertainty associated with the potential rewards for states that are not visited by the agent often enough.

Applying RL methods to physical systems brings its own set of challenges that are different compared to the commonly used benchmarking environments in RL research, like the Arcade Learning Environment (Bellemare et al., 2013) and simulated continuous control tasks like those in OpenAI Gym (Brockman et al., 2016). Physical systems tend to be partially observable, and involve high-dimensional, continuous state and action spaces which makes them particularly difficult to address using value based methods (Sutton and Barto, 2018), which are based on estimating the expected return of a policy at every point in the state space. The partially observable nature of these tasks violate the Markov assumption that value based methods rely on. Convergence guarantees of value based methods are usually based on tabular settings or linear function approximators and may not hold in these settings (Sutton and Barto, 2018).

In this thesis we are primarily concerned with policy search methods, particularly direct policy search and policy gradients (Williams, 1992; Sutton et al., 1999; Konda and Tsitsiklis, 2000). Compared to value based methods, these methods search for the optimal policy directly in the policy space. They also scale better with the dimensionality of the environment, and continuous action spaces do not pose any particular issue since the search is performed in the policy space.

Policy gradient methods improve upon a policy by performing gradient ascent on the policy parameters. While this requires estimating the gradient of the policy's performance, the key advantage is that these methods scale well with the dimensionality of policy parametrisation, and have shown considerable success in environments characterised by large or continuous action spaces (Mordatch et al., 2015; Schulman et al., 2016; Rajeswaran et al., 2017b). Since the outputs of policies are usually non-linear functions of their parameters (especially for large neural net policies), changes in policy parameters can lead to unexpectedly large changes in outputs. Since these methods update the policy gradually, they reduce the possibility of deploying policies that can lead to dangerous outcomes.

Direct policy search treats the problem of finding the optimal policy as a black box optimisation problem, while ignoring any underlying temporal structure. While these methods are less amenable to high dimensional policies, like deep neural networks, they have been shown to be competitive to policy gradient methods on various continuous control tasks (Duan et al., 2016a; Salimans et al., 2017; Mania et al., 2018). They also do not make any assumptions about the full observability or Markov property of the task, and specific methods can also be highly sample efficient which makes them suitable for tasks where sample efficiency is especially important.

In this thesis we note in particular two issues that have a severe impact on the sample efficiency of existing RL methods:

1. In certain environments there exists rare events that yield significantly different rewards. This can severely compromise the sample efficiency of existing methods, as their reliance on random sampling to observe these rare events can lead to very slow learning, or worse, failing to learn at all.

2. Existing policy gradient methods are known to be sensitive to certain hyperparameters. Thus to achieve high performance, these algorithms require careful tuning, which in turn requires multiple training runs, thereby sacrificing sample

efficiency.

To address these two issues, we need to develop methods that are robust to such rare events, and their own hyperparameters. In the following two sections we discuss these issues in detail.

1.1 Robustness to Rare Events

Unlike the benchmarking environments where obtaining simulated experience is relatively fast and cheap, a key consideration when applying RL to a physical setting is the risk and expense of running trials. Another consideration is the robustness of the learned policies. Since it is typically infeasible to test a policy in all contexts, it is difficult to ensure it works as broadly as intended. Fortunately, policies can often be trained and tested in a simulator that exposes key *environment variables* – state features that are unobservable to the agent and randomly determined by the environment in a physical setting, but that are controllable in the simulator.

Environment variables are ubiquitous in real-world settings. For example, localisation errors may mean that a robot is much nearer to an obstacle than expected, increasing the risk of a collision. While an actual robot cannot directly trigger such errors, they are trivial to introduce in simulation. Similarly, if we want to train a helicopter to fly robustly, we can easily simulate different wind conditions, even though we cannot control weather in the real world.

A naïve application of existing policy search methods would involve randomly sampling the environment variable in each trial, so as to estimate expected performance. However, this approach (1) often requires testing each policy in a prohibitively large number of scenarios, and (2) is not robust to *significant rare events* (SREs), i.e., it fails any time there are rare events that substantially affect expected performance. For example, rare localisation errors may mean that a robot is much nearer to an obstacle than expected, increasing the risk of a collision. Since collisions are so catastrophic,

avoiding them is key to maximising expected performance, even though the factors contributing to the collision occur only rarely. Similarly, when learning to fly a helicopter, some rare wind conditions may increase the risk of a crash. Since crashes are catastrophic, accurately reasoning about them is key to evaluating the expected performance, even though the wind conditions contributing to the crash occur only rarely. In such cases, the naïve approach will not see such rare events often enough to learn an appropriate response.

Simply increasing the number of trials is infeasible as this can be orders of magnitude higher than in the case where there are no SREs. This is even more of a challenge when applying RL to a physical setting, for example while learning the optimal policy for a robot, where running trials can be expensive; repeated trials on a robot can lead to wear and tear of the joints which can increase the noise associated with system, or even fail altogether. Thus, in order to preserve sample efficiency, any method applied to such problems cannot rely on random sampling alone to discover the impact of SREs.

1.2 Robustness to Hyperparameters

Like other gradient-based optimisation methods, the performance of policy gradient methods can be sensitive to a number of key hyperparameters. For example, the performance of REINFORCE (Williams, 1992) and A2C (Mnih et al., 2016) can depend critically on the learning rate, the choice of which in turn often depends on the task, the particular policy gradient method in use, and even the choice of optimiser. Henderson et al. (2018b) has shown that in certain simulated continuous control tasks, RMSProp (Tieleman and Hinton, 2012) and ADAM (Kingma and Ba, 2014) have narrow ranges for good learning rates which may not be known a priori. Even though methods based on natural gradients, like Natural Policy Gradients (NPG) (Kakade, 2001) or Trust Region Policy Optimisation (TRPO) (Schulman et al., 2015), are more

robust to the KL divergence constraint (which can be interpreted as a learning rate), significant performance gains can often be obtained by tuning this parameter (Duan et al., 2016a). Similarly, variance reduction techniques such as Generalised Advantage Estimators (GAE) (Schulman et al., 2016), which trade variance for bias in policy gradient estimates, introduce key hyperparameters (γ, λ) that can also greatly affect performance (Schulman et al., 2016; Mahmood et al., 2018).

Given such sensitivities, there is a great need for methods that are robust to their hyperparameters. One way to achieve this would be specifically develop robust variants of policy gradient methods. An alternative approach, and perhaps also more powerful, would be to develop general purpose hyperparameter optimisation techniques that can automatically tune the hyperparameter of any policy gradient algorithm.

Perhaps the most popular hyperparameter optimisation technique is simply grid search (Schulman et al., 2015; Mnih et al., 2016; Duan et al., 2016a; Igl et al., 2018; Farquhar et al., 2018). More sophisticated techniques such as Bayesian optimisation (BO) (Srinivas et al., 2010; Hutter et al., 2011; Snoek et al., 2012; Chen et al., 2018) have also proven effective, and new innovations such as Population Based Training (PBT) (Jaderberg et al., 2017) and meta-gradients (Xu et al., 2018) have shown considerable promise. Furthermore, a host of methods have been proposed for hyperparameter optimisation in supervised learning (see Section 6.4).

However, all these methods suffer from a major problem: they require performing many learning runs to identify good hyperparameters. This is particularly problematic in reinforcement learning, where it incurs not just computational costs but sample costs, as new learning runs typically require fresh interactions with the environment. This sample inefficiency is obvious in the case of grid search, BO based methods and PBT. However, even meta-gradients, which reuses samples collected by the underlying policy gradient method to train the meta-learner, requires multiple training runs. This is because the meta-learner introduces its own set of hyperparameters, e.g.,

meta learning rate and reference (γ, λ), all of which need tuning to achieve good performance.

Furthermore, grid search and BO based methods typically estimate only the best fixed values of the hyperparameters, which often actually need to change dynamically during learning (Jaderberg et al., 2017; François-Lavet et al., 2015). This is particularly important in reinforcement learning, where the distribution of visited states, the need for exploration, and the cost of taking suboptimal actions can all vary greatly during a single learning run.

To make hyperparameter optimisation practical for reinforcement learning methods such as policy gradients, we need radically more efficient methods that can dynamically set key hyperparameters on the fly, not just find the best fixed values, and do so within a single run, using only the data that the baseline method would have gathered anyway, without introducing new hyperparameters that need tuning.

1.3 Contributions and Thesis Overview

In this thesis we study the problem of sample efficient learning in environments characterised by SREs. To this end we develop three main algorithms:

1. **Alternating Optimisation and Quadrature**: This is a direct policy search algorithm, based on Bayesian Optimisation and Bayesian Quadrature, that is both highly sample efficient and can learn in the presence of SREs. We also develop a variant, **Transferable Alternating Optimisation and Quadrature**, which additionally uses trials from an imperfect simulator to further reduce the number of physical trials required for learning. Since ALOQ and TALOQ use BO, they also suffer from its disadvantages, i.e. they do not scale to high dimensional policies, for example, deep neural networks which might have thousands of parameters.

2. **Fingerprint Policy Optimisation**: This method address the issue of scalability to high dimensional policies by using a policy gradient method instead of Bayesian Optimisation, while still being able to learn a good policy in the presence of SREs through active selection of the environment variable. However, it is far less sample efficient, which is further exacerbated by the need for tuning the hyperparameters of the underlying policy gradient method.

3. **Hyperparameter Optimisation on the Fly**: This addresses the problem of finding the optimal hyperparameters of a policy gradient algorithm sample efficiently. Compared to existing methods, it learns the optimal hyperparameters within a single training run.

These are presented in Chapters 3, 4, and 5 respectively. Some of the background material relevant to understanding these algorithms is presented in Chapter 2.

In Chapter 6 we present a review of the existing literature on learning in the presence of rare events, and hyperparameter optimisation.

Finally, in Chapter 7 we briefly discuss some of the highlights of this thesis.

A summary of the algorithms is presented in Table 1.1.

Algorithm	Problem addressed	Leverages	Advantages	Disadvantages
ALOQ	Robustness to SREs	Bayesian optimisation and quadrature	Sample efficient No MDP assumption	Can only learn low dimensional policies Assumes no reality gap
TALOQ	Robustness to SREs	Bayesian optimisation and quadrature	Sample efficient No MDP assumption Bridges the reality gap	Can only learn low dimensional policies
FPO	Robustness to SREs	Bayesian optimisation and policy gradients	Can learn high dimensional neural net policies	Requires hyperparamter tuning which makes it sample inefficient
HOOF	Robustness to hyper-parame-ters	Importance sampling and policy gradients	Only requires one training run to learn a policy Fast, simple, and easy to implement	Can only be used to learn hyperparameters that directly affect the gradient

Table 1.1: Summary of the algorithms presented in this thesis.

Chapter 2

Background

In this chapter we formalise the problem settings and define the notation that will be used throughout the thesis. We also present some of the technical background necessary to understand the algorithms presented in subsequent chapters.

2.1 Markov Decision Process

We use a *Markov decision process* (MDP) defined by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma \rangle$, to formalise our learning environment. Here

- $\mathcal{S}$ is the state space for the environment.
- $\mathcal{A}$ the set of actions that can be taken by the agent.
- $\mathcal{P} = P(s'|s, a,)$ the transition function. Given a state s, and action a, the transition function determines the probability distribution over the next state s'.
- $r = r(s, a, s')$ the reward function, which determines the distribution over the immediate reward that the agent receives by taking action a in state s and transitioning to state s'.
- $\gamma \in [0, 1)$ is the discount factor which determines how future rewards should be discounted.

The actions taken by the agent are determined by a *policy* which maps states to actions: $\Pi(a|s)$ determines the probability distribution over action $a \in \mathcal{A}$ given the state $s \in \mathcal{S}$.

We are inherently limited in our ability to model any reasonably complex physical system perfectly – even if the underlying system is deterministic, it is impossible to capture all the variables that affect the system. The effect of any unobserved/unmodelled variable on a transition is captured as part of the stochasticity in the transition function. While such variables are usually not explicitly considered in the notation, we make this dependence explicit in settings characterised by environment variables by using the notation $\mathcal{P}_\theta$, where θ is the setting of the environment variable. However, note that we do not include it as part of the state space of the MDP since it is unobservable to our agent.

For example, while working with a mobile robot, the true coefficient of friction between different surfaces might be unknown, and unobservable to our agent. If we weren't particularly concerned about reasoning about its effect on the transitions, we could treat it as part of the stochastic transition function. However, to explicitly take into account its effect we can treat it as an environment variable, and make it clear as such through the notation.

2.2 The Reinforcement Learning Problem

At any given timestep t, the agent observes the current state s_t and takes an action a_t based on some policy $\Pi_\pi(a_t|s_t)$, where Π is a policy with its parameters denoted by π. It receives a reward r_t, and the environment transitions to the state s_{t+1} based on the transition function $\mathcal{P}_\theta$. In this thesis we specifically consider the episodic setting, i.e. starting from any state, and given any policy, the environment will eventually transition to some terminal state s_T, upon which the episode ends. We refer to this sequence of states, actions, and rewards $\{s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T\}$ as a *trajectory* τ.

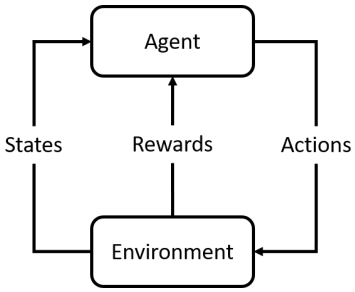


Figure 2.1: The reinforcement learning problem: agent - environment interaction

The *return* R of a trajectory τ is defined as $\sum_{t=0}^{T-1} \gamma^t r_t$. Note that both τ and R depend on the policy defined by its parameters π, and the environment variable θ. This is presented in Figure 2.1.

The expected return of a policy defined by its parameters π is defined as:

$$\begin{aligned} J(\pi) &= \mathbb{E}_{\theta \sim p(\theta), a \sim \Pi_\pi(a|s), s \sim \mathcal{P}} \left[R(\pi, \theta) \right] \\ &= \mathbb{E}_{p(\tau|\pi)}[R(\tau)]. \end{aligned} \tag{2.1}$$

Note that here τ is a random variable with $p(\tau|\pi)$ being the probability distribution of observing the trajectory τ under the policy Π. For ease of notation we will write $J(\pi) = \mathbb{E}_\theta \left[R(\pi, \theta) \right]$ when working with settings with environment variables.

Our objective is to find an optimal policy Π^*. Since we consider parametrised policies, this is equivalent to finding the optimal parameters π^* that maximises $J(\pi)$:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, J(\pi). \tag{2.2}$$

The problem of estimating $J(\pi)$ is known as *policy evaluation*. In general, $J(\pi)$ cannot be computed analytically. However, it can be estimated using a Monte Carlo approach based on observed trajectories: We simply sample multiple trajectories $\tau_1, \tau_2, ..., \tau_K$ by following Π, and estimate $J(\pi)$ as $\frac{1}{K} \sum_{k=1}^{K} R_k$, where R_k is the corresponding return for τ_k. While easy to implement, a significant disadvantage of this approach is that the variance of the estimate can be quite large for highly stochastic environments (or policies), and thus a large number of trajectories might be required to form reliable estimates of $J(\pi)$.

Note that for settings with environment variables, θ is unobservable to the agent, and thus the policy does not condition on θ at any point.

2.3 Direct Policy Search

Direct policy search treats the objective of finding the optimal parameters π^* as a black box optimisation problem. Direct policy search can involve any gradient free optimisation technique, like the cross entropy method (Rubinstein, 1999), covariance matrix adaptation - evolutionary search (Hansen and Ostermeier, 2001), simulated annealing Kirkpatrick et al. (1983), and Bayesian Optimisation. Starting with some candidate policies, these methods rely on Monte Carlo evaluation to estimate the value of the candidates. This information is then used for policy improvement by generating fresh candidates from potentially high performing areas of the search space.

By treating policy search as a black box optimisation problem, direct policy search methods ignore the temporal aspect of the problem. They do not take into account the information provided by the immediate rewards available at each timestep, or the states that are experienced throughout individual trajectories. While it has been argued that ignoring such information makes these methods ill-suited for RL problems Sutton and Barto (2018), it has been shown that these methods can be quite competitive in certain situations Duan et al. (2016a). On the other hand, one advantage of treating the problem as black box optimisation is that it does not require the underlying environment to be an MDP.

2.4 Policy Gradient Methods

The main idea behind policy gradient methods is to perform gradient ascent on the policy parameters by repeatedly computing the gradient of $J(\pi)$ with respect to the

policy parameters. Concretely, the update follows the rule:

$$\pi' = \pi + \alpha \nabla_\pi J(\pi), \tag{2.3}$$

where α is the learning rate. Note that this requires computing the gradient of an expectation: $\nabla_\pi J(\pi) = \nabla_\pi \{\mathbb{E}_{p(\tau|\pi)}[R(\tau)]\}$. Since no prior knowledge about the MDP is assumed, this cannot be computed analytically. Thus, we resort to Monte Carlo gradient estimation, wherein $\nabla_\pi J(\pi)$ is estimated through interactions with the environment. While this problem has been studied extensively, and multiple approaches have been proposed in the literature to estimate this gradient (see Mohamed et al. (2020) for an overview), in this thesis we focus on the likelihood-ratio trick.

2.4.1 REINFORCE

One of the earliest policy gradient algorithms, REINFORCE (Williams, 1992), uses the likelihood-ratio trick to compute the gradient $\nabla J(\pi)$. Suppose we are interested in computing $\nabla_\pi p(\tau|\pi)$. The likelihood-ratio trick is given by the identity $\nabla_\pi p(\tau|\pi) = p(\tau|\pi)\nabla_\pi \log p(\tau|\pi)$. Now, from (2.1) we have

$$\begin{aligned}
\nabla_\pi J(\pi) &= \nabla_\pi \mathbb{E}_{p(\tau|\pi)}[R(\tau)] \\
&= \nabla_\pi \int_\tau R(\tau) p(\tau|\pi) d\tau \\
&= \int_\tau R(\tau) \nabla_\pi p(\tau|\pi) d\tau \\
&= \int_\tau R(\tau) p(\tau|\pi) \nabla_\pi \log p(\tau|\pi) d\tau \\
&= \mathbb{E}_{p(\tau|\pi)} \left[R(\tau) \nabla_\pi \log p(\tau|\pi) \right],
\end{aligned} \tag{2.4}$$

Interestingly, we do not need the transition function $p(s'|s,a)$ to compute $\nabla_\pi \log p(\tau|\pi)$:

$$\begin{aligned}
\nabla_\pi \log p(\tau|\pi) &= \nabla_\pi \log \left[p(s_0)\Pi_\pi(a_0|s_0)p(s_1|s_0,a_0)...\Pi_\pi(a_{T-1}|s_{T-1})p(s_T|s_{T-1},a_{T-1})\right] \\
&= \nabla_\pi \left[\log p(s_0) + \sum_{t=0}^{T-1} \log \Pi_\pi(a_t|s_t) + \sum_{t=0}^{T-1} \log p(s_{t+1}|s_t,a_t)\right] \\
&= \nabla_\pi \log p(s_0) + \sum_{t=0}^{T-1} \nabla_\pi \log \Pi_\pi(a_t|s_t) + \sum_{t=0}^{T-1} \nabla_\pi \log p(s_{t+1}|s_t,a_t) \\
&= \sum_{t=0}^{T-1} \nabla_\pi \log \Pi_\pi(a_t|s_t), \qquad (2.5)
\end{aligned}$$

where the last line follows from that fact that $p(s_0)$ and $p(s_{t+1}|s_t,a_t)$ are independent of π. The update for REINFORCE thus simplifies to:

$$\pi' = \pi + \alpha \mathbb{E}_{p(\tau|\pi)} \left[\sum_t \nabla_\pi \log \Pi_\pi(a_t|s_t) R(\tau)\right]. \qquad (2.6)$$

where the expectation can be approximated by a Monte Carlo estimate using sampled trajectories.

2.4.2 Variance Reduction Techniques

A major problem with the REINFORCE update is that the estimate of $\nabla J(\pi)$ can have high variance due to stochasticity in the policy and environment, yielding slow learning (Williams, 1992; Glynn, 1990). One technique to address this is through the use of control variates (Owen, 2013; Ross, 1990) which subtracts a baseline from the return $R(\tau)$. For a baseline to successfully reduce the variance, it needs to be correlated to the return. Further variance reduction can be achieved using the insight that past rewards do not depend on future actions. Combining these two yields the policy gradient update:

$$\pi' = \pi + \alpha \mathbb{E}_{p(\tau|\pi)} \left[\sum_{t=0}^{T-1} \nabla_\pi \log \Pi_\pi(a_t|s_t) \left(\sum_{t'=t}^{T-1} r'_t - b(s_t)\right)\right]. \qquad (2.7)$$

A commonly used baseline $b(s_t)$ is the value function $V(s_t) = \mathbb{E}_{a\sim\Pi, s\sim\mathcal{P}}[r_t + \gamma r_{t+1} + ... + \gamma^{T-2} r_{T-1}]$.

The gradient estimate can be written in a general form as

$$g = \sum_{t=0}^{T-1} \nabla_\pi \log \Pi_\pi(a_t|s_t)\Omega_t, \tag{2.8}$$

where Ψ_t can take different forms. For example, $\Omega_t = R(\tau)$ yields the REINFORCE update, while $\Omega_t = \sum_{t'=t}^{T-1} r'_t - b(s_t)$ is the baselined version.

By discounting future rewards and bootstrapping off the value function, Generalised Advantage Estimation (GAE) (Schulman et al., 2016) reduces the variance due to rewards observed far in the future, but adds bias to the policy gradient estimate. It introduces two hyperparameters - $(\gamma, \lambda) \in [0, 1]^2$, which control the bias-variance tradeoff in the estimates. Let $A_t^{(k)} = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$. The GAE for Ω_t is given by:

$$A_t^{GAE(\gamma,\lambda)} = (1-\lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + ...). \tag{2.9}$$

Well chosen (γ, λ) can significantly speed up learning (Schulman et al., 2016; Henderson et al., 2018a; Mahmood et al., 2018).

2.4.3 Natural Policy Gradients

REINFORCE updates the policy in the parameter space. A major problem with this is that small changes in the parameter space can easily lead to large changes in the policy space, or conversely, large updates in the parameter space might be needed to get even small updates in the policy space. We illustrate this through a simple example: Consider an environment with two actions a_0 and a_1. Let

$$p_\pi(a) = \begin{cases} \frac{\exp(\pi)}{1+\exp(\pi)} & \text{if } a = 0 \\ \frac{1}{1+\exp(\pi)} & \text{if } a = 1. \end{cases}$$

The probability of taking each action with $\pi = \{6, 3, 0\}$ is presented in Table 2.1. Going from $\pi = 6$ to $\pi = 3$ leads to an increase 0.05 increase in $p(a = 1)$. However, the same change starting with $\pi = 3$ leads to an increase of 0.45 in $p(a = 1)$.

Table 2.1: Illustration of how similar changes in parameter space can lead to very different changes in the policy space. $p_\pi(a=0) = \exp(\pi)/(1+\exp(\pi))$ and $p_\pi(a=1) = 1/(1+\exp(\pi))$

$p_\pi(a)$	$a=0$	$a=1$
$\pi = 6$	1.00	0.00
$\pi = 3$	0.95	0.05
$\pi = 0$	0.50	0.50

Amari (1998) showed that in the case of perceptrons, the gradient on the parameter space does not correspond to the steepest direction, which instead is given by the natural gradient. Kakade (2001) addresses this problem in RL through Natural Policy Gradient (NPG), and showed that the direction of steepest descent is given by $I(\pi)^{-1}g$, where $I(\pi)$ is the estimated Fisher information matrix (FIM) given by

$$I(\pi) = \mathbb{E}_{a\sim\Pi, s\sim\mathcal{P}}\left[\nabla_\pi \log \Pi_\pi(a|s) \nabla_\pi \log \Pi_\pi(a|s)^T\right]. \tag{2.10}$$

Peters and Schaal (2008) showed that this is the approximate solution to the problem of restricting the change to the policy subject to the constraint $KL(\Pi'||\Pi) \leq \delta$, which leads to the update rule:

$$\pi' = \pi + \sqrt{\frac{2\delta}{g^T I(\pi)^{-1} g}} I(\pi)^{-1} g. \tag{2.11}$$

A major disadvantage of natural policy gradient methods is that they require the inversion of the FIM in (2.11), which can be prohibitively expensive for large neural net policies with thousands of parameters. Truncated Natural Policy Gradients (TNPG) (Duan et al., 2016a) address this by using the conjugate gradient algorithm to efficiently approximate $I(\pi)^{-1}g$. The conjugate gradient method finds an approximate solutions to problems of the form $Ax = b$, where A is a symmetric and positive-definite matrix (in this case, the Fisher information matrix). See Shewchuk (1994) for a detailed description of the conjugate gradient method.

Algorithm 1 Backtracking line search for TRPO

1: Compute the possible policy update $\Delta = \sqrt{\frac{2\delta}{g^T I(\pi)^{-1} g}} I(\pi)^{-1} g$.
2: **for** $n = 0, 1, ..., N$ **do**
3: Updated policy parameters $\pi' = \pi + \alpha^n \Delta$.
4: **if** $KL(\Pi' || \Pi) \leq \delta$ and $J(\pi') > J(\pi)$ **then** Accept the update.
5: **end for**

2.4.4 Trust Region Policy Optimisation

Since the NPG update in (2.11) is only an approximate solution, it is possible for the $KL(\Pi' || \Pi) < \delta$ constraint to be violated. Further, since δ is not adaptive, it might be too large for some iterations leading to a reduction in performance. Trust Region Policy Optimisation (TRPO) addresses these issues by ensuring that the KL-divergence constraint is satisfied. It does this by performing a backtracking line search along the gradient direction. It further requires that the estimated value of the updated policy Π' is greater than that of Π. The backtracking line search is presented in Algorithm 1.

As a result, TRPO is more robust to the choice of δ (Schulman et al., 2015). Concretely, at each iteration the policy update is first computed:

$$\Delta = \sqrt{\frac{2\delta}{g^T I(\pi)^{-1} g}} I(\pi)^{-1} g. \tag{2.12}$$

2.5 Gaussian Process

Gaussian Processes (GP) (Rasmussen and Williams, 2005) provide a principled way of quantifying uncertainties associated with modelling unknown functions. A GP is a distribution over functions, and is fully specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ (see Rasmussen and Williams (2005) for an in-depth treatment) which encode prior belief about the nature of the function. The prior can be combined with observed values to update the belief about the function in a Bayesian way to generate a posterior distribution.

The prior mean function of the GP is often assumed to be 0 for convenience. A popular choice for the covariance function is the squared exponential which has the form

$$k(\mathbf{x}, \mathbf{x}') = \exp\left\{ -\sum_{d=1}^{D} \frac{(x_d - x'_d)^2}{2l_d^2} \right\}, \tag{2.13}$$

where D is the dimensionality of $\mathbf{x}$ and l_d is the lengthscale hyperparameter associated with the dth dimension. The squared exponential covariance function belongs to the class of stationary functions of the form $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x} - \mathbf{x}')$, which implies that the covariance between the function values of any two points depends only on the distance between them. Informally, a large lengthscale along a particular dimension implies that the function values change relatively slowly along that dimension.

In GP regression, it is assumed that the observed function values $\{f(\mathbf{x}_i)\}_{i=1}^N$ are a sample from a multivariate Gaussian distribution. The prediction for a new point $\mathbf{x}^*$ is connected with the observations through the mean and covariance functions. By conditioning on the observed data, this can be computed analytically as a Gaussian $\mathcal{N}\Big(\mathbb{E}\big(f(\mathbf{x}^*)\big), \mathrm{Cov}\big(f(\mathbf{x}^*)\big)\Big)$:

$$\mathbb{E}\big(f(\mathbf{x}^*)\big) = k(\mathbf{x}^*, \mathbf{X})(\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I})^{-1} f(\mathbf{X}) \tag{2.14a}$$

$$\mathrm{Cov}\big(f(\mathbf{x}^*)\big) = k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{X})(\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I})^{-1} k(\mathbf{X}, \mathbf{x}^*), \tag{2.14b}$$

where $\mathbf{X}$ denotes the vector of observed inputs, $f(\mathbf{X})$ the vector of corresponding function values, and $\mathbf{K}$ is the matrix with entries $k(\mathbf{x}_i, \mathbf{x}_j), i, j = 1, 2, ..., N$.

2.5.1 Learning the Hyperparameters

The predictions for any unobserved point given above (2.14) assumes that the hyperparameters ψ are known. This is unrealistic as we are unlikely to have any knowledge about these a priori. However, we can treat these as latent variables that can be inferred from data. In a fully Bayesian setting, we place a prior over the hyperparameters

$p(\psi)$ and then marginalise them out while computing the predictions:

$$p\left(f(\mathbf{x}^*)|\mathbf{X}, f(\mathbf{X}), \mathbf{x}^*\right) = \int p\left(f(\mathbf{x}^*)|\mathbf{X}, f(\mathbf{X}), \mathbf{x}^*, \psi\right) p(\psi) d\psi. \tag{2.15}$$

In general the integral on the right cannot be computed analytically, and we must resort to Monte Carlo methods instead.

Alternatively, one can compute a maximum likelihood estimate for the hyperparameters based on the observed data, and condition our predictions on this point estimate. Maximising the likelihood is equivalent to maximising the log likelihood since it is a monotonic transformation. Thus, the MLE of the hyperparameters is given by $\psi_{MLE} = \text{argmax}_{\psi} \log p(\psi|\mathbf{X}, f(\mathbf{X}))$. Using Bayes rule and a prior over ψ, the log likelihood

$$\log p(\psi|\mathbf{X}, f(\mathbf{X})) \propto \log p(\psi) + \log p(f(\mathbf{X})|\mathbf{X}, \psi). \tag{2.16}$$

Using a flat (improper) prior over ψ, maximising the log likelihood is equivalent to maximising $\log p(f(\mathbf{X})|\mathbf{X}, \psi)$. In general this is a non-convex optimisation problem. However, gradient based methods combined with multiple random starts are often employed to find a (locally) optimal ψ_{MLE}.

2.5.2 Bayesian Optimisation for Direct Policy Search

Bayesian optimisation (BO) is a framework for optimising black-box gradient-free functions that are expensive to evaluate. It works by iteratively building a surrogate model of the function based on observed data, and using this surrogate model to actively select the next point to query. The property of generating estimates of the uncertainty associated with any prediction makes GPs particularly suited as the surrogate model. Once the surrogate GP model has been built, BO uses an *acquisition function* to guide the search and balance exploitation (searching the space expected to have the optimum) and exploration (searching the space which has not been explored well) to choose the next point to be queried.

Concretely, at iteration n, BO fits a GP to the observed data $\mathcal{D}_{1:n-1} = \{(\mathbf{x}_i, f(\mathbf{x}_i))\}_{i=1}^{n-1}$. The next point for evaluation is then set as $\mathbf{x}_n = \text{argmax}_{\mathbf{x}}\, \alpha(\mathbf{x})$, where $\alpha(\mathbf{x})$ is the acquisition function. Two commonly used acquisition functions are *expected improvement* (EI) (Močkus, 1975; Jones et al., 1998) and *upper confidence bound* (UCB) (Cox and John, 1992, 1997). Defining $\mathbf{x}^+$ as the current optimal evaluation, i.e., $\mathbf{x}^+ = \text{argmax}_{\mathbf{x}_i} f(\mathbf{x}_i)$, EI seeks to maximise the expected improvement over the current optimum,

$$\alpha_{EI}(\mathbf{x}) = \mathbb{E}[I(\mathbf{x})], \text{ where } I(\mathbf{x}) = \max\{0, f(\mathbf{x}) - f(\mathbf{x}^+)\}. \tag{2.17}$$

By contrast, UCB does not depend on $\mathbf{x}^+$ but directly incorporates the uncertainty in the prediction by defining an upper bound:

$$\alpha_{\text{UCB}}(\mathbf{x}) = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \tag{2.18}$$

where κ controls the exploration-exploitation tradeoff. Srinivas et al. (2010) has shown that with an appropriate choice of κ, the asymptotic mean regret $\lim_{T\to\infty} \sum_T \{f(x^*) - f(x_T)\}/T$ is bounded by $\mathcal{O}(\sqrt{T(\log T)^{D+1}})$.

Information theoretic acquisition functions (for example, Predictive Entropy Search (PES) (Villemonteix et al., 2009; Hernández-Lobato et al., 2014)) select points for evaluation based on the entropy of the distribution for x^*. These seek to evaluate points that minimise the uncertainty about the location of x^* rather than focusing the search on locations where the optimal function values are expected to lie. However, these are slow to evaluate compared to EI, PI and GP-UCB as they involve expensive sampling processes which involve constructing a good approximation for the objective function based on Bochner's theorem (Bochner, 2003), thereby limiting the choice of kernels. *Fast information-theoretic Bayesian optimisation* (FITBO) (Ru et al., 2017) addresses these issues by approximating the objective function by: $f(x) = \eta - \frac{1}{7}2g(x)^2$, where the hyperparameter η represents the global maximum, and is sampled together with other hyperparameters. Unlike other information theoretic acquisition functions,

Algorithm 2 Direct Policy Search with Bayesian Optimisation

input Initial policy parameters π_1, acquisition function $\alpha(\pi)$, number of policy iterations N, initial dataset $\mathcal{D}_0 = \{\}$

1: **for** n = 1, 2, 3, ..., N **do**
2: Sample a trajectory following the current policy and observe return $R(\pi_n)$.
3: Set $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{(\pi_n, R(\pi_n))\}$ and update the GP to condition on $\mathcal{D}_n$
4: Set $\pi_{n+1} = \text{argmax}_\pi \, \alpha(\pi)$
5: **end for**

output $\pi^* = \text{argmax}_{\pi_i} \, R(\pi_i) \qquad i = 1, 2, ..., N$

FITBO seeks to reduce uncertainty about the global optimum $f(\mathbf{x}^*)$ by selecting the $\mathbf{x}$ that minimises the entropy of the distribution $p(f(\mathbf{x}^*) \mid \mathbf{x}, \mathcal{D}_{1:n})$:

$$\alpha_{\text{FITBO}}(\mathbf{x} \mid \mathcal{D}_{1:n}) = H[p(f(\mathbf{x}) \mid \mathbf{x}, \mathcal{D}_{1:n})]$$
$$- \mathbb{E}_{p(f(\mathbf{x}^*)|\mathcal{D}_{1:n})}\big[H[p(f(\mathbf{x}) \mid f(\mathbf{x}^*), \mathbf{x}, \mathcal{D}_{1:n})]\big],$$

which cannot be computed analytically, but can be approximated efficiently.

BO mainly minimises simple regret: The acquisition function suggests the next point $\mathbf{x}_i$ for evaluation at each timestep, but then the algorithm suggests what it believes to be the optimal point $\hat{\mathbf{x}}_i^*$, and the regret is defined as $\sum_{i=1}^N f(\mathbf{x}^*) - f(\hat{\mathbf{x}}_i^*)$. This is different from a bandit setting where the cumulative regret is defined as $\sum_{i=1}^N f(\mathbf{x}^*) - f(\mathbf{x}_i)$. Krause and Ong (2011) show that the UCB acquisition function is also a viable strategy to minimise cumulative regret in a contextual GP bandit setting, where selection of $\mathbf{x}_i$ conditions on some observed context.

We illustrate how BO can be used for direct policy search in Algorithm 2.

2.5.3 Bayesian Quadrature

Bayesian quadrature (O'Hagan, 1991; Rasmussen and Ghahramani, 2003) is a sample-efficient technique for computing integrals of the form $\bar{f} = \int f(\mathbf{x}) p(\mathbf{x}) d\mathbf{x}$, where $p(\mathbf{x})$ is a probability distribution. Using GP regression to compute the prediction for any $f(\mathbf{x})$ given some observed data, $\bar{f}$ is a Gaussian whose mean and variance can be computed analytically for particular choices of the covariance function and $p(\mathbf{x})$ (Briol et al.,

2015). If no analytical solution exists, we can approximate the mean and variance via Monte Carlo quadrature by sampling the predictions of various $f(\mathbf{x})$.

Given some observed data $\mathcal{D}$, we can also devise acquisition functions for BQ to actively select the next point $\mathbf{x}^*$ for evaluation. A natural objective here is to select $\mathbf{x}$ that minimises the posterior variance of $\bar{f}$ after observing $f(\mathbf{x})$ (Osborne et al., 2012),

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \, \mathbb{V}(\bar{f}|\mathcal{D}, \mathbf{x}). \tag{2.19}$$

Since $\mathbb{V}(\bar{f}|\mathcal{D}, \mathbf{x})$ is a linear combination of the predictive variances $\mathbb{V}(f(\mathbf{x})|\mathcal{D}, \mathbf{x})$ (2.14b), which do not depend on $f(\mathbf{x})$, $\mathbb{V}(\bar{f}|\mathcal{D}, \mathbf{x})$ does not depend on $f(\mathbf{x})$ either and is thus computationally feasible to evaluate. *Uncertainty sampling* (Settles, 2010) is an alternative acquisition function that chooses the $\mathbf{x}^*$ with the maximum posterior variance:

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \, \mathbb{V}(f(\mathbf{x})|\mathcal{D}). \tag{2.20}$$

Although simple and computationally cheap, it is not the same as reducing uncertainty about $\bar{f}$ since evaluating the point with the highest prediction uncertainty does not necessarily lead to the maximum reduction in the uncertainty of the estimate of the integral.

Monte Carlo (MC) quadrature simply samples $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$ from $p(\mathbf{x})$ and estimates the integral as $\bar{f} \approx \frac{1}{N} \sum_{i=1}^{N} f(\mathbf{x}_i)$. This typically requires a large N and so is less sample efficient than BQ: it should only be used if f is cheap to evaluate. The many merits of BQ over MC, both philosophically and practically, are discussed by O'Hagan (1987) and Hennig et al. (2015).

Chapter 3

Alternating Optimisation and Quadrature

In this chapter, we consider the problem of learning a policy that is robust to significant rare events. We assume that we have access to a simulator during learning, where the environment variable is controllable. Although trials in a simulator are cheaper and safer than physical trials, the computational cost of each simulated trial can still be high. The challenge then is to develop algorithms that are sample efficient, i.e., that minimise the number of such trials. In such settings, Bayesian optimisation (BO) is a sample-efficient approach that has been successfully applied to RL in multiple domains (Lizotte et al., 2007; Martinez-Cantin et al., 2007, 2009; Cully et al., 2015; Calandra et al., 2015; Pautrat et al., 2018).

We present an approach called *alternating optimisation and quadrature* (ALOQ) specifically aimed at learning policies that are robust to these rare events while remaining sample efficient. The main idea is to *actively* select the environment variables (instead of randomly sampling them) in a simulator. While a naïve application of BO for policy search would use a Gaussian Process (GP) to model the returns as a function of the policy only, ALOQ uses a GP to model returns as a function of *both* the policy and the environment variables. Then, it alternately uses BO and *Bayesian quadrature* (BQ) (O'Hagan, 1991; Rasmussen and Ghahramani, 2003) to select a

This chapter is based on material from Paul et al. (2018) which introduced ALOQ.

policy and environment setting, respectively, to evaluate. A scheme that significantly oversamples rare events risks finding a policy that achieves high performance only when a rare event occurs, but fails to perform well under the more common events. Thus, to find the optimal policy an algorithm needs to strike a proper balance between oversampling the rare events while undersampling the non-rare events. ALOQ's active selection of the environment variable does this automatically.

A simulator is only an imperfect representation of reality and policies that are exclusively learnt in simulation can exploit badly modelled aspects of reality and end up performing significantly worse on the physical system. This poses the challenge of bridging the *reality gap* (Jakobi et al., 1995; Jakobi, 1997; Koos et al., 2013), i.e., transferring policies from the simulator to the real system without any significant downgrade in performance. To address this, we present an extension to ALOQ, called *transferable ALOQ* (TALOQ), that automatically trades off the cost and accuracy of running a trial on the simulator against running it on the physical system.

We first apply ALOQ to a number of primarily simulated problems, where the reality gap does not pose any significant challenge. Our results demonstrate that ALOQ learns better and faster than multiple baselines. Next, we demonstrate that TALOQ can use an imperfect simulator to learn policies that bridge the reality gap, while requiring significantly fewer trials than if we were to train with ALOQ solely on the physical system.

3.1 Problem Setting

We assume access to a computationally expensive simulator that takes as input a policy Π_π and environment variable θ and produces as output the return $R(\pi, \theta)$. Here both π and θ belong to some compact sets in $\mathbb{R}^{d_\pi}$ and $\mathbb{R}^{d_\theta}$, respectively. We assume that this simulator is highly accurate and poses little to no reality gap and thus policies learnt on this simulator transfer well to the physical system.

We also assume access to $p(\theta)$, the probability distribution over θ. $p(\theta)$ may be known a priori, or an approximation of it may be available. For example, this could be based on the knowledge of some human expert, or as in the case of a mobile robot it could be the robot's belief about its distance from potential obstacles. While the inferred distribution for $p(\theta)$ based on limited trials may only be a rough approximation of the true distribution, our experiments show that taking account of this approximate distribution can lead to better results than running policy search on some point estimate of θ.

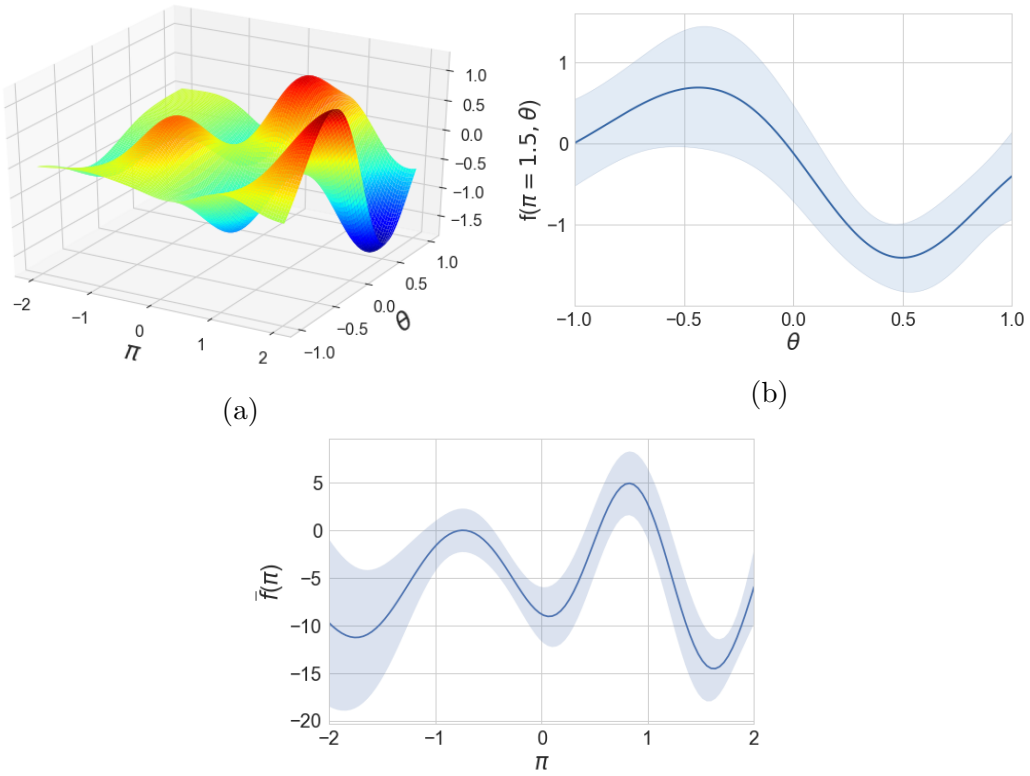
Our objective is to find an optimal policy parameters π^* as given in Eq (2.2). Since obtaining a sample of $R(\pi, \theta)$ from the simulator is expensive, Monte Carlo policy evaluation is unfeasible as it would require far too many samples. However, BQ is well suited for policy evaluation in this setting, while BO can be a sample efficient framework for finding the optimal policy.

In Section 3.3, we present a method for a setting in which we relax the assumption that the simulator is accurate. Instead, we assume that the simulator is imperfect but still useful enough that running trials on it is informative about the performance on the physical system. We assume that during training θ can be controlled on the physical system, which is not very restrictive in practice. For example, a broken joint in a robot can be emulated by disabling or limiting the actuator for that joint.

In this case, we model the return $R = R(\pi, \theta, \delta)$, where $\delta \in \{0, 1\}$ is an indicator function that denotes whether the return is from a simulated trial or from the physical system. Our objective is to find the optimal policy π^*:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\theta[R(\pi, \theta, 1)], \tag{3.1}$$

where $\delta = 1$ since we want the policy that maximises the return on the physical system.



(a) (b) (c)

Figure 3.1: ALOQ models the return R as a function of (π, θ); (a) the predicted mean based on some observed data; (b) the predicted return of $\pi = 1.5$ for different θ, together with the uncertainty associated with them; (c) given $p(\theta)$, ALOQ marginalises out θ and computes $J(\pi)$ and its associated uncertainty, which is used to actively select π.

3.2 ALOQ

The key idea behind ALOQ is to model the return as a GP with inputs (π, θ), and use BO to optimise the policy while using BQ to marginalise out the effect of the environment variable on returns. This is illustrated in Figure 3.1.

Concretely, at each iteration l, ALOQ fits a GP to dataset $\mathcal{D}_{1:l-1} = \{((\pi_i, \theta_i), R(\pi_i, \theta_i))\}_{i=1}^{l-1}$. We emphasise again that both π and θ are inputs to the GP. Next, it uses a modified UCB acquisition function:

$$\alpha_{\text{ALOQ}}(\pi) = \mu(J(\pi) \mid \mathcal{D}_{1:l-1}) + \kappa\sigma(J(\pi) \mid \mathcal{D}_{1:l-1}), \tag{3.2}$$

to set $\pi_l = \text{argmax}_\pi \, \alpha_{\text{ALOQ}}(\pi)$. For discrete θ with support $\{\theta_1, \theta_2, \ldots, \theta_{N_\theta}\}$, the

Algorithm 3 ALOQ

input A simulator that outputs $R(\pi, \theta)$, initial dataset $\mathcal{D}_{1:l}$, the maximum number of simulator calls L, and a GP prior.

1: **for** $n = l+1, l+3, ..., L-1$ **do**
2: Update the beta warping parameters and transform the inputs.
3: Update the GP to condition on the (transformed) dataset $\mathcal{D}_{1:n-1}$
4: Use (3.3) to estimate $p(J(\pi)|\mathcal{D}_{1:n-1})$
5: Use the BO acquisition function (3.2) to select $\pi_n = \text{argmax}_\pi \, \alpha_{\text{ALOQ}}(\pi)$
6: Use the BQ acquisition function (3.4) to select $\theta_n|\pi_n$
7: Perform a simulator call with (π_n, θ_n) to obtain $R(\pi_n, \theta_n)$ and update $\mathcal{D}_{1:n-1}$ to $\mathcal{D}_{1:n}$
8: Find $\hat{\pi}^* = \text{argmax}_{\pi_i} \, J(\pi_i)|\mathcal{D}_{1:n}$ and $\theta^*|\hat{\pi}^*$ using the BQ acquisition function (3.4).
9: Perform a simulator call with $(\hat{\pi}^*, \theta^*)$ to obtain $f(\hat{\pi}^*, \theta^*)$ and update $\mathcal{D}_{1:n}$ to $\mathcal{D}_{1:n+1}$
10: **end for**

output $\pi^* = \text{argmax}_{\pi_i} \, J(\pi_i) \mid \mathcal{D}_{1:L} \quad i = 1, 2, ..., L$

estimate of the mean μ and variance σ^2 for $J(\pi) \mid \mathcal{D}_{1:l-1}$ is:

$$\mu = \frac{1}{N_\theta} \sum_{i=1}^{N_\theta} \mathbb{E}[R(\pi, \theta_i)|\mathcal{D}_{1:l-1}] \tag{3.3a}$$

$$\sigma^2 = \frac{1}{N_\theta^2} \sum_{i=1}^{N_\theta} \sum_{j=1}^{N_\theta} \text{Cov}[R(\pi, \theta_i)|\mathcal{D}_{1:l-1}, R(\pi, \theta_j)|\mathcal{D}_{1:l-1}], \tag{3.3b}$$

where $R(\pi, \theta)$ is the prediction from the GP with mean and covariance computed using (2.14). For continuous θ, we apply Monte Carlo quadrature. Although this requires sampling a large number of θ and evaluating the corresponding $R(\pi, \theta) \mid \mathcal{D}_{1:l-1}$, it is feasible since we evaluate $R(\pi, \theta) \mid \mathcal{D}_{1:l-1}$, not from the expensive simulator, but from the computationally cheaper GP. This can be viewed as performing policy evaluation in our approach, i.e., estimating the expected return of a given π after marginalising out the effect of θ.

Although it is possible to define an EI-based acquisition function: $\alpha = \mathbb{E}_{J(\pi)|\mathcal{D}_{1:l-1}}[I(\pi)]$, where $I(\pi) = \max\{0, J(\pi) - J(\pi^+)\}$ (2.17), as an alternative to the UCB based acquisition function, it is prohibitively expensive to compute in practice. The stochastic $J(\pi^+) \mid \mathcal{D}_{1:l-1}$ renders this analytically intractable. Approximating it using Monte

Carlo sampling would require performing predictions on $(l-1) \times N_\theta$ points, i.e., all the $(l-1)$ observed π's paired with all the N_θ possible settings of the environment variable, which is infeasible even for moderate $(l-1)$ as the computational complexity of GP predictions scales quadratically with the number of predictions.

Once π_l is chosen, ALOQ uses a BQ acquisition function to select θ_l. Since the presence of SREs leads to high variance in the returns associated with any given policy, it is critical to minimise the uncertainty associated with our estimate of the expected return. Hence, ALOQ selects $\theta_l \mid \pi_l$ by minimising the posterior variance of $J(\pi_l)$:

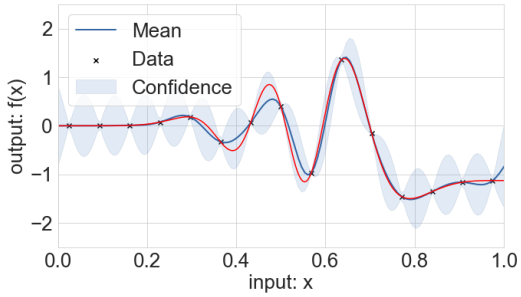
$$\theta_l|\pi_l = \operatorname*{argmin}_\theta \mathbb{V}(J(\pi_l)|\mathcal{D}_{1:l-1}, \pi_l, \theta). \tag{3.4}$$

As an alternative, we also tried uncertainty sampling in our experiments. Unsurprisingly, it performed worse since it is not as good at reducing the uncertainty associated with the expected return of a policy, as discussed in Section 2.5.3.

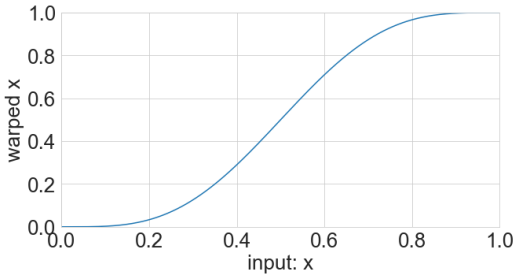
After (π_l, θ_l) is selected, ALOQ evaluates it on the simulator and updates the GP with the new datapoint $((\pi_l, \theta_l), R(\pi_l, \theta_l))$. Our estimate of π^* is thus:

$$\hat{\pi}^* = \operatorname*{argmax}_\pi \mathbb{E}[J(\pi)|\mathcal{D}_{1:l}]. \tag{3.5}$$

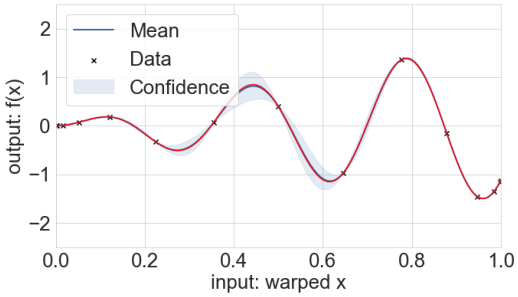
Beta Warping: Although the approach described so far actively selects π and θ through BO and BQ, it is unlikely to perform well in practice. A key observation is that the presence of SREs, which we seek to address with ALOQ, implies that the scale of the returns R varies considerably, e.g., returns in case of collision versus no collision. This nonstationarity cannot be modelled with our stationary kernel. While using a non-stationary kernel could address the issue, this would require careful consideration in the choice of the kernel, which might be problem specific. Instead, we work with a standard squared exponential kernel, but transform the inputs to ensure stationarity of R. In particular, we employ *beta warping*, i.e., we transform the inputs using beta CDFs with parameters (α, β) (Snoek et al., 2014). The CDF of the beta distribution



(a) Fitting a GP to the unwarped inputs



(b) Warping function



(c) GP fitted to the warped inputs

Figure 3.2: Effect of beta warping: (a) A GP (blue line) is fitted to the observations (marked with x) from the original function (red line) with unwarped inputs. Note how it struggles to model the function because of the non-stationary nature. (b) The warping function $\beta(4, 4)$ used to transform the input space. (c) The GP fitted to the transformed inputs is able to model the function much better.

on the support $0 < x < 1$ is given by:

$$\text{BetaCDF}(x, \alpha, \beta) = \int_0^x \frac{u^{\alpha-1}(1-u)^{\beta-1}}{B(\alpha, \beta)} \mathrm{d}u, \tag{3.6}$$

where $B(\alpha, \beta)$ is the beta function. While a transformation with a beta CDF cannot convert any arbitrary function to a stationary one, it is particularly suitable for our purpose as it can model a variety of warpings based on the settings of only two parameters (α, β). An example of such a transformation is provided in Figure 3.2. ALOQ transforms each dimension of π and θ independently and treats the corresponding (α, β) as hyperparameters. While this adds two hyperparameters – α, β, tuning these pose no additional issues as they can be learned together with the kernel hyperparameters. In the rest of this chapter, we assume that we are working with the transformed inputs.

Intensification: While the resulting algorithm should be able to cope with SREs, the $\hat{\pi}^*$ that it returns at each iteration may still be poor, since our BQ evaluation of $J(\pi)$ leads to a noisy approximation of the true expected return. This is particularly problematic in high dimensional settings. To address this, *intensification* (Bartz-Beielstein et al., 2005; Hutter et al., 2009), i.e., re-evaluation of selected policies in the simulator, is essential. Therefore, ALOQ performs two simulator calls at each timestep. In the first evaluation, (π_l, θ_l) is selected via the BO/BQ scheme described above. In the second stage, $(\hat{\pi}^*, \theta^*)$ is evaluated, where $\hat{\pi}^* \in \pi_{1:l}$ is selected using (3.5) and $\theta^*|\hat{\pi}^*$ using the BQ acquisition function (3.4).

3.3 TALOQ

ALOQ assumes access to a reliable simulator. In this section, we consider the setting where this assumption does not hold and instead there exists a reality gap between the simulator and the physical system. In this case, at each iteration we face the additional choice to perform the evaluation on the simulator or on the physical system, which is even more expensive. We present a variant of ALOQ called *transferable ALOQ* (TALOQ) that addresses this setting.

3.3.1 BO with Reality Gap

Before presenting the full TALOQ method, we first formalise the problem setting and describe a simplified BO approach for coping with the reality gap but without considering environment variables. We extend this method to TALOQ in Section 3.3.2.

Let $\delta \in \{0, 1\}$ denote the simulator or physical system respectively. Assume that at each iteration we can choose to query the simulator or the physical system to directly obtain the expected return $J(\pi, \delta)$ for an input π. Our objective is to find $\pi^* = \text{argmax}_\pi J(\pi, \delta = 1)$. In this setting we can model J as a GP with inputs (π, δ).

Observing the return of any π in the simulator gives us some information about the corresponding return on the physical system. Since the lengthscale for any dimension encodes the similarity of the function values along that dimension, we can use the lengthscale for δ to quantify how well the simulator reflects reality. If the reality gap is small the lengthscale should be long, and vice versa.

We develop a two-stage algorithm: at iteration l, first select π_l for evaluation, and then $\delta_l|\pi_l$. π_l can be selected by maximising the UCB acquisition function:

$$\alpha(\pi) = \mu(J(\pi, \delta = 1)) + \kappa\sigma(J(\pi, \delta = 1)), \tag{3.7}$$

where $\delta = 1$ since our objective is to maximise the performance on the physical system.

We define the relative reduction in uncertainty by evaluating π_l in simulation vs. on the physical system:

$$\gamma = \frac{\mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}] - \mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}, \pi_l, \delta_l = 0]}{\mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}] - \mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}, \pi_l, \delta_l = 1]}, \tag{3.8}$$

and set $\delta_l = 0$ if $\gamma > k$, and $\delta_l = 1$ otherwise. Here $k \in [0, 1]$ is a hyperparameter that should be set based on the relative cost of running physical trials against simulated trials – a large k encourages more physical evaluations, and vice versa. This formulation of γ as a ratio of the relative reduction in variance ensures that it is less problem dependent than a formulation with absolute reduction.

We illustrate this with an example in Figure 3.3. Let $J_\alpha(\pi, \delta) = (1 - \alpha\mathbb{1}_{\delta=0}) \sin \pi$. Here $\alpha \in [0, 1]$ is a proxy for the reality gap: $\alpha = 1$ corresponds to an uninformative simulator, while $\alpha = 0$ to a perfect simulator with no reality gap. Note that $J_\alpha(\pi, 1)$ is independent of α and hence labelled $J_-(\pi, 1)$ in Figure 3.3a. Let $J_\alpha(2, 0)$ and $J_\alpha(2, 1)$ be observed. For each setting of α, we can fit a GP to these two observed points and check how the learnt lengthscale varies with α. This is shown in Figure 3.3b. Note how larger reality gap corresponds to shorter lengthscale. Finally, lets assume that our acquisition function (3.8) suggests π_{eval} for evaluation. Figure 3.3c shows how γ varies with α (and hence the lengthscale for δ) for three such π_{eval}. This shows how

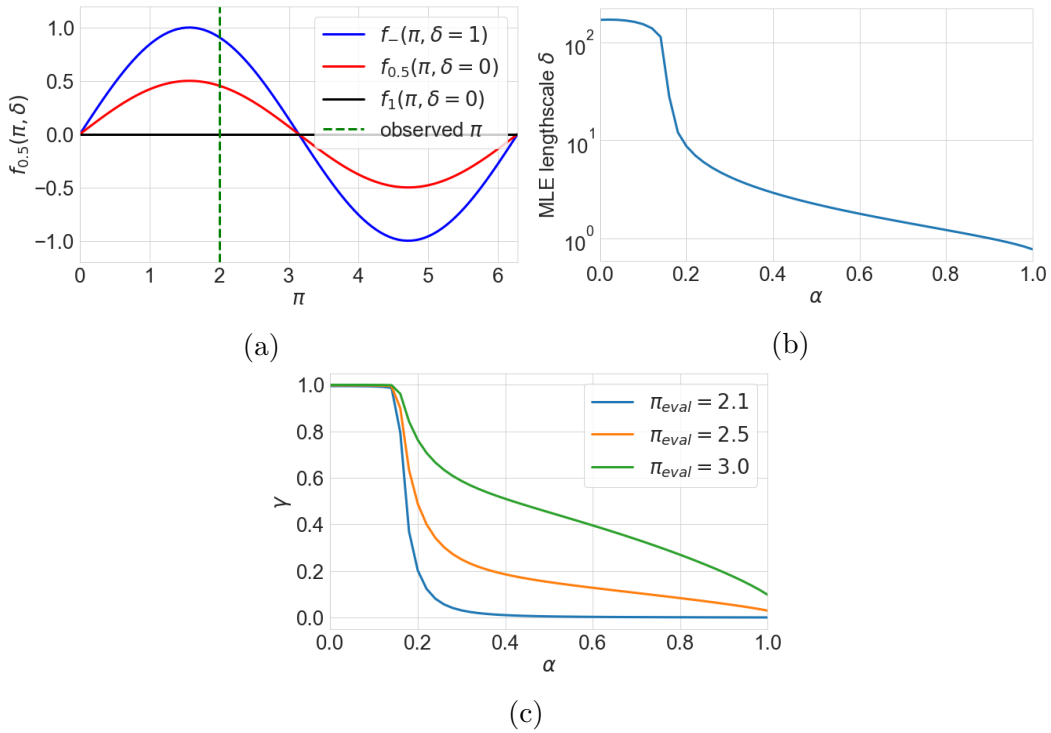


Figure 3.3: An illustrative example: (a) true and simulated function values for different α; (b) the MLE of the lengthscale for δ wrt α; (c) γ for different settings of α for different π_{eval}.

the relative reduction in uncertainty drops as the reality gap increases, but this drop is much sharper for π_{eval} close to already observed points. Thus our approach has a tendency to make use of the simulator more for unexplored regions, while using reality to fine tune in well explored regions.

3.3.2 Complete TALOQ Method

In our setting with environment variables, our objective is to find $\pi^* = \text{argmax}_\pi J(\pi, 1)$ where $J(\pi, \delta = 1) = \mathbb{E}_\theta[R(\pi, \theta, \delta = 1)]$, and we can only evaluate $R(\pi, \theta, \delta)$. To address this, TALOQ combines the above selection method for $\delta|\pi$ with ALOQ.

First, we model the return as a GP with three inputs (π, θ, δ). Then, at iteration l, π_l is selected with the acquisition function presented in (3.2) with the additional condition that $\delta = 1$. Next, we modify the relative reduction in uncertainty for our

Algorithm 4 TALOQ

input A simulator and physical system that outputs $R(\pi, \theta)$, initial dataset $\mathcal{D}_{1:l}$, the maximum number of trials evaluations L, and a GP prior.

for $n = l+1, l+3, ..., L-1$ **do**
  Update the beta warping parameters and transform the inputs.
  Update the GP to condition on the (transformed) dataset $\mathcal{D}_{1:n-1}$
  Use (3.3) to estimate $p(J(\pi, 1)|\mathcal{D}_{1:n-1})$
  Use the UCB acquisition function with $\delta = 1$ to select π_n
  Compute γ as per (3.9) and set $\delta_n = 1$ if $\gamma > k$, and $\delta_n = 0$ otherwise.
  Use the BQ acquisition function (3.10) to select $\theta_n|\pi_n, \delta_n$
  Perform an evaluation with $(\pi_n, \theta_n, \delta_n)$ to obtain $R(\pi_n, \theta_n, \delta_n)$ and update $\mathcal{D}_{1:n-1}$ to $\mathcal{D}_{1:n}$
  Find $\hat{\pi}^* = \text{argmax}_{\pi_i} J(\pi_i, 1)|\mathcal{D}_{1:n}$
  Compute γ as per (3.9) and set $\delta^* = 1$ if $\gamma > k$, and $\delta^* = 0$ otherwise.
  Use the BQ acquisition function (3.10) to select $\theta^*|\hat{\pi}^*, \delta^*$
  Perform a second evaluation with $(\hat{\pi}^*, \theta^*, \delta^*)$ to obtain $R(\hat{\pi}^*, \theta^*, \delta^*)$ and update $\mathcal{D}_{1:n}$ to $\mathcal{D}_{1:n+1}$
end for

output $\pi^* = \text{argmax}_{\pi_i} J(\pi_i, 1) \mid \mathcal{D}_{1:L} \quad i = 1, 2, ..., L$

acquisition function for $\delta_l|\pi_l$ to take into account that θ is yet to be selected:

$$\gamma = \frac{\mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}] - \text{argmin}_\theta \, \mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}, \pi_l, \theta, \delta_l = 0]}{\mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}] - \text{argmin}_\theta \, \mathbb{V}[J(\pi_l, 1)|\mathcal{D}_{1:l-1}, \pi_l, \theta, \delta_l = 1]}, \tag{3.9}$$

and select δ_l based on the hyperparameter k as earlier. Finally, we select $\theta_l|(\pi_l, \delta_l)$ using the BQ acquisition function given in (3.4) with the slight modification that the conditioning set now includes δ_l:

$$\theta_l|\pi_l, \delta_l = \underset{\theta}{\text{argmin}} \, \mathbb{V}[J(\pi_l)|\mathcal{D}_{1:l-1}, \pi_l, \theta, \delta_l]. \tag{3.10}$$

Marco et al. (2017) propose a related acquisition function for δ for the simplified BO setting described above. However, because it is entropy based, its application to TALOQ is not straightforward, as it would require first integrating over θ. By contrast, our selection criteria for δ is easy to implement and follows the same variance reduction principle as the BQ aspect of selecting θ.

3.4 Properties of ALOQ and TALOQ

Existing convergence guarantees for BO using α_{UCB} (Srinivas et al., 2010) hold only for cases with fixed hyperparameters. Similarly, existing convergence guarantees for BQ (Kanagawa et al., 2016; Briol et al., 2015) apply only to methods that do not actively select points. Since our methods rely on beta warping, active selection of quadrature points (whether for simulated or real evaluation in TALOQ), and the hyperparameters of the GP are updated after each iteration, these convergence guarantees for BO and BQ cannot be applied to ALOQ and TALOQ. However, we expect such active selection to only improve the rate of convergence of our algorithms over passive versions, and our empirical results in Section 3.5.2 show that in practice our methods efficiently optimise policies in the presence of SREs across a variety of tasks.

The computational complexity of ALOQ and TALOQ is dominated by an $\mathcal{O}(l^3)$ matrix inversion, where l is the sample size of the dataset $\mathcal{D}$. This cubic scaling is common to all BO methods involving GPs. The BQ integral estimation in each iteration requires only GP predictions, which are $\mathcal{O}(l^2)$.

3.5 Experimental Results

In this section, we empirically compare ALOQ and TALOQ to a number of baselines. Details about the experimental setup, including the environments, choice of kernels, and treatment of hyperparameters amongst others are presented in the Supplementary Materials.

3.5.1 Baseline Methods

A naïve approach is to directly apply a policy search algorithm (for example, BO, REINFORCE (Williams, 1992), or Trust Region Policy Optimisation (TRPO) (Schulman et al., 2015)) while treating the variability in returns caused by the environment variable as noise.

In particular, we can apply BO directly on $J(\pi) = \mathbb{E}_\theta[R(\pi, \theta)]$ and attempt to estimate π^*. Formally, this approach models $J(\pi)$ as a GP with a zero mean function and a suitable covariance function $k(\pi, \pi')$. Since R is expensive to evaluate, at the lth BO iteration only one $R(\pi_l, \theta_l)$ with $\theta_l \sim p(\theta)$ is evaluated in the simulator and $J(\pi_l)$ is approximated by this one sample Monte Carlo estimate. This approach will almost surely fail since the estimates of $J(\pi)$ are extremely noisy, especially in the presence of SREs.

For policy gradient methods like REINFORCE and TRPO, this translates to sampling a batch of trajectories from the environment at each iteration, and then using the observed returns to approximate gradient ascent on the policy. In general these methods suffer from high variance in the gradient estimates, leading to slow learning (Glynn, 1990; Peters and Schaal, 2006). In our setting with SREs, this problem is compounded by the variance due to θ, which is not explicitly considered while approximating the gradient. Furthermore, if SREs are not observed due to the random sampling from $p(\theta)$, these methods can converge to a highly suboptimal policy. While these methods are not particularly sample efficient and thus not suitable for our setting, we include them in our experiments for completeness.

We also compare ALOQ to several other baselines. The complete list of baselines is 1) the *naïve* method described above with BO, REINFORCE, and TRPO as the underlying policy search algorithms; 2) the method of Williams et al. (2000), which we refer to as *WSN*. To show the importance of each component of ALOQ, we also perform experiments with ablated versions, namely: 1) *random quadrature ALOQ* (RQ-ALOQ), in which θ is sampled randomly from $p(\theta)$ instead of being chosen actively; 2) *unwarped ALOQ*, which does not perform beta warping of the inputs; and 3) *one-step ALOQ*, which does not use intensification.

		x_3			
		0.2	0.4	0.6	0.8
x_2	0.25	0.0375	0.0875	0.0875	0.0375
	0.5	0.075	0.1750	0.1750	0.075
	0.75	0.0375	0.0875	0.0875	0.0375

Table 3.1: Probability distribution of the environment variables for the Branin test function.

3.5.2 ALOQ Experiments

To evaluate ALOQ, we applied it to 1) artificial test functions, including those used by Williams et al. (2000), who consider a setting very similar to ours, but without any potential SREs, 2) a simulated robot arm control task, including a variation where $p(\theta)$ is not known a priori but must be inferred from data, and 3) a hexapod locomotion task. All plotted results are the median of 20 independent runs.

3.5.2.1 Artificial Test Functions

We begin with modified versions of the *Branin* and *Hartmann 6* test functions used by Williams et al. (2000). The modified Branin test function is a four-dimensional problem, with two dimensions treated as discrete environment variables with a total of 12 support points. The function is defined as:

$$f(\mathbf{x}) = y_b(15x_1 - 5, 15x_2) \times y_b(15x_3 - 5, 15x_4), \text{where} \tag{3.11}$$
$$y_b(u, v) = \left(v - \frac{5.1}{4\pi^2}u^2 + \frac{5}{\pi}u - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(u) + 10.$$

Here $\{x_1, x_4\} \in [0, 1]^4$, and $\{x_2, x_3\}$ are the environment variables with the joint probability distribution given in Table 3.1.

The modified Hartmann 6 test function is six-dimensional with two dimensions treated as environment variables with a total of 49 support points. It is defined as

$$f(x_1, ..., x_6) = -\log\left[\sum_{i=1}^{4} c_i \exp\left\{-\sum_{j=1}^{6} \alpha_{ij}(x_j - p_{ij})^2\right\}\right]. \tag{3.12}$$

i	$\alpha_{ij}, \quad j = 1, ..., 6$						c_i
1	10	3	17	3.5	1.7	8	1
2	0.05	10	17	0.1	8	14	1.2
3	3	3.5	1.7	10	17	8	3
4	17	8	0.05	10	0.1	14	3.2

i	$p_{ij}, \quad j = 1, ..., 6$					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Table 3.2: Coefficients for Hartman 6 test function.

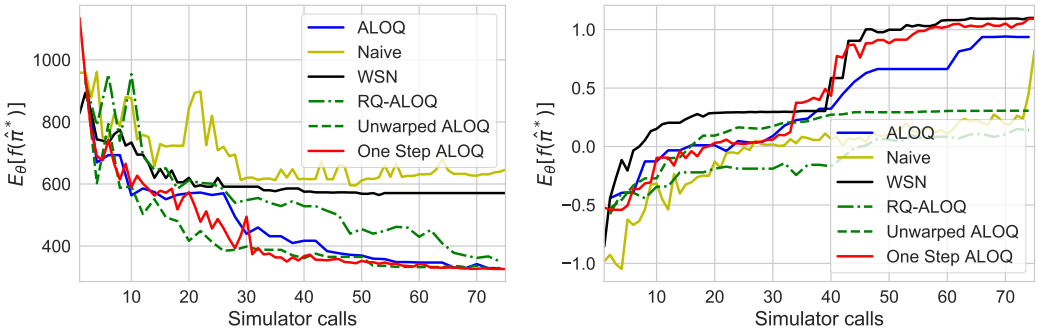
x	0.125	0.25	0.375	0.5	0.625	0.75	0.875
$p(x)$	9/128	1/8	3/16	15/64	3/16	1/8	9/128

Table 3.3: Probability distribution of the environment variables for the Hartman 6 test function.

The coefficients c_i, α_{ij}, and p_{ij} are given in Table 3.2. Here $\{x_1, x_2, x_4, x_6\} \in [0, 1]^4$, and x_3 and x_5 are treated as the environment variables, which are assumed to be distributed independently as given in Table 3.3.

The performance of the algorithms on the two functions is presented in Figure 3.4. In the Branin function, ALOQ, RQ-ALOQ, unwarped ALOQ, and one-step ALOQ all substantially outperform WSN. WSN performs better in the Hartmann 6 function as it does not get stuck in a local maximum. However, it still cannot outperform one-step ALOQ. Note that ALOQ slightly underperforms one-step ALOQ. This is not surprising: since the problem does not have SREs, the intensification procedure used by ALOQ does not yield any significant benefit.

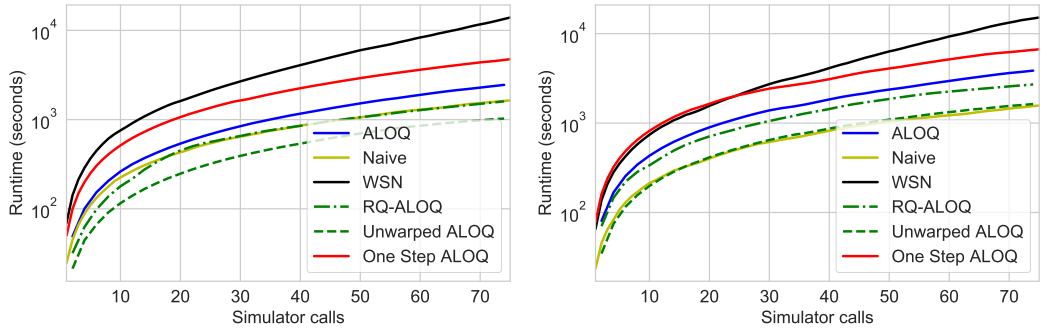
Figure 3.5 plots in log scale the total runtime of each algorithm. WSN takes significantly longer than ALOQ or the other baselines, and shows a clear increasing trend. The slow runtime of WSN is as expected for the reasons mentioned in Section 6.1. However, its failure to outperform RQ-ALOQ is surprising as these are the test problems Williams et al. use in their own evaluation. However, they never compared WSN to these (or any other) baselines. Consequently, they never validated the benefit



(a) Branin (min) - expected value of $\hat{\pi}^*$ (lower is better)

(b) Hartmann 6 (max) - expected value of $\hat{\pi}^*$ (higher is better)

Figure 3.4: Comparison of performance of all methods on the modified Branin and Hartmann 6 test functions used by Williams et al..



(a) Branin (min)

(b) Hartmann 6 (max)

Figure 3.5: Comparison of runtime of all methods on the modified Branin and Hartmann 6 test function used by Williams et al..

of modelling θ explicitly, much less selecting it actively. In retrospect, these results make sense because the function is not characterised by significant rare events and there is no other a priori reason to predict that simpler methods will fail.

These results underscore the fact that a meaningful evaluation must include a problem with SREs, as such problems do demand more robust methods. To create such an evaluation, we formulated two test functions, F-SRE1 and F-SRE2, that are

characterised by significant rare events. For $\pi \in [-2, 2]$, F-SRE1 is defined as:

$$\begin{aligned} f_{F-SRE1}(\pi, \theta) =& 75\pi \exp(-\pi^2 - (4\theta + 2)^2) \\ &+ \sin(2\pi) \sin(2.7\theta), \\ \text{with } p(\theta = \theta_j) =& \begin{cases} 0.47\% & \text{for } \theta_j = -1.00, -0.95, ..., 0.00 \\ 1.0\% & \text{for } \theta_j = 0.05, 0.10, ..., 4.50. \end{cases} \end{aligned} \tag{3.13}$$

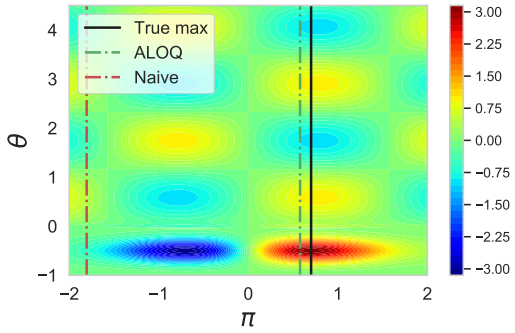
For $\pi \in [-2, 2]$, F-SRE2 is defined as:

$$\begin{aligned} f_{F-SRE2}(\pi, \theta) =& \sin^2 \pi + 2 \cos \theta \\ &+ 200 \cos(2\pi)(0.2 - \min(0.2, |\theta|)), \\ \text{with } p(\theta = \theta_j) =& \begin{cases} 1.2\% & \text{for } \theta_j = -1.00, -0.98..., -0.22 \\ 0.2\% & \text{for } \theta_j = -0.20, -0.18, ..., 0.20 \\ 1.2\% & \text{for } \theta_j = 0.22, 0.24..., 1.00. \end{cases} \end{aligned} \tag{3.14}$$

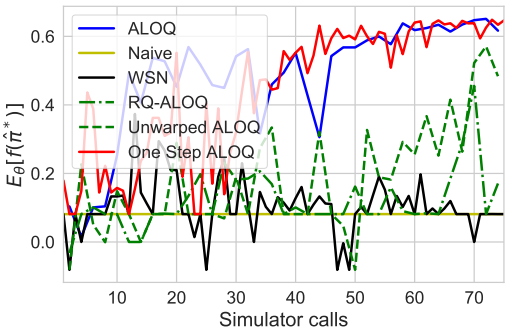
Figures 3.6a and 3.7a shows the contour plots of these two functions. Both functions have a narrow band of θ that corresponds to the SRE regions, i.e., the scale of the rewards is much larger in these regions. In F-SRE1 this is $-1 < \theta < 0$ while in F-SRE2 this is $-0.2 < \theta < 0.2$. We downscaled the region corresponding to the SRE by a factor of 10 to make the plots more readable. The final learned policy, i.e., $\hat{\pi}^*$, of ALOQ and the Naïve approach is shown as a vertical line, along with π^* (the true maximum). These lines illustrate that not accounting for SREs properly can lead to learning significantly suboptimal policies.

Figures 3.6b and 3.7b, which plot the performance of all methods for the two functions, show that ALOQ substantially outperforms all the other algorithms except for one-step ALOQ (note that both WSN and the naïve approach fail completely in these settings). As expected, intensification does not yield any additional benefit in this low dimensional problem. However, our experiments on the robotics tasks presented next show that intensification is crucial for success in higher dimensional problems.

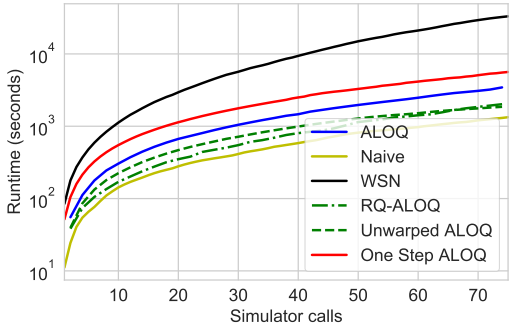
The total runtime is presented in Figures 3.6c and 3.7c (note the log scale). Again WSN is significantly slower than all other methods. In fact, it was not computationally feasible to run WSN beyond 100 data points for F-SRE2.



(a) F-SRE1



(b) F-SRE1 - expected value of $\hat{\pi}^*$



(c) F-SRE1

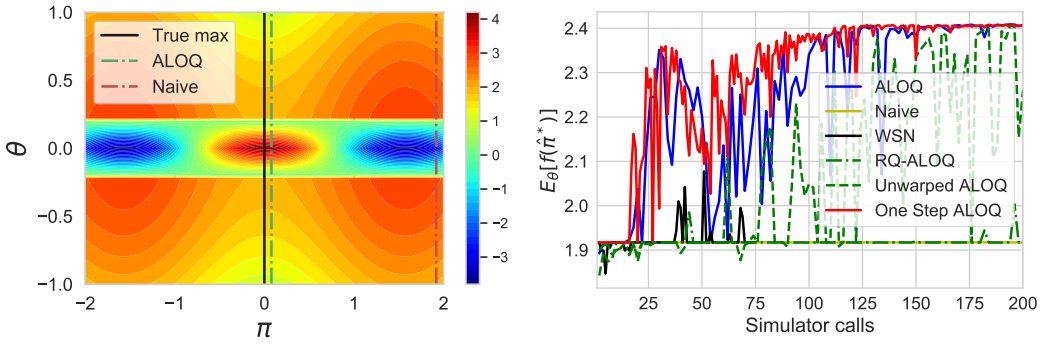
Figure 3.6: Contour plot of F-SRE1 (values in SRE region have been reduced by a factor of 10), and comparison of performance (higher is better) and runtimes of all methods on F-SRE1.

3.5.2.2 Robotic Arm Simulator

In the next experiment, we evaluate ALOQ's performance on a robot control problem implemented in a kinematic simulator. The goal is to configure each of the three controllable joints of a robot arm such that the tip of the arm gets as close as possible to a predefined target point. The configuration of the robot arm is determined by three joint angles, each of which is normalised to lie in $[0, 1]$. The arm has a reach of $[-0.54, 0.89]$ on the x-axis. We set $\kappa = 1.5$ for all three experiments in this section based on grid search.

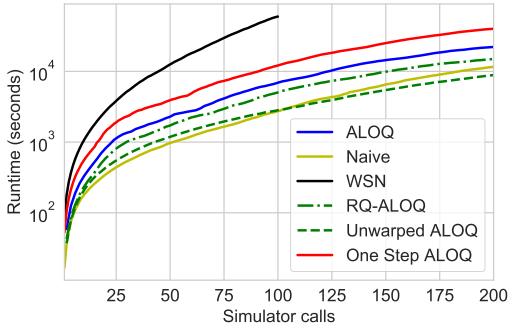
Collision Avoidance

In the first setting, we assume that the robotic arm is part of a mobile robot that has localised itself near the target. However, due to localisation errors, there is a small



(a) F-SRE2

(b) F-SRE2 - expected value of $\hat{\pi}^*$



(c) F-SRE2

Figure 3.7: Contour plot of F-SRE2 (values in SRE region have been reduced by a factor of 10), and comparison of performance (higher is better) and runtimes of all methods on F-SRE2.

possibility that it is near a wall and some joint angles may lead to the arm colliding with the wall and incurring a large cost. Minimising cost entails getting as close to the target as possible while avoiding the region where the wall may be present.

The environment variable θ in this setting is the location of the wall. It is discrete with 20 support points logarithmically distributed in $[-0.2, 0.14]$. The first 10 locations closest to the arm have a probability of 0.75% each, the next 7 have 1.5% each, and final 3 locations furthest from the arm have a probability of 27.33% each. The cost incurred is 100 times the distance between the target and the final position of the end effector, and collisions with the wall yield a cost of 2500.

Figures 3.8a and 3.8b show the expected cost (lower is better) of the arm configurations after each timestep for each method. ALOQ, unwarped ALOQ, and RQ-ALOQ

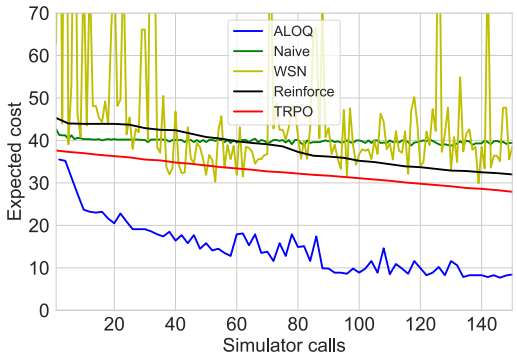
greatly outperform the other baselines. REINFORCE and TRPO are not sample efficient, exhibiting a slow rate of improvement, while WSN fails to converge at all.

Figure 3.8c shows the learned arm configurations, as well as the policy that would be learned by ALOQ if there was no wall (No Wall). The shaded region represents the possible locations of the wall. This plot illustrates that ALOQ learns a policy that gets closest to the target. Furthermore, while all the BO based algorithms learn to avoid the wall, active selection of θ allows ALOQ to do so more quickly: smart quadrature allows it to more efficiently observe rare events and accurately estimate their boundary. For readability we have only presented the arm configurations for algorithms with performance comparable to ALOQ. Note that due to constraints on the joint angles, it is not possible to reach the target while avoiding the region where the wall may be present.

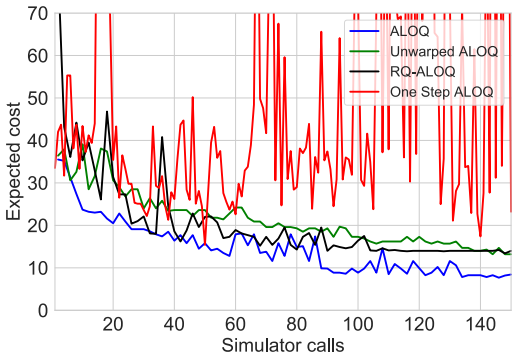
Joint Breakage

To see how ALOQ performs in settings with continuous environment variables, we consider a variation in which instead of uncertainty introduced by localisation, some settings of the first joint carry a 5% probability of it breaking which leads to a cost of 2000. Otherwise the agent incurs a cost of 100 times the distance between the target and the final position of the end effector. Minimising cost thus entails getting as close to the target as possible, while minimising the probability of the joint breaking.

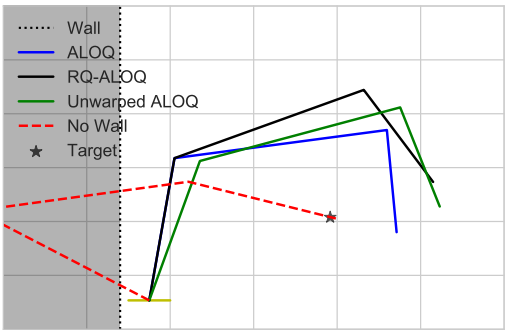
Figures 3.9a and 3.9b shows the expected cost (lower is better) of the arm configurations after each timestep for each method. Since θ is continuous in this setting, and WSN requires discrete θ, it was run on a slightly different version with θ discretised by 100 equidistant points. The results are similar to the previous experiment, except that the baselines perform worse. In particular, the Naïve baseline, WSN, and REINFORCE seem to converge to a suboptimal policy since the lack of seeing SREs often enough leads them to converge to policies which have high performance in non-SREs, but are clearly highly suboptimal when the effect of SREs is factored in.



(a) Expected costs of different $\hat{\pi}^*$ - Baselines



(b) Expected costs of different $\hat{\pi}^*$ - Ablations



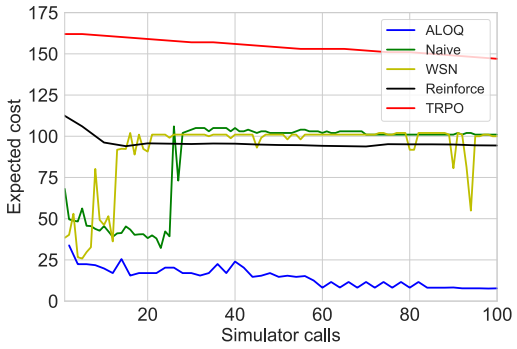
(c) Learned arm configurations

Figure 3.8: Performance and learned configurations on the robotic arm collision avoidance task.

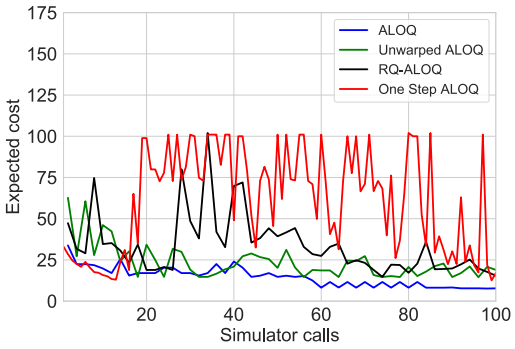
Figure 3.9c shows the learned arm configurations together with the policy that would be learned if there were no SREs (No break). The shaded region represents the joint angles that can lead to failure. This figure illustrates that ALOQ learns a qualitatively different policy than the other algorithms, one that avoids the joint angles that might lead to a breakage while still getting close to the target faster than the other methods. Again, for readability we only present the arm configurations for the most competitive algorithms.

Performance of REINFORCE and TRPO

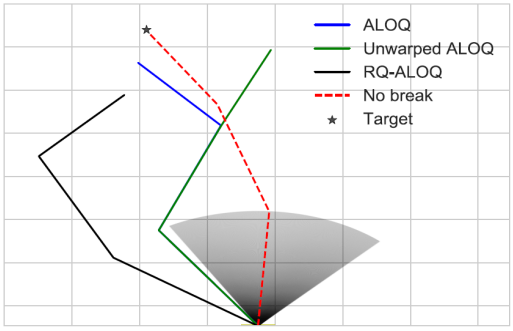
Both REINFORCE and TRPO are relatively sample inefficient. However, one question that arises is whether these methods eventually find the optimal policy. To check this, we ran these on both the Collision Avoidance and Joint Breakage settings. For



(a) Expected costs of different $\hat{\pi}^*$ - Baselines



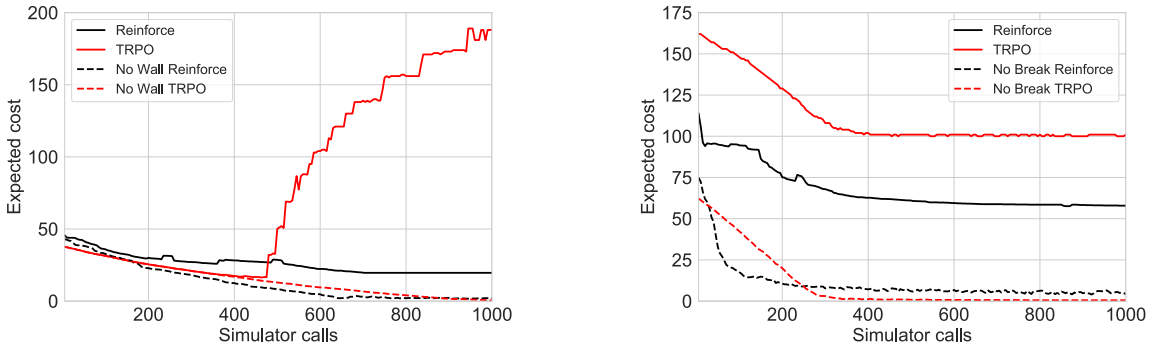
(b) Expected costs of different $\hat{\pi}^*$ - Ablations



(c) Learned arm configurations

Figure 3.9: Performance and learned configurations on the robotic arm joint breakage task.

TRPO we used a neural net with two hidden layers with 5 units each as this seemed to perform well on the task without any SREs, and had a KL constraint of 0.01 as it has been shown to perform well on wide range of tasks (Duan et al., 2016a). For REINFORCE we used a linear Gaussian policy and set the learning rate to 10^{-4}. Both methods were run for 2000 iterations with a batch size of 5 trajectories (thus a total of 10000 simulator calls). The expected cost of the arm configurations after each iteration are presented in Figure 3.10 (we only present the results up to 1000 simulator calls for readability; there is no improvement beyond what can be seen in the plot). Both baselines can solve the tasks in settings without SREs, i.e., where there is no possibility of a collision or a breakage (No Wall and No Break in the figures). However, in settings with SREs, they converge rapidly to a suboptimal policy from which they are unable recover even if run for much longer, since they do not experience the SREs



(a) Collision avoidance task

(b) Arm breakage task

Figure 3.10: Performance of REINFORCE and TRPO on the robotic arm simulator tasks.

often enough. This is especially striking in the collision avoidance task where TRPO converges to a policy that has a relatively high probability of leading to a collision. This phenomenon has been explored further in Paul et al. (2019b).

Setting with approximate $p(\theta)$

Now we consider the setting where the exact distribution $p(\theta)$ is not known a priori, but only an approximation is available. In this setting, instead of directly setting the robot arm's joint angles, we set the torque applied to each joint ($\pi \in [0, 1]^3$). The final joint angles are determined by the torque and the rigidity of the joints ($\theta \in [0.5, 1]$). Setting the torque too high can lead to the joint being damaged or broken, which incurs a large cost.

We use the simulator as a proxy for both real trials as well as the simulated trials. To simulate a set of n physical trials with a baseline policy with parameters π_b, we sample θ from $U(0.5, 1)$ and observe the return $f(\pi_b, \theta)$ and add iid Gaussian noise to them. The posterior can then be computed as $p(\theta|\mathcal{D}^b_{1:n}, \pi_b) \propto p(\theta)p(\mathcal{D}^b_{1:n}|\pi_b, \theta)$, where $\mathcal{D}^b_{1:n} = \{(\pi_b, f_1), (\pi_b, f_2), ..., (\pi_b, f_n)\}$. We can approximate this using slice sampling since both the prior and the likelihood are analytical. An alternative formulation would be to corrupt the joint angles with Gaussian noise instead of the observed returns. The posterior can still be computed in this case, but instead of using slice

sampling, we would have to make use of *approximate Bayesian computation* Rubin (1984); Tavaré et al. (1997); Pritchard et al. (1999), which would be computationally expensive.

Next use ALOQ to compute the optimal policy over this posterior ('ALOQ policy'). For comparison, we also compute the *maximum a posteriori* (MAP) estimate of θ and the corresponding optimal policy (MAP policy). To show that active selection of θ is advantageous, we also compare against the policy learned by RQ-ALOQ. To ensure that only the information gained about θ gets carried over from the physical trials to the final ALOQ/MAP policy being learned, the target for the baseline policy was different to the target for the final policy.

Since we are approximating the unknown $p(\theta)$ with a set of samples, it makes sense to keep the sample size relatively low for computational efficiency when finding the ALOQ policy (50 samples in this instance). However, to show that ALOQ is robust to this approximation, when comparing the performance of the ALOQ and MAP policies, we used a much larger sample size of 400 for the posterior distribution.

For evaluation, we drew 1000 samples of θ from the more granular posterior distribution and measured the returns of the three policies for each of the samples. The average cost incurred by the ALOQ policy (presented in Table 3.4) was 31% lower than that incurred by the MAP policy and 23.6% lower than the RQ-ALOQ policy. This is because ALOQ finds a policy that slightly underperforms the MAP policy in some cases but avoids over 95% of the SREs (cost $\geq$70 in Table 3.4) experienced by the MAP and RQ-ALOQ policies.

3.5.2.3 Hexapod Locomotion Task

As robots move from fully controlled environments to more complex ones, they have to face the inevitable risk of getting damaged. However, it may be expensive or even impossible to decommission a robot whenever any damage condition prevents it from completing its task. Hence, it is desirable to develop methods that enable robots to

	Average Cost	% Episodes in Cost Range 0-20	20-70	≥70
ALOQ Policy	19.82	61.3%	38.5%	0.2%
MAP Policy	28.76	67.1%	28.7%	4.2%
RQ-ALOQ	25.95	-	94.5%	5.5%

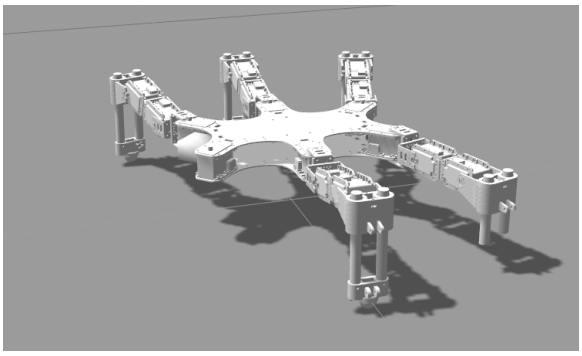
Table 3.4: Comparison of the performance of ALOQ, MAP, and RQ-ALOQ policies when $p(\theta)$ must be estimated.

recover from failure.

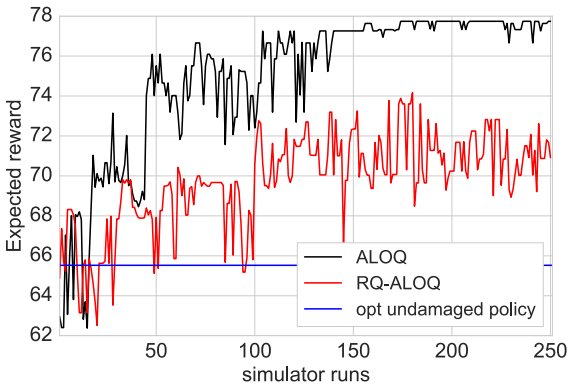
Intelligent trial and error (IT&E) (Cully et al., 2015) has been shown to recover from various damage conditions and thereby prevent catastrophic failure. Before deployment, IT&E uses the simulator to create an archive of diverse and locally high performing policies for the intact robot that are mapped to a lower dimensional *behaviour space*. If the robot becomes damaged after deployment, it uses BO to quickly find the policy in the archive that has the highest performance on the damaged robot. However, it can only respond after damage has occurred. Though it learns quickly, performance may still be poor while learning during the initial trials after damage occurs. To mitigate this effect, we propose to use ALOQ to learn in simulation the policy with the highest expected performance across the possible damage conditions. By deploying this policy, instead of the policy that is optimal for the intact robot, we can minimise in expectation the negative effects of damage in the period before IT&E has learned to recover.

We consider a hexapod locomotion task with a setup similar to that of Cully et al. (2015) to demonstrate this experimentally. The robot has six legs with three degrees of freedom each. We built a fairly accurate model of the robot which involved creating a URDF model with dynamic properties of each of the legs and the body, including their weights, and used the DART simulator for the dynamic physics simulation.[1] We also used velocity actuators. The low-level controller (or *policy*) is the same open-loop

[1] `https://dartsim.github.io`



(a) Hexapod with a shortened and a missing leg.



(b) Expected value of $\hat{\pi}^*$

Figure 3.11: Hexapod locomotion problem.

controller as in Cully and Mouret (2015) and Cully et al. (2015). The position of the first two joints of each of the six legs is controlled by a periodic function with three parameters: an offset, a phase shift, and an amplitude (we keep the frequency fixed). The position of the third joint of each leg is the opposite of the position of the second one, so that the last segment always stays vertical. This results in 36 parameters.

The objective is to cross a finish line a fixed distance from its starting point. We set the reward such that failure to cross the finish line within 5 seconds yields zero reward, while crossing the finish line gives a reward of $100 + 50v$ where v is the average velocity in m/s.

It is possible that a subset of the legs may be damaged or broken when deployed in a physical setting. For our experiments, we assume that, based on prior experience, any of the front two or back two legs can be shortened or removed with probability of 10% and 5% respectively, independent of the other legs, leading to 81 possible configurations. We excluded the middle two legs from our experiment as their failure has relatively little impact on the hexapod's movement. The configuration of the six legs acts as our environment variable. Figure 3.11a shows one such setting. The environment variable here has a discrete support with 81 states, one for each setting of the four legs with the corresponding probability for that setting. For example, the probability of the

setting {no damage, removed, shortened, no damage} is $0.85 \times 0.05 \times 0.1 \times 0.85$.

We applied ALOQ to learn the optimal policy given these damage probabilities, but restricted the search to the policies in the archive created by Cully et al. (2015). The archive consists of 12514 policies in the behaviour space created using the MAP-Elites algorithm Mouret and Clune (2015). MAP-Elites searches for the highest-performing solution for each point in the duty factor space Cully et al. (2015), i.e., the time each tip of the leg spent touching the ground. MAP-Elites also acts as a dimensionality reduction algorithm and maps the high dimensional controller/policy space (in our case 36D) to the lower dimensional behaviour space (in our case 6D). We also used this lower dimensional representation of the policies in the archive as the policy search space (π) for ALOQ. Figure 3.11b shows that ALOQ finds a policy with much higher expected reward than RQ-ALOQ. It also shows the policy that generates the maximum reward when none of the legs are damaged or broken ('opt undamaged policy').

To check if a policy learnt by ALOQ in simulation transfers successfully, we ran an experiment where we used ALOQ to learn a policy entirely in simulation and then deployed the learnt policy on a real hexapod. In order to limit the number of physical trials required to evaluate the ALOQ policy, we limited the possibility of damage to the rear two legs. The learnt policy performed better than the opt undamaged policy (which was also learnt only in simulation) on the physical robot because it optimised performance on the rare configurations that matter most for expected return (e.g., either leg shortened). However, the performance of both policies were worse than in simulation due to the reality gap. For example, in simulation the undamaged hexapod traveled more than $1m$ with both the ALOQ policy and the opt undamaged policy, while in reality it traveled no more than $0.75m$. These results underscore the need for an algorithm like TALOQ to bridge the reality gap. In the next section we present an experiment using TALOQ with a robotic arm without using MAP-Elites.

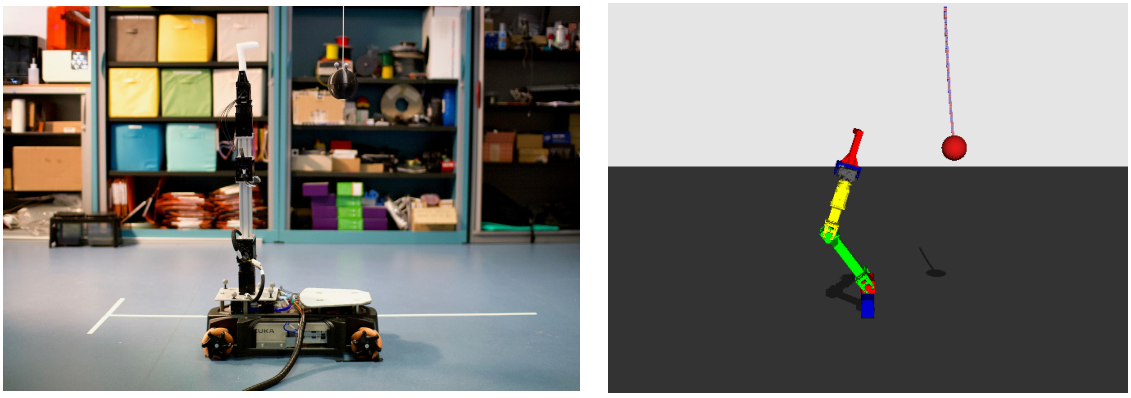
3.5.3 TALOQ Experiments

To see how well TALOQ learns transferable policies compared to ALOQ and other baselines, we applied it to a robotic arm. The arm has actuators across 4 joints that control the position of its end effector in 3D space. The objective is to strike a ball hanging from a rope with its end effector and achieve a target velocity. Since physical trials are too expensive to compare TALOQ against all the other baselines on the physical robot, we first do an extensive evaluation of TALOQ in simulation, and then compare TALOQ to ALOQ in an experiment on the physical robot. The reward function was defined as follows:

$$r = -25d_{\min} + 100\exp(-10(v_x - v_{\text{target}})^2) \tag{3.15}$$

where $d_{\min}$ is the minimum distance between the end-effector and the ball observed throught the episode, v_x is the velocity of the ball (in the x-direction) at the time of the impact with the robotic arm (if there is no impact, $v_x = 0$) and v_{target} is the target velocity. In essence, we want to hit the ball with the arm and generate a velocity of v_{target} in the x-direction. In the simulated experiments (Sec. 3.5.3.1) we set $v_{\text{target}} = 1\,m/s$, whereas in the physical experiments (Sec. 3.5.3.2) we set $v_{\text{target}} = 0.75\,m/s$. The constants were chosen to ensure that the scale of rewards varies by a few orders of magnitude so as to simulate the effect of SREs.

The policy space is 5-D consisting of 4 joint angles and a frequency parameter for the arm to oscillate between the initial configuration and the specified joint angles. As an SRE, we assume that the arm is damaged with probability 5%, making the actuator for the third joint unresponsive and stuck in the initial configuration. This can cause policies that perform well on the fully functional arm to perform poorly on the damaged arm.



(a) Physical robot (b) Simulated robot

Figure 3.12: Experimental setup for TALOQ: The objective is to learn a policy for striking the ball and achieving a particular velocity. (a) The physical setup, (b) the setup in simulation.

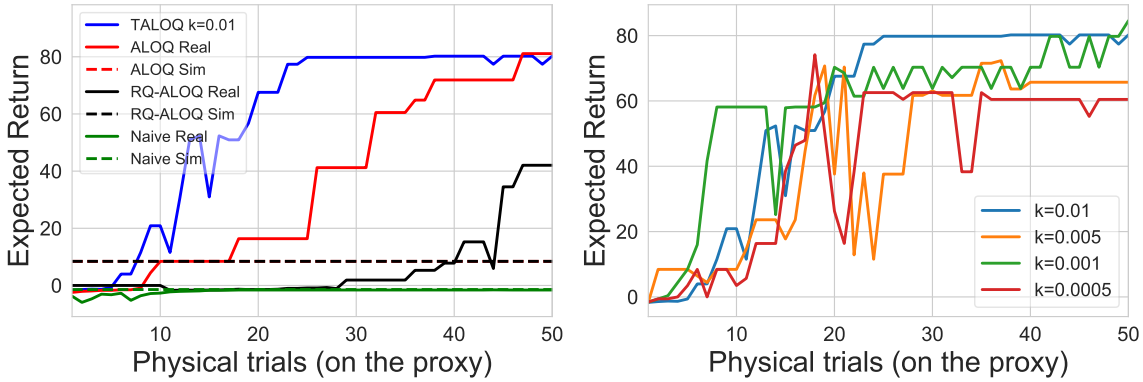
3.5.3.1 Simulated Experiments

To enable extensive evaluation of TALOQ, we devised a simulated version of the experiment. We used DART (Lee et al., 2018) to design a simulator for our experimental setup.[2] Figures 3.12a and 3.12b show the physical robot and the simulated model.

We treated the simulator for the robotic arm as a proxy for reality, and developed another version which we treated as the simulator. To create a reality gap between the two, instead of the ball hanging from the rope we modelled it as a pendulum. We also set the mass of the ball to $1kg$ compared to $60g$ in the proxy. Since simulated trials are relatively cheap, we ran 20 random replicates and after each iteration evaluated the expected return of the policy TALOQ specified as optimal.

In Figure 3.13a we compare the performance of TALOQ against three baselines: ALOQ, RQ-ALOQ, and the naïve approach. We learnt two versions of each of the baseline, one solely in simulation to check if the learnt policies transfer well, and the other solely on the proxy for reality to check that TALOQ indeed makes use of the simulated trials to improve sample efficiency on the proxy. Note that the x-axis is the number of physical trials. The policy learnt only in simulation using ALOQ (ALOQ

[2]We used the robot_dart wrapper: `https://github.com/resibots/robot_dart`.



(a) Comparison of performance (b) Effect of k on TALOQ's performance

Figure 3.13: Performance of TALOQ on the simulated experiments.

Sim)/RQ-ALOQ (RQ-ALOQ Sim)/Naïve (Naïve Sim) approach does not transfer at all to the proxy for reality, since the reality gap is quite large. Learning a policy with ALOQ exclusively in this proxy (ALOQ Real) performs better since there is no reality gap. However it is less sample efficient than TALOQ since it does not leverage the information provided by the simulator. This shows that TALOQ can effectively leverage simulated trials to improve sample efficiency on the physical system. On the other hand, the naïve approach (Naïve Real) does not learn at all due the presence of the SRE.

To see how the performance of TALOQ varies based on the choice of k in (3.9), we ran also the experiment with different values of k. The results presented in Figure 3.13b shows that performance is stable across a wide range of k, though higher values can lead to slightly faster learning.

3.5.3.2 Physical Experiments

Next we used TALOQ to learn a policy that is transferable to the real physical robot using simulated and physical trials. We compare its performance to that of the policy learnt by ALOQ only in simulation. Policies learnt by ALOQ only in simulation with a budget of 200 trials achieved a median return of 4.23 on the robot across 20 random replicates, which shows that it transfers poorly. This is unsurprising since the reality

Physical trials	Replicate # 1	2	3
1	-2.19	-6.07	-4.98
≤ 10	7.99	-6.07	35.85
≤ 20	28.18	-6.07	35.85
≤ 35	28.18	89.18	56.40
≤ 50	28.18	89.18	93.60

Table 3.5: Evolution of the expected reward on the physical robot using TALOQ

gap can be significant due to the modelling errors and the differences in the controller. Compared to this, across 3 random replicates with a total budget of 200 trials, TALOQ needed 13, 34, and 48 physical trials to find policies with expected returns of 28.18, 89.18, and 93.60 (see Table 3.5). This demonstrates that combining physical and simulated trials during the learning process using TALOQ learns a much better policy. A video of one learnt policy is presented in the supplementary materials.[3]

3.5.4 General Experimental Details

We provide further details of our experiments in this section.

Covariance function: Across all experiments we use a squared exponential covariance function given in (2.13).

Treatment of hyperparameters: Algorithms 3 and 4 requires all the hyperparameters to be updated after every iteration. While this update can be performed using MLE or MAP, we follow a full Bayesian approach and compute the marginalised posterior distribution $p(f \mid \mathcal{D})$ by first placing a hyperprior distribution on ζ, the set of all hyperparameters (including beta warping parameters and lengthscale δ for TALOQ), and then marginalising it out from $p(f \mid \mathcal{D}, \zeta)$. In practice, an analytical solution for this is unlikely to exist so we approximate it using random samples from the posterior $p(\zeta \mid \mathcal{D})$ drawn using slice sampling (Neal, 2000).

Choice of hyperpriors: We assume a log-normal hyperprior distribution for all

[3]Also available at `https://youtu.be/R8Ss-dhDCmo`.

the above hyperparameters. For the variance we use ($\mu = 0, \sigma = 1$), while for the lengthscales we use ($\mu = 0, \sigma = 0.75$) across all experiments. For the beta warping parameters we used ($\mu = 2, \sigma = 0.5$) for all artificial test functions, and ($\mu = 0, \sigma = 0.5$) for the robotic simulator tasks.

Optimising the BO/BQ acquisition functions: Optimising the acquisition function is a crucial part of the BO/BQ routine. However, this is non-trivial since it tends to be non-convex, often high-dimensional, and may not have an analytic solution. Both gradient-free and gradient based optimisation methods (after suitable approximation of the acquisition function in case of non-differentiability) have been used in practice (Wilson et al., 2018). In our experiments, we used DIRECT (Jones et al., 1993) to maximise the BO acquisition function α_{ALOQ}. To minimise the BQ acquisition function, we exhaustively computed $\mathbb{V}(J(\pi_{t+1})|\mathcal{D}_{1:t}, \pi_{t+1}, \theta)$ for each θ as this was very cheap computationally.

3.6 Discussion

In this chapter, we presented ALOQ, a novel approach using BO and BQ to perform sample-efficient RL in a way that is robust to the presence of significant rare events, defined through particular settings of the environment variable which are unknown a priori. Learning a policy with ALOQ requires a simulator where we can control the environment variable. A significant disadvantage of ALOQ is that it assumes that the simulator is a perfect representation of the physical system, i.e. there is no reality gap. We also presented TALOQ, an extension to ALOQ that addresses the problem of the reality gap by actively selecting when to evaluate on the physical system instead of the imperfect simulator. Both ALOQ and TALOQ owe their remarkable sample efficiency due to the fact that they use BO and BQ. As a result, they also suffer from the disadvantages of BO and BQ, namely, and can only be used to learn low dimensional policies.

We empirically evaluated ALOQ on different simulated tasks involving a robotic arm simulator, and a hexapod locomotion task and showed how it can be also be applied to settings where the distribution of the environment variable is unknown a priori. Our results demonstrated that ALOQ outperforms multiple baselines, including related methods proposed in the literature. Further, ALOQ is computationally efficient and does not require any restrictive assumptions to be made about the environment variables. We also showed that TALOQ can be used to successfully learn a policy that is robust to the SREs while addressing the challenge posed by the reality gap.

Chapter 4

Fingerprint Policy Optimisation

In the previous chapter we presented ALOQ. We empirically demonstrated that it is highly sample efficient, which can be attributed to it using BO and BQ. However, using BO as the policy optimiser also carries with it a significant disadvantage in that it scales poorly with the dimensionality of the search space. This restricts ALOQ to low dimensional policies, which in turn might not be expressive enough for complicated problems.

In this chapter we present a new algorithm called *fingerprint policy optimisation* (FPO) to learn policies that are robust to SREs while addressing the problem of scalability mentioned above. At its core, FPO uses a policy gradient method as the policy optimiser. Compared to BO, high dimensional policies pose no particular challenge to policy gradient methods, though it comes at a cost of lower sample efficiency.

A naïve application of a policy gradient method updates a policy at each iteration by using a batch of trajectories sampled from the original distribution over environment variables irrespective of the current policy or the training iteration. Thus, it does not explicitly take into account how environment variables affect learning for different policies. Furthermore, this approach is not robust to SREs as the naïve approach will not see such rare events often enough to learn an appropriate response.

This chapter is based on material from Paul et al. (2019b) which introduced FPO.

Unlike the naïve approach, FPO explicitly models the effect of the environment variable on the policy updates, as a function of the policy. Using BO, FPO actively selects the environment distribution at each iteration of the policy gradient method in order to maximise the improvement that one policy gradient update step generates. While this can yield biased gradient estimates, FPO implicitly optimises the bias-variance tradeoff in order to maximise its one-step improvement objective.

A key design challenge in FPO is how to represent the current policy, in cases where the policy is a large neural network with thousands of parameters. To this end, we present two low-dimensional policy *fingerprints* that act as proxies for the policy. The first approximates the stationary distribution over states induced by the policy, with a size equal to the dimensionality of the state space. The second approximates the policy's marginal distribution over actions, with a size equal to the dimensionality of the action space.

We apply FPO to different continuous control tasks and show that it can outperform existing methods, including those for learning in environments with SREs. We show that both fingerprints work equally well in practice, which implies that, for a given problem, the lower dimensional fingerprint can be chosen without sacrificing performance.

4.1 Problem Setting

We assume that the environment can be characterised as an MDP. At the beginning of each episode, the environment randomly samples the environment variable θ from some (known) distribution $p(\theta)$. The agent's goal is to learn a policy $\Pi_\pi(a|s)$ mapping states $s \in \mathcal{S}$ to actions $a \in \mathcal{A}$ that maximises the expected return $J(\pi) = \mathbb{E}_\theta[R(\pi, \theta)]$. We consider environments characterised by significant rare events (SREs), i.e., there exist some low probability values of θ that generate large magnitude returns (positive or negative), yielding a significant impact on $J(\pi)$. We assume that learning is

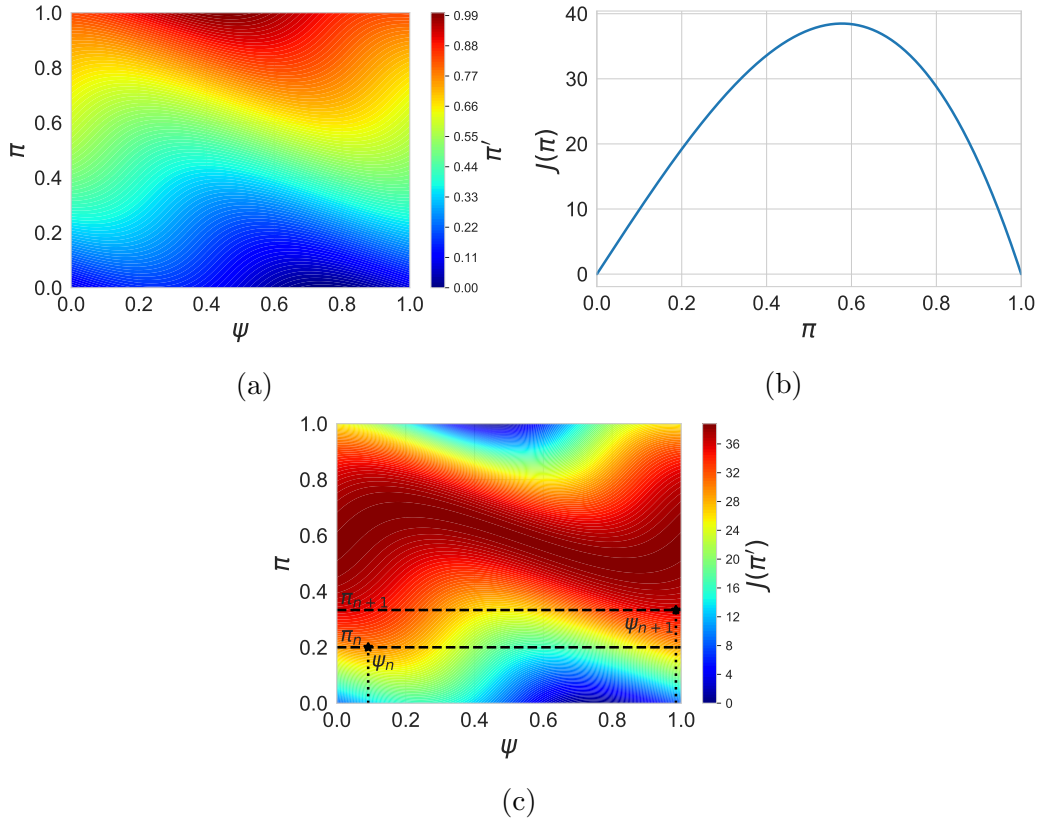


Figure 4.1: (a) The policy optimisation routine PolOpt takes input (ψ, π) and updates the policy to π'; (b) FPO directly models $J(\pi')$ as a function of (ψ, π). At iteration n, π_n is fixed and FPO selects $\psi_n|\pi_n$ based on the UCB or FITBO acquisition function, (c) after which PolOpt(ψ_n, π_n) updates the policy to π_{n+1}, and this process repeats.

undertaken in a simulator, or under laboratory conditions where θ can be actively set.

The setting of the environment variable can be considered to be a context, i.e. it specifies the objectives of agent or certain properties of the environment for particular rollout. While such problems have been studied under contextual policy search (Kupcsik et al., 2013; Fabisch and Metzen, 2014), our problem setting differs in a couple of key aspects. Contextual policy search assumes that the context is observable to the agent, while in our setting the environment variable is never observed by the agent. Furthermore, while in contextual policy search the objective is primarily to learn to generalise between contexts, our objective is to learn a policy that performs optimally under a specified distribution of the environment variable.

4.2 The Algorithm

The main idea behind FPO is to sample θ from a parametrised distribution $q_{\psi_n}(\theta)$ where the parameters ψ_n are conditioned on a fingerprint of the current policy, such that it helps the policy optimisation routine learn a policy that takes into account any SREs. Concretely, FPO executes the following steps at each iteration n. First, it selects ψ_n by approximately solving an optimisation problem defined below. Second, it samples trajectories from the environment using the current policy parametrised by π_n, where each trajectory uses a value for θ sampled from a distribution parametrised by ψ_n: $\theta \sim q_{\psi_n}(\theta)$. Third, these trajectories are fed to a policy optimisation routine, e.g., a policy gradient algorithm PolOpt, which uses them to compute an updated policy parameters, $\pi_{n+1} = \textsc{PolOpt}(\psi, \pi_n)$. Fourth, new, independent trajectories are generated from the environment with $\Pi_{\pi_{n+1}}$ and used to estimate $J(\pi_{n+1})$. Fifth, a new point is added to a dataset $\mathcal{D}_{1:n} = \{\big((\psi_{i-1}, \pi_{i-1}), J(\pi_i)\big)\}_{i=1}^{n}$, which is input to a GP. The process then repeats, with BO using the GP to select the next ψ.

The key insight behind FPO is that at each iteration n, FPO should select the ψ_n that it expects will maximise the performance of the *next* policy with parameters π_{n+1}:

$$\begin{aligned} \psi_n | \pi_n &= \operatorname*{argmax}_{\psi} J(\pi_{n+1}) \\ &= \operatorname*{argmax}_{\psi} J\big(\textsc{PolOpt}(\psi, \pi_n)\big). \end{aligned} \tag{4.1}$$

In other words, FPO chooses the ψ_n that it thinks will help PolOpt maximise the improvement to π_n that can be made in a single policy update. By modelling the relationship between ψ_n, π_n, and $J(\pi_{n+1})$ with a GP, FPO can learn from experience how to select an appropriate ψ for the current π. Modelling $J(\pi_{n+1})$ directly also bypasses the issue of modelling π_{n+1} and its relationship to $J(\pi_{n+1})$, which is infeasible when π_{n+1} is high dimensional. Note that while PolOpt has inputs (ψ, π_n), the optimisation is performed over ψ only, with π_n fixed. FPO is summarised in Algorithm

5 and illustrated in Figure 4.1. The remainder of this section describes in more detail elements of FPO that are essential to make it work in practice.

4.2.1 Selecting ψ_n

The optimisation problem in (4.1) is difficult for two reasons. First, solving it requires calling PolOpt, which is expensive in both computation and samples. Second, the observed $J(\pi)$ can be noisy due to the inherent stochasticity in the policy and the environment.

BO is particularly suited to such settings as it is sample efficient, gradient free, and can work with noisy observations. In this chapter, we consider both the UCB and FITBO acquisition functions to select $\psi_n|\pi_n$ in (4.1), and compare their performance. Formally, we model the returns $J(\pi_n)$ as a GP with inputs (ψ_{n-1}, π_{n-1}). Given a dataset $\mathcal{D}_{1:n} = \{((\psi_{i-1}, \pi_{i-1}), J(\pi_i))\}_{i=1}^n$, ψ_n is selected by maximising the UCB or FITBO acquisition function:

$$\alpha_{\text{UCB}}(\psi_n|\pi_n) = \mu(J(\pi_{n+1})|\pi_n) + \kappa\sigma(J(\pi_{n+1})|\pi_n) \tag{4.2}$$

$$\begin{aligned}\alpha_{\text{FITBO}}(\psi_n|\pi_n) =& H[p(J(\pi_{n+1})|\psi, \pi_n)] \\ &- \mathbb{E}_{p(J(\pi^*_{n+1})|\pi_n)}[H[p(J(\pi_{n+1})|\psi, \pi_n, J(\pi^*_{n+1}))]],\end{aligned} \tag{4.3}$$

where we drop conditioning on $\mathcal{D}_{1:n}$ for ease of notation. See Figure 4.1b for an illustration.

Estimating the gradient using trajectories sampled from the environment with $\theta \sim q_\psi(\theta)$ introduces bias since our objective is to learn a policy that is optimal under the target distribution $p(\theta)$ and not $q_\psi(\theta)$. While importance sampling methods (Frank et al., 2008; Ciosek and Whiteson, 2017) could correct for this bias, FPO does not explicitly do so. Instead FPO lets BO implicitly optimise a bias-variance tradeoff by selecting ψ to maximise the one-step improvement objective.

Algorithm 5 Fingerprint Policy Optimisation

input Initial policy parameters π_0, original distribution $p(\theta)$, randomly initialised $q_{\psi_0}(\theta)$, policy optimisation method PolOpt, number of policy iterations N, dataset $\mathcal{D}_0 = \{\}$

1: **for** n = 1, 2, 3, ..., N **do**
2: Sample $\theta_{1:k}$ from $q_{\psi_{n-1}}(\theta)$, and with π_{n-1} sample trajectories $\tau_{1:k}$ corresponding to each $\theta_{1:k}$
3: Compute $\pi_n = \text{PolOpt}(\tau_{1:k}) = \text{PolOpt}(\psi_{n-1}, \pi_{n-1})$
4: Compute $J(\pi_n)$ using numerical quadrature as described in Section 4.2.2. Use the sampled trajectories to compute the policy fingerprint as described in Section 4.2.3.
5: Set $\mathcal{D}_n = \mathcal{D}_{n-1} \cup \{((\psi_{n-1}, \pi_{n-1}), J(\pi_n))\}$ and update the GP to condition on $\mathcal{D}_n$
6: Use either the UCB (4.2) or FITBO (4.3) acquisition functions to select ψ_n.
7: **end for**

4.2.2 Estimating $J(\pi_n)$

Estimating $J(\pi_n)$ accurately in the presence of SREs can be challenging. A Monte Carlo estimate using samples of trajectories from the original environment requires prohibitively many samples. One alternative would be to apply an importance sampling (IS) correction to the trajectories generated from $q_{\psi_n}(\theta)$ for the policy optimisation routine. However, this is not feasible since it would require computing the IS weights $\frac{p(\tau|\theta)}{q_\psi(\tau|\theta)}$, which depend on the unknown transition function. Furthermore, even if the transition function is known, there is no reason why ψ_n should yield a good IS distribution since it is selected with the objective of maximising $J(\pi_{n+1})$.

Instead, FPO applies exhaustive summation for discrete θ and numerical quadrature for continuous θ to estimate $J(\pi_n)$. That is, if the support of θ is discrete, FPO simply samples a trajectory from each environment defined by θ and estimates $J(\pi_n) = \sum_{l=1}^{L} p(\theta = \theta_l) R(\theta_l, \pi_n)$. To reduce the variance due to stochasticity in the policy and the environment, we can sample multiple trajectories from each θ_l. For continuous θ, we apply an adaptive Gauss-Kronrod quadrature rule to estimate $J(\pi_n)$. While numerical quadrature may not scale to high dimensions, in practice θ is usually low dimensional, making this a practical design choice.

Since for discrete θ we evaluate $J(\pi_n)$ through exhaustive summation, it is natural to consider a variation of the naïve approach, wherein $\nabla J(\pi_n)$ is also evaluated in the same manner during training, i.e., $\nabla J(\pi_n) = \sum_{l=1}^{L} p(\theta = \theta_l) \nabla J(\theta_l, \pi_n)$. We call this the 'Enum' baseline since the gradient is estimated by enumerating over all possible values of θ. Our experiments in Section 4.3 show that this baseline is unable to match the performance of FPO.

4.2.3 Policy Fingerprints

True global optimisation is limited by the curse of dimensionality to low-dimensional inputs, and BO has had only rare successes in problems with more than twenty dimensions (Wang et al., 2013). In FPO, many of the inputs to the GP are policy parameters and in practice, the policy may be a neural network with thousands of parameters. While Rajeswaran et al. (2017b) show that linear and radial basis function policies can perform as well as neural networks in some simulated continuous control tasks, even these policies have hundreds of parameters, far too many for a GP.

Thus, we need to develop a policy *fingerprint*, i.e., a representation that is low dimensional enough to be treated as an input to the GP but expressive enough to distinguish the policy from others. Foerster et al. (2017) showed that a surprisingly simple fingerprint, consisting only of the training iteration, suffices to stabilise multi-agent Q-learning. Such a fingerprint is insufficient for FPO, as the GP fails to model the response surface and treats all observed $J(\pi)$ as noise. However, the principle still applies: a simplistic fingerprint that discards much information about the policy can still be sufficient for decision making, in this case to select ψ_n.

In this spirit, we propose two fingerprints. The first, the *state fingerprint*, augments the training iteration with an estimate of the stationary state distribution induced by the policy. In particular, we fit an anisotropic Gaussian to the set of states visited in the trajectories sampled while estimating $J(\pi)$ (see Section 4.2.2). The size of this

fingerprint grows linearly with the dimensionality of the state space, instead of the number of parameters in the policy.

In many settings, the state space is high dimensional, but the action space is low dimensional. Therefore, our second fingerprint, the *action fingerprint*, is a Gaussian approximation of the marginal distribution over actions induced by the policy: $a(\pi) = \int \Pi_\pi(a|s)p(s|\pi)ds$ (here $p(s|\pi)$ is the stationary state distribution induced by Π_π), sampled from trajectories as with the state fingerprint.

Of course, neither the stationary state distribution nor the marginal action distribution are likely to be Gaussian and could in fact be multimodal. Furthermore, the state distribution is estimated from samples used to estimate $J(\pi)$, and not from $p(\theta)$. However, as our results show, these representations are nonetheless effective, as they do not need to accurately describe each policy, but instead just serve as low dimensional fingerprints on which FPO conditions.

4.2.4 Covariance Function

Our choice of policy fingerprints means that one of the inputs to the GP is a probability distribution. Thus for our GP prior we use a covariance that is the product of three terms, $k_1(\psi, \psi'), k_2(n, n')$ and $k_3\big(\textsc{fgp}(\Pi), \textsc{fgp}(\Pi')\big)$, where each of k_1, k_2, and k_3 is a squared exponential covariance function and $\textsc{fgp}(\Pi)$ is the state or action fingerprint of Π. Similar to Malkomes et al. (2016), we use the Hellinger distance to replace the Euclidean in k_3: this covariance remains positive-semi-definite as the Hellinger is effectively a modified Euclidean.

4.3 Experiments

To evaluate the empirical performance of FPO, we start by applying it to a simple problem: a modified version of the cliff walker task (Sutton and Barto, 2018), with one dimensional state and action spaces. We then move on to simulated robotics

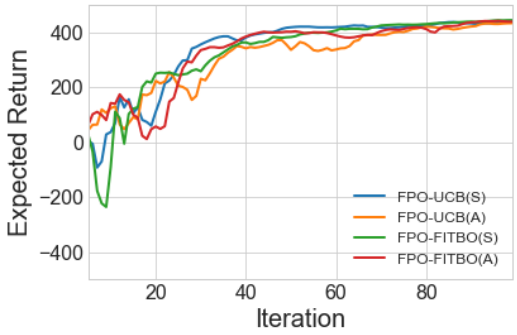
problems based on the MuJoCo simulator (Brockman et al., 2016) with much higher dimensionalities. These were modified to include SREs. We aim to answer two questions: (1) How do the different versions of FPO (UCB vs. FITBO acquisition functions, state (S) vs. action (A) fingerprints) compare with each other? (2) How does FPO compare to existing methods (Naïve, Enum, OFFER, EPOpt, ALOQ), and ablated versions of FPO. We use TRPO as the policy optimisation method combined with neural net policies. We also include θ as an input to the baseline (but not to the policy) since it is observable during training.

Since ALOQ does not scale well to high dimensional policies, we were able to apply it only on the cliff walker problem. The policy dimensionality and the total number of iterations for the simulated robotic tasks were far too high. Note also that, while we compare FPO to EPOpt, these methods optimise for different objectives.

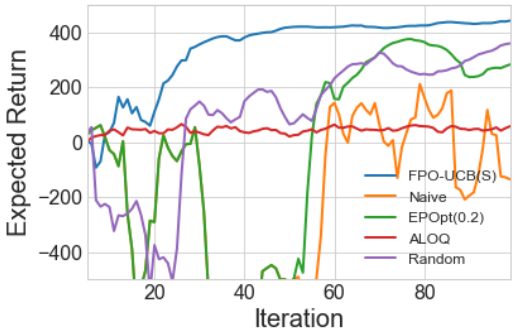
4.3.1 Cliff Walker

We start with a modified version of the cliff walker problem where instead of a gridworld we consider an agent moving in a continuous state space $\mathbb{R}$; the agent starts randomly near the state 0, and at each timestep can take an action $a \in \mathbb{R}$. The environment then transitions the agent to a new location $s' = s + 0.025\text{sign}(a) + 0.005\epsilon$, where ϵ is standard Gaussian noise. The location of the cliff is given by $1 + \theta$, where $p(\theta) \sim Beta(2, 1)$. If the agent's current state is lower than the cliff location, it gets a reward equal to its state; otherwise it falls off the cliff and gets a reward of -5000, terminating the episode. Thus the objective is to learn to walk as close to the cliff edge as possible without falling over.

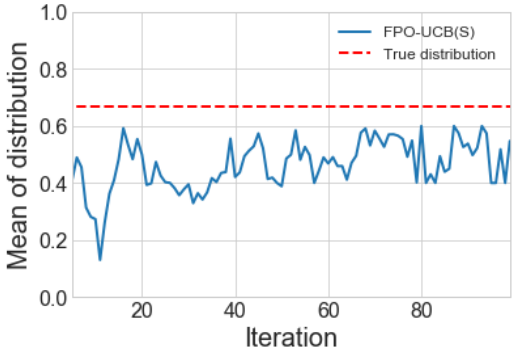
Figure 4.2a shows that all versions of FPO do equally well on the task. Figure 4.2b shows that FPO-UCB(S) learns a policy with a higher expected return than all other baselines. This is not surprising since, as discussed in Section 2.4, the gradient estimates without active selection of ψ are likely to have high variance due to the



(a) Different versions of FPO



(b) FPO vs baselines



(c) FPO vs baselines

Figure 4.2: Results for the Cliff Walker environment. (a) Comparison of the performance of different versions of FPO. (b) Comparison of FPO-UCB(S) against the baselines. (c) The mean of the true distribution of the cliff location ('True distribution'), and the mean of the distribution selected by FPO-UCB(S) during learning (FPO-UCB(S)).

presence of SREs. For EPOpt we set $\epsilon = 0.2$ and perform rejection sampling after 50 iterations. The poor performance of ALOQ is expected since even in this simple problem, the policy dimensionality of 47 is high for BO. We could not run OFFER since an analytical solution of $\nabla_\psi q_\psi(\theta)$ does not exist. In Table 4.1 we present the quartiles of the expected return of the final learnt policy for each method across the 10 random starts.

Figure 4.2c compares the mean of sampling distribution for the cliff location chosen by FPO-UCB(S), i.e., $\mathbb{E}_{q_{\psi_n}}[\theta]$, against the mean of true distribution of the cliff location ($\mathbb{E}_p[\theta]$). FPO-UCB(S) selects sampling distributions for the cliff location that are more conservative than the true distribution. This is expected as falling off the cliff has a

	Q1	Median	Q2
FPO-UCB(S)	427.1	441.5	450.0
FPO-UCB(A)	335.2	432.6	440.4
FPO-FITBO(S)	428.1	443.6	453.1
FPO-FITBO(A)	372.2	438.2	451.5
Naïve	-1478.7	-135.5	243
EPOpt	-44.4	282.1	354.4
ALOQ	33.5	57.2	77.2
Random	345.8	358.9	373.4

Table 4.1: Quartiles of expected return across ten random starts for the Cliff Walker environment.

significant cost and closer cliff locations need to be sampled more frequently to learn a policy that avoids the low probability events where the cliff location is close under the true distribution.

The large negative reward for falling off the cliff acts like an SRE in this setting, so we can modify it to have no SRE by setting the reward to 0, but retain the randomised location of the cliff as an environment variable that affects the dynamics.

4.3.2 Half Cheetah

Next we consider simulated robotic locomotion tasks using the Mujoco simulator. In the original OpenAI Gym HalfCheetah task, the objective is to maximise forward velocity. We modify the original problem such that in 98% of the cases the objective is to achieve a target velocity of 2, with rewards decreasing linearly with the distance from the target. In the remaining 2%, the target velocity is set to 4, with a large bonus reward, which acts as an SRE.

Figure 4.3a shows that FPO with UCB outperforms FPO with FITBO. We suspect that this is because FITBO tends to over-explore. Also, as mentioned in Section 2.5.2, it was developed with the aim of minimising simple regret and it is not known how efficient it is at minimising cumulative regret. As in cliff walker, both the action and state fingerprints perform equally well with UCB. In Table 4.2 we present the quartiles

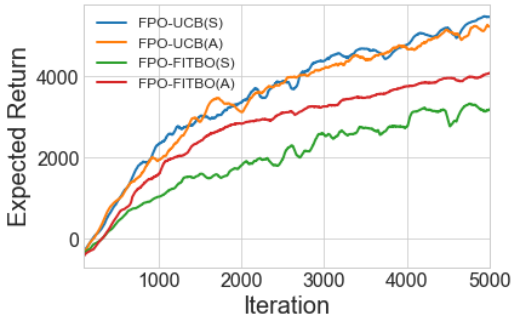
	Q1	Median	Q2
FPO-UCB(S)	3913.7	5464.0	5905.5
FPO-UCB(A)	4435.6	5231.8	5897.6
FPO-FITBO(S)	2973.9	3187.2	3923.7
FPO-FITBO(A)	3686.2	4091.1	7247.3
Naïve	1059.9	1071.1	1086.0
EPOpt	803.6	4066.0	4421.0
OFFER	1093.9	1097.4	1111.2
Random	1722.3	2132.6	2645.5
Enum	2442.8	2796.0	3428.4

Table 4.2: Quartiles of expected return across 10 random starts for the Half Cheetah environment.

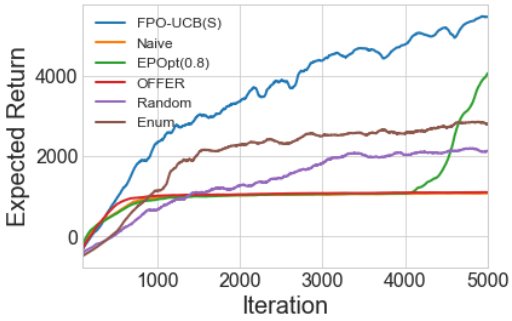
of the expected return of the final learnt policy for each method across the 10 random starts.

Figure 4.3b shows that the Naïve method and OFFER converge to a locally optimal policy that completely ignores the SRE, while the random selection of ψ does slightly better. The Enum baseline performs better than random, but is still far worse than FPO. This shows that the key to learning a good policy is the active selection of ψ_n, which also includes the implicit bias-variance tradeoff performed by FPO. We set $\epsilon = 0.8$ for EPOpt, but in this case we use the trajectories with returns exceeding the threshold for the policy optimisation since the SRE has a large positive return. Although its performance increases after iteration 4,000, it is extremely sample inefficient, requiring about five times the samples of FPO. We did not run ALOQ as it is entirely infeasible given the policy dimensionality ($> 20,000$).

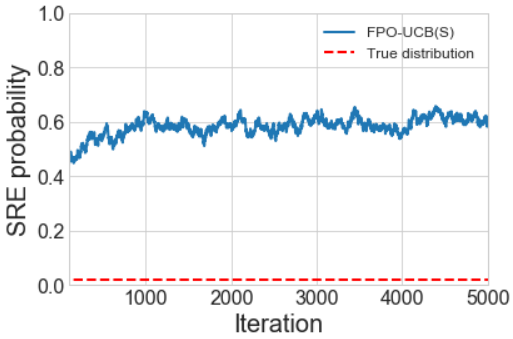
Finally, Figure 4.3c presents the schedule of ψ, (i.e., the probability of the target velocity being 4) as selected by FPO-UCB(S) across different iterations. FPO learns to vary ψ, starting with 0.5 initially, to hovering around 0.6 once the cheetah can consistently reach velocities greater than 2.



(a) Different versions of FPO



(b) FPO vs baselines



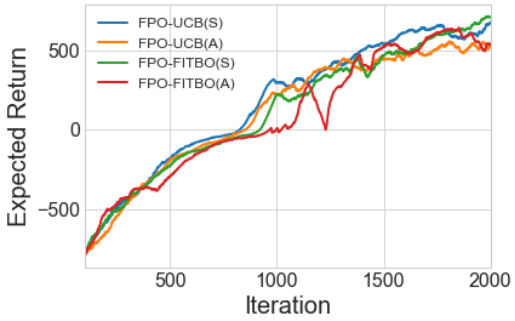
(c) Learnt ψ

Figure 4.3: Results for the Half Cheetah environment. (a) Comparison of the performance of different versions of FPO. (b) Comparison of FPO-UCB(S) against the baselines. (c) The actual probability of velocity target being 4 ('True distribution'), and the probability selected by FPO-UCB(S) during learning (FPO-UCB(S)).

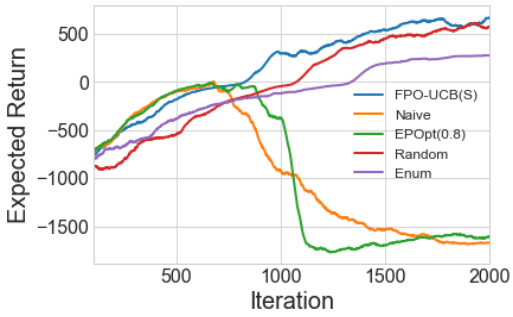
4.3.3 Ant

The ant environment is much more difficult than half cheetah since the agent moves in 3D, and the state space has 111 dimensions compared to 17 for half cheetah. The larger state space also makes learning difficult due to the higher variance in the gradient estimates. We modify the original problem such that velocities greater than 2 carry a 5% chance of damage to the ant. On incurring damage, which we treat as the SRE, the agent receives a large negative reward, and the episode terminates.

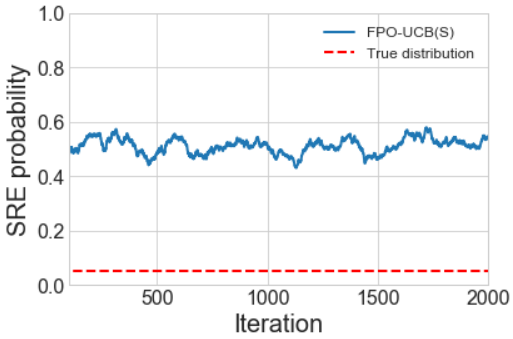
Figure 4.4a compares the performance of the UCB versions of FPO. There is no significant difference between the performance of the state or action fingerprint, or between the UCB and FITBO acquisition functions. Thus, the lower dimensional fingerprint (in this case the action fingerprint) can be chosen without compromising



(a) Different versions of FPO



(b) FPO vs baselines



(c) Learnt ψ

Figure 4.4: Results for the Ant environment. (a) Comparison of the performance of different versions of FPO. (b) Comparison of FPO-UCB(S) against the baselines. (c) The actual probability of velocity target being 4 ('True distribution'), and the probability selected by FPO-UCB(S) during learning (FPO-UCB(S)).

performance.

In Table 4.3 we present the quartiles of the expected return of the final learnt policy for each method across the 10 random starts.

Figure 4.4b shows that for the Naïve method and EPOpt, performances drops significantly after about 750 iterations. This is because the learned policies generate velocities beyond 2, which after factoring in the effect of the SRE, yield much lower expected returns. Since the SREs are not seen often enough, these methods do not learn that higher velocities actually yield lower expected returns. The Enum baseline once again performs better than the naïve approach, but still worse than FPO. The random baseline performs better, and eventually matches the performance of FPO. We could not run OFFER in this setting as computing the IS weights would require

	Q1	Median	Q2
FPO-UCB(S)	490.6	674.2	713.3
FPO-UCB(A)	408.1	519.4	629.7
FPO-FITBO(S)	626.8	704.2	770.0
FPO-FITBO(A)	455.7	533.1	707.0
Naïve	-1746.9	-1669.4	-1585.2
EPOpt	-1732.3	-1606.6	-1454.5
Random	460.3	575.6	640.4
Enum	255.7	273.4	285.6

Table 4.3: Quartiles of expected return across 10 random starts for the Ant environment.

knowledge of the transition model.

Figure 4.4c shows that the optimal schedule for ψ, the probability of damage for velocities greater than 2, as learnt by FPO-UCB(S), hovers around 0.5. This helps explain the relatively good performance of the random baseline: since the baseline samples $\psi \sim U(0, 1)$ at each iteration, the expected value of ψ is 0.5, and thus we can expect it to find a good policy eventually. Of course, active selection of ψ still matters, as evidenced by FPO outperforming it initially.

A natural question that arises based on the above result, is whether the schedule learnt by FPO-UCB(S) is better than a fixed schedule wherein the probability of breakage is fixed to some value throughout training. We investigate this by running another set of baselines wherein the probability of breakage is fixed to each of $\{0.3, 0, 4, 0.5, 0.6, 0.7\}$ throughout training. The learning curves are presented in Figure 4.5, with 'Fixed(x)' referring to the baseline with the probability of breakage fixed at x. While all of the baselines finally converge to a policy that performs on par with FPO-UCB(S), they learn much slower initially. This goes to show that learning a schedule that takes into account the current policy is key to faster learning.

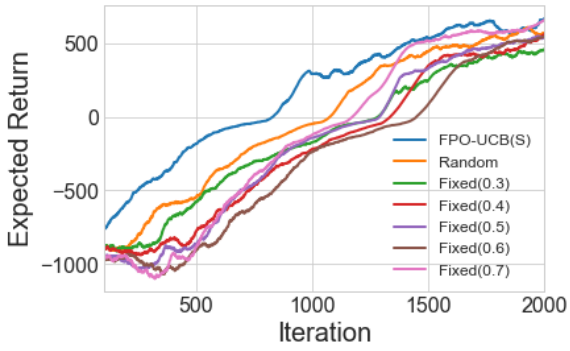
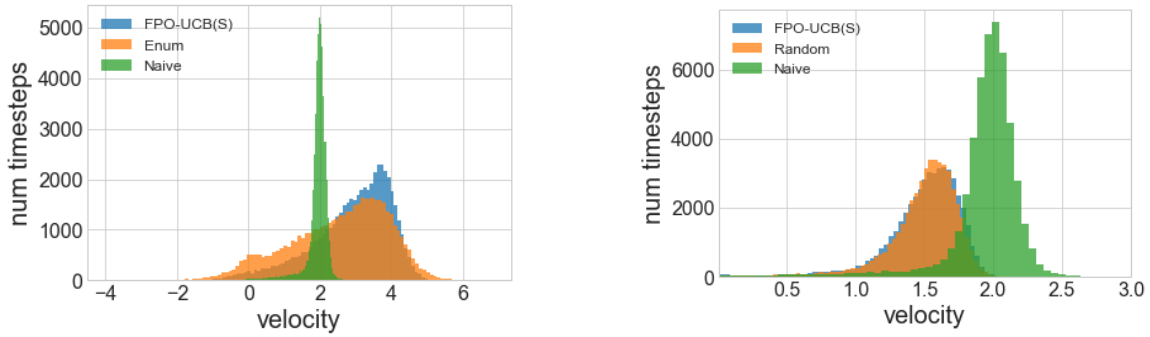


Figure 4.5: Comparison of FPO-UCB(S) against the Fixed baselines



(a) HalfCheetah

(b) Ant

Figure 4.6: Histogram of the velocity profile of the final learnt policies for each method.

4.3.4 Further Examination of the Learnt Policies

As explained in the Experiments section, the SREs for the HalfCheetah and Ant experiments are based on the velocities achieved by the agent; for HalfCheetah the SRE is defined as the velocity target being 4 (and carrying a large bonus reward) instead of 2 with a 2% probability of occurrence, while for Ant velocities greater than 2 has a 5% probability of incurring a large cost. Here we compare the performance of FPO-UCB(S) against the next best baseline (Enum for HalfCheetah and Random for Ant) and the Naïve baseline by visualising the velocity profiles of the final learnt policy. For each random start for each method we sampled 10 trajectories (for a total of 100 trajectories per method) and plot the histogram of the velocity at each timestep. This is presented in Figure 4.6.

For the HalfCheetah task, from Figure 4.6a we can see that the velocity profile

for the Naïve approach is highly concentrated around 2. This goes to show that the Naïve approach learns a policy that does not take into account the SRE at all. On the other hand, both Enum and FPO-UCB(S) have velocity profiles with much higher variance with a lot of mass spread between 2 and 4. This goes to show that both of them take into account the effect of SREs. However, FPO-UCB(S) manages to better balance the SRE/non-SRE rewards and has slightly higher mass concentrated on 4, which in turn leads to it significantly outperforming Enum.

For the Ant task, unsurprisingly once again the Naïve approach completely ignores the SREs, and exhibits a velocity profile that is greater than 2 roughly 50% of the time. The velocity profiles of the Random baseline and FPO-UCB(S) are almost exactly the same. This is unexpected as there is no significant difference between the expected return of the final policies learnt by these two methods, as shown in Section 4.3. As noted earlier, the good performance of Random is not unsurprising since the schedule for ψ chosen by FPO-UCB(S) is close to 0.5, which is also the mean of ψ under the random baseline as $\psi \sim U(0, 1)$.

4.3.5 Performance in settings without SREs

FPO considers the setting where environments are characterised by SREs. A natural question to ask is how does its performance compare to the naïve method in settings where there are no SREs. To investigate this we applied FPO-UCB(S), and the naïve baseline, to the cliff walker problem presented in Section 4.3.1, with the modification that falling off the cliff now carries 0 reward instead of -5000. This removes the SRE, but the environment variable (the location of the cliff) is still relevant since it has a significant effect on the dynamics. The results are presented in Figure 4.7. Note that the performance of the Naïve method is far more stable than in the setting with SRE. However, while it is able to learn a good policy, FPO-UCB(S) still performs better since it takes into account the effect of the environment variable.

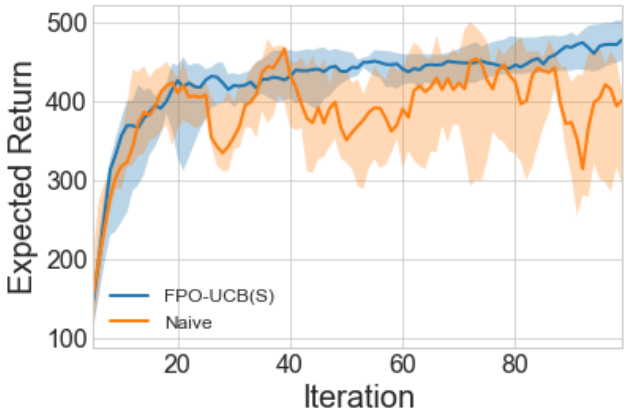


Figure 4.7: Results for the Cliff Walker environment without any SRE. Solid line shows the median and the shaded region the quartiles across 10 random starts.

4.3.6 Hyperparameter Settings

Our implementation is based on rllab Duan et al. (2016b) and as such most of the hyperparameter settings were kept to their default settings. The details are provided in Table 4.4.

Table 4.4: Experimental details

	Cliff Walker	HalfCheetah	Ant
PolOpt	TRPO	TRPO	TRPO
KL constraint	0.01	0.01	0.01
Discount rate	0.99	0.99	0.99
GAE λ	1.0	1.0	1.0
Batch size	10,000	12,500	12,500
Policy layers	(5,5)	(100,100)	(100,100)
Policy units	Tanh	ReLU	ReLU
FPO-UCB κ	2	2	2

4.4 Discussion

Environment variables can have a significant impact on the performance of a policy and are pervasive in real-world settings. In the previous chapter, we presented ALOQ and TALOQ, both of which suffer from a significant drawback – they can only be

used to learn low dimensional policies. In this chapter we presented FPO, which addresses this by using policy gradients methods as the underlying policy optimisation algorithm, which can be used to learn high dimensional neural net policies. FPO ensure robustness to SREs by actively selecting the environment variables during learning using BO to actively sample the distribution of the environment variables at each iteration of the training loop.

We introduced novel state and action fingerprints that can be used by BO with a one-step improvement objective to make FPO scalable to high dimensional tasks irrespective of the policy dimensionality. We applied FPO to a number of continuous control tasks of varying difficulty and showed that FPO can efficiently learn policies that are robust to significant rare events, which are unlikely to be observable under random sampling but are key to learning good policies. We also showed that if applied to environments without SREs, the performance of FPO is no worse than the underlying policy gradient algorithm.

Chapter 5

Hyperparameter Optimisation on the Fly

In the previous chapter we showed how FPO enables an underlying policy gradient method to be robust to SREs, by performing active selection of the environment variable distribution at each iteration. However, even in environments without SREs, the performance of policy gradient methods tend to be highly sensitive to their hyperparameters, for example, the learning rate, or the discount factor. As a result, these methods require significant hyperparameter tuning. While many hyperparameter tuning methods have been suggested in the context of supervised learning, directly applying these to reinforcement learning would require performing multiple learning runs, which in turn would compromise the sample efficiency.

In this chapter we present Hyperparameter Optimisation on the Fly (HOOF), a method to automatically learn the optimal hyperparameters of an underlying policy gradient method. The main idea is as follows: At each iteration, sample trajectories using the current policy. Next, generate some candidate policies using a policy gradient method with different hyperparameters, and estimate their value sample efficiently by using an *off-policy* method. Finally, update the policy greedily with respect to the estimated value of the candidates. In practice, HOOF uses importance sampling (IS) to construct off-policy estimates of the value of each candidate policy.

This chapter is based on material from Paul et al. (2019a) which introduced HOOF.

The viability of such a simple approach is counter-intuitive since off-policy evaluation using IS tends to have high variance that grows rapidly as the behaviour and evaluation policies diverge (Thomas, 2015). However, HOOF is motivated by the insight that the high variance does not pose a problem as long as the IS estimates are able to accurately rank the relative performance of the different candidate policies. In natural gradient methods such as Natural Policy Gradient (NPG) and Trust Region Policy Optimisation (TRPO), constraints on the magnitude of the update in policy space ensure that the IS estimates remain informative with regards to ranking the relative performance of the candidate policies. While this is not the case for first order methods, we show that adding a simple KL constraint, without any of the complications of second order methods, suffices to keep IS estimates informative and enable effective hyperparameter optimisation. We further show that the performance of HOOF is robust to the setting of this KL constraint. HOOF is 1) sample efficient, requiring no more than one training run; 2) computationally efficient compared to sequential and parallel search methods; 3) able to learn a dynamic schedule for the hyperparameters that outperforms methods that learn fixed hyperparameter settings; and 4) simple to implement.

We evaluate HOOF across a range of simulated continuous control tasks using the Mujoco OpenAI Gym environments (Brockman et al., 2016). First, we apply HOOF to A2C (Mnih et al., 2016), and show that using it to learn the learning rate can improve performance. We also perform a benchmarking exercise where we use HOOF to learn both the learning rate and the weighting for the entropy term and compare it against a grid search across these two hyperparameters. Next, we show that using HOOF to learn optimal hyperparameter schedules for NPG can outperform TRPO. This suggests that while strictly enforcing the KL constraint enables TRPO to outperform NPG, doing so becomes unnecessary once we can properly adapt NPG's hyperparameters.

5.1 Method

Consider the problem of applying a policy gradient method to learn a policy for an MDP. The update for the policy parameters given in (2.3) can be written as:

$$\pi' = \pi + f(\psi). \tag{5.1}$$

Here $f(\psi)$ represents a step along the gradient direction for some objective function estimated from a batch of sampled trajectories $\{\tau_1^\pi, \tau_2^\pi, \ldots, \tau_K^\pi\}$, and ψ is the set of hyperparameters. For policy gradient methods with generalised advantage estimation (GAE), $\psi = (\alpha, \gamma, \lambda)$, (here α is the learning rate, and γ, λ are the GAE hyperparameters - see 2.4.2) and the update takes the form:

$$f(\alpha, \gamma, \lambda) = \underbrace{\alpha \sum_t \nabla \log \Pi_\pi(a_t|s_t) A_t^{GAE(\gamma,\lambda)}}_{g(\gamma,\lambda)}. \tag{5.2}$$

The main idea behind HOOF is to automatically adapt the hyperparameters during training by greedily maximising the value of the updated policy, i.e., starting with policy parameters π_n at iteration n, HOOF sets

$$\begin{aligned} \psi_n &= \underset{\psi}{\operatorname{argmax}}\, J(\pi_{n+1}) \\ &= \underset{\psi}{\operatorname{argmax}}\, J(\pi_n + f(\psi)), \end{aligned} \tag{5.3}$$

Given a set of sampled trajectories, $f(\psi)$ can be computed for any ψ, and thus we can generate different candidate π_{n+1} without requiring any further samples. However, solving the optimisation problem in (5.3) requires evaluating $J(\pi_{n+1})$ for each such candidate. Any on-policy approach would have prohibitive sample requirements, so HOOF uses weighted importance sampling (WIS) to construct an off-policy estimate of $J(\pi_{n+1})$. Given sampled trajectories $\{\tau_1^{\pi_n}, \tau_2^{\pi_n}, .., \tau_K^{\pi_n}\}$, with corresponding returns $\{R_1^{\pi_n}, R_2^{\pi_n}, ..., R_K^{\pi_n}\}$, the WIS estimate of $J(\pi_{n+1})$ is given by:

$$J(\pi_{n+1}) = \sum_{k=1}^{K} \left(\frac{w_k}{\sum_{k=1}^{K} w_k} \right) R_k^{\pi_n}, \tag{5.4}$$

where $w_k = \frac{P(\tau_k^{\pi_n} \sim \Pi_{\pi_{n+1}})}{P(\tau_k^{\pi_n} \sim \Pi_{\pi_n})}$. Since $p(\tau|\pi) = p(s_0) \prod_{i=0}^{T} \Pi_\pi(a_i|s_i) p(s_{i+1}|s_i, a_i)$, the transitions cancel out and we have:

$$w_k = \frac{\prod_{i=0}^{T} \Pi_{\pi_{n+1}}(a_i|s_i^k)}{\prod_{i=0}^{T} \Pi_{\pi_n}(a_i|s_i^k)}. \tag{5.5}$$

The success of this approach depends critically on the quality of the WIS estimates, which can suffer from high variance that grows rapidly as the distributions of π_{n+1} and π_n diverge. Fortunately, for natural gradient methods like NPG, $KL(\Pi_{n+1}||\Pi_n)$ is automatically approximately bounded by the update, ensuring reasonable WIS estimates when HOOF directly uses (5.3). In the following, we consider the more challenging case of first order methods.

5.1.1 First Order HOOF

Without a KL bound on the policy update, it may seem that WIS will not yield adequate estimates to solve (5.3). However, a key insight is that, while the estimated policy value can have high variance, the relative ordering of the policies, which HOOF solves for, has much lower variance. Nonetheless, HOOF could still fail if $KL(\Pi_{n+1}||\Pi_n)$ becomes too large, which can occur in first order methods. Hence, First Order HOOF modifies (5.3) by constraining $KL(\Pi_{n+1}||\Pi_n)$:

$$\psi_n = \underset{\psi}{\operatorname{argmax}}\, J(\pi_{n+1}) \quad \text{s.t.} \quad KL(\Pi_{n+1}||\Pi_n) < \epsilon. \tag{5.6}$$

While this yields an update that superficially resembles that of natural gradient methods, the KL constraint is applied only during the search for the optimal hyperparameter settings using WIS. The direction of the update is determined solely by a first order gradient update rule, and estimation and inversion of the FIM is not required. From a practical perspective, this constraint is enforced by computing the KL for each candidate policy based on the observed trajectories, and the candidate is rejected if this sample KL is greater than the constraint.

Algorithm 6 HOOF

input Initial policy parameters π_0, number of policy iterations N, search space for ψ, KL constraint ϵ if using first order policy gradient method.

for n = 0, 1, 2, 3, . . . , N **do**
  Sample trajectories $\tau_{1:K}$ using π_n.
  for z = 1, 2, . . . Z **do**
    Generate candidate hyperparameter $\{\psi_z\}$ from the search space.
    Compute candidate policy π^z using ψ_z in (5.1)
    Estimate $J(\pi^z)$ using WIS (5.4)
    Compute $KL(\Pi^z||\Pi_n)$ if using first order policy gradient method,
  end for
  Select ψ_n, and hence π_{n+1}, according to (5.6) or (5.3)
end for

If learning the learning rate using HOOF, we can also use the KL constraint to dynamically adjust the search bounds: At each iteration, if none of the candidates violate the KL constraint, we increase the upper bound of the search space by a factor ν, while if a large proportion of the candidates violate the KL constraint, we reduce the upper bound by ν. This makes HOOF even more robust to the initial setting of the search space. Note that this is entirely optional, and is simply a means to reduce the number of number of candidates that would otherwise need to be generated and evaluated to ensure that a good solution of (5.3) is found.

5.1.2 (γ, λ) Conditioned Value Function

If we use HOOF to learn (γ, λ), $g(\gamma, \lambda)$ has to be computed for each setting of (γ, λ). With neural net value functions, we modify our value function such that its inputs are (s, γ, λ), similar to Universal Value Function Approximators (Schaul et al., 2015). Thus we learn a (γ, λ)-conditioned value function that can make value predictions for any candidate (γ, λ) at the cost of a single forward pass.

5.1.3 Robustness to HOOF Hyperparameters and Computational Costs

HOOF introduces two types of hyperparameters of its own: the search spaces for the various hyperparameters it tunes, and the number of candidate policies generated for evaluation. Since the candidate policies are generated using random search, these hyperparameters express a straight up trade-off between performance and computational cost: A larger search space and larger number of candidates should lead to better solution for (5.3), but incur higher computational cost. However, just like in random search, the generation and evaluation of the candidate policies can be performed in parallel to reduce wall clock time. Alternatively, Bayesian Optimisation could be used to solve (5.3) efficiently. Finally, we note that HOOF with random search is always more computationally efficient than grid/random search over the hyperparameters with the same number of candidates, as HOOF saves on the additional computational cost of sampling trajectories for each candidate incurred by grid/random search. HOOF additionally introduces the KL constraint hyperparameter for first order methods. We show experimentally that the performance of HOOF is robust to a wide range of settings for this.

5.1.4 Choice of Optimiser

We use random search as the optimiser for (5.3) to show that the simplest methods suffice. However, any gradient-free optimiser could be used instead. For example, grid search, CMA-ES (Hansen and Ostermeier, 2001), or Bayesian Optimisation (Brochu et al., 2010) are all viable alternatives.

Gradient based methods are not viable for two reasons. First, they require that $J(\pi_{n+1})$ be differentiable w.r.t. the hyperparameters, which might be difficult or impossible to compute, e.g. with the TRPO update. Second, they introduce learning rate and initialisation hyperparameters, which require tuning at the expense of sample

efficiency.

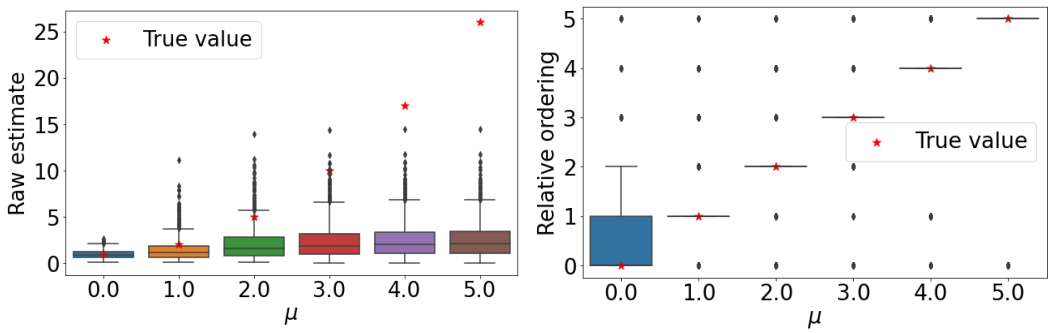
5.2 Experiments

In this section we first illustrate the ordering effet of WIS using a simple toy example. Next, to experimentally validate HOOF, we apply it to four simulated continuous control tasks from MuJoCo OpenAI Gym (Brockman et al., 2016): HalfCheetah, Hopper, Ant, and Walker. We start with A2C, and show that HOOF performs better than multiple baselines, and is also far more sample efficient. Finally, we use NPG as the underlying policy gradient method and apply HOOF to learn $(\delta, \gamma, \lambda)$ and show that it outperforms TRPO.

We repeat all experiments across 10 random starts. In all figures solid lines represent the median, and shaded regions the quartiles. Similarly all results in tables represent the median. Hyperparameters that are not tuned are held constant across HOOF and baselines to ensure comparability.

5.2.1 Illustration of the Ordering Effect of WIS

We illustrate the assertion about the relative ordering of WIS estimates through a simple experiment: Let $p(x) = N(0, 1)$ be our behaviour distribution. We are interested in $E_{q_i(x)}[X^2]$ where $q_i(x) = N(\mu_i, 1)$, $\mu_i = \{0, 1, 2, 3, 4, 5\}$. We can compute the true value analytically as $1 + \mu_i^2$. Now we compare this to a WIS estimate: we sample 10 points from $p(x)$ and use them to estimate $E_{q_i(x)}[X^2]$. We repeat this 1000 times. The boxplot of the WIS estimates in Fig 5.1a shows that we cannot rely on them directly as they becomes worse as $q_i(x)$ diverges from $p(x)$. However, in Fig 5.1b we see that the relative ordering is reliable. Note this does not guarantee that by using WIS to estimate the value of each candidate policy will always lead to a correct solution in (5.3). However, any factor that leads to better estimates of the policy value (for example, increasing the number of trajectories sampled) is also likely



(a) (b)

Figure 5.1: In (a) the WIS estimates of $E_{q_i(x)}[X^2]$ diverges from the true values as $q_i(x)$ diverges from $p(x)$. However (b) shows that the relative ordering based on the WIS estimates is reliable.

to lead to a better estimate of the relative ordering which (5.3) relies on.

5.2.2 HOOF with A2C

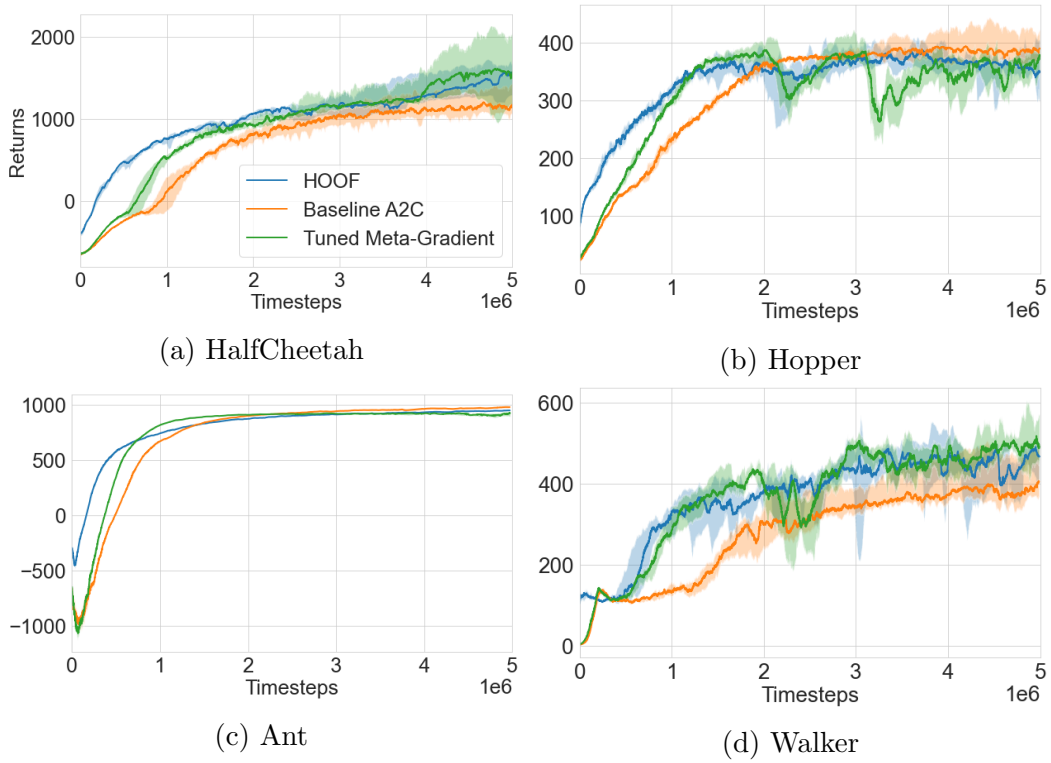
In the A2C framework, a neural net with parameters π is commonly used to represent both the policy and the value function, usually with some shared layers. The update function (5.1) for A2C is a linear combination of the gradients of the policy loss, the value loss, and the policy entropy:

$$f_\pi(\alpha) = \alpha\{\nabla_\pi \log \Pi_\pi(a|s)(R - V_\pi(s)) + c_1 \nabla_\pi (R - V_\pi(s))^2 + c_2 \nabla_\pi H(\Pi_\pi(s))\}, \quad (5.7)$$

where we have omitted the dependence on the timestep and other hyperparameters for ease of notation. The performance of A2C is particularly sensitive to the choice of the learning rate α (Henderson et al., 2018b), which requires careful tuning.

We learn α using HOOF with the KL constraint $\epsilon = 0.03$ ('HOOF'). We compare this against two baselines: (1) Baseline A2C, i.e., A2C with the initial learning rate set to the OpenAI Baselines default (0.0007), and (2) learning rate being learnt by meta-gradients ('Tuned Meta-Gradient'), where the hyperparameters introduced by meta-gradients were tuned using grid search.

The learning curves in Figure 5.2 shows that across all environments HOOF learns faster than Baseline A2C, and also outperforms it in HalfCheetah and Walker,



(a) HalfCheetah

(b) Hopper

(c) Ant

(d) Walker

Figure 5.2: Performance of HOOF with $\epsilon = 0.03$ compared to Baseline A2C and Tuned Meta-Gradients. The hyperparameters (α_0, β) of meta gradients had to be tuned using grid search which required 36x the samples used by HOOF.

demonstrating that learning the learning rate online can yield significant gains.

The hyperparameters learnt by HOOF are presented in Figure 5.3.

5.2.2.1 Comparison to Meta-Gradients

The meta-gradient algorithm for hyperparameters ψ proceeds as follows: 1) Sample trajectories $\tau^{\pi}_{1:K} \sim \Pi_{\pi}$. 2) Update $\pi' = \pi + f_{\pi}(\psi)$ (where f_{π} is as per (5.7)). 3) Sample trajectories $\tau^{\pi'}_{1:K} \sim \Pi_{\pi'}$. 4) Update $\psi' = \psi + \beta \frac{\partial J'(\tau^{\pi'}_{1:K}, \bar{\psi})}{\partial \pi'} \frac{\partial f_{\pi}(\psi)}{\partial \psi}$, where J' is the meta-objective with $\bar{\psi}$ the set of reference hyperparameters introduced by the meta-gradient algorithm to balance bias-variance tradeoff within the meta-objective, and β is the meta learning rate. For $\psi = \alpha$, $\frac{\partial f_{\pi}(\psi)}{\partial \psi} = \frac{f_{\pi}(\psi)}{\alpha}$, and we can use the policy loss as the meta objective, with $\frac{\partial J'(\tau^{\pi'}_{1:K}, \bar{\psi})}{\partial \pi'} = \nabla_{\pi'} \log \pi_{\pi'}(a|s)(R - V_{\pi'}(s))$.

An unconstrained meta-update can lead to α being negative. Clipping α to 0 after

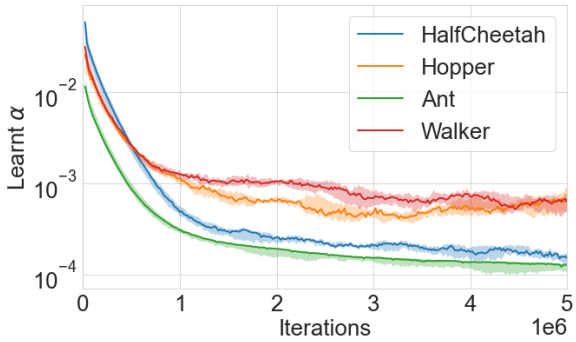


Figure 5.3: Schedule for the learning rates learnt by HOOF. Refer to Equation (5.7).

each meta update is not feasible since it leads to the situation where the policy does not update at all. Hence a log transform was used instead to ensure $\alpha > 0$.

We are faced with two more issues: what should the learning rate be initialised to (α_0), and what should the meta learning rate β be set to? Like all gradient based methods, the performance of meta gradients can be sensitive to the choices of these two hyperparamters. When we set α_0 to the OpenAI baselines default setting and β to 0.001 as per Xu et al. (2018), A2C fails to learn at all. Thus, we had to run a grid search over (α_0, β) to find the optimal settings across these hyperparameters. In Figure 5.2 we plot the best run from this grid search.

The returns after 5 millions timesteps for each setting of (α_0, β) on the grid is given in Table 5.1. Note that very few settings of the hyperparameters can match the performance of HOOF, while some settings of (α_0, β) can lead to the algorithm failing to learn at all. Setting $\alpha_0 = 1e-3$, which is closest to the OpenAI Baselines default setting, and $\beta = 1e-3$ as was used by Xu et al. (2018) in their experiments leads to performance well below that of HOOF, or even Baseline A2C.

It is also important to note that HOOF only requires 1 training run of samples (i.e. 5 million timesteps) while the grid search over the hyperparameters means that meta-gradients requires 36x samples to be able to match HOOF.

Table 5.1: Results of grid search over meta-gradients hyperparameters. * denotes algorithm diverged with returns $< -10^5$. Settings with returns greater than HOOF returns are shown in bold.

HalfCheetah		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-10,972	-1,230	*	*	*	-498	1523
	$1e-3$	-7,137	-221	468	1,080	**1,568**	1,272	
	$1e-4$	*	-245	313	441	-223	86	
	$1e-5$	*	-247	324	-499	-515	-518	
	$1e-6$	-641	-224	-404	-618	-616	-631	
	$1e-7$	-643	-351	-611	-633	-638	-639	

Hopper		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-2,950	-13,271	-808	-1,845	87	103	350
	$1e-3$	-12,045	-508	-801	54	**378**	**368**	
	$1e-4$	-20,086	68	67	225	215	236	
	$1e-5$	-3,309	70	65	67	61	61	
	$1e-6$	35	67	63	64	64	50	
	$1e-7$	-7,793	72	64	54	20	18	

Ant		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	*	393	*	*	*	*	952
	$1e-3$	*	761	752	950	926	884	
	$1e-4$	-47,393	-156	687	672	666	655	
	$1e-5$	*	375	588	-595	-739	-692	
	$1e-6$	*	283	373	-1,081	-1,073	-1,006	
	$1e-7$	-1,257	-514	-361	-1,017	-1,042	-964	

Walker		β						HOOF
		$1e-2$	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	
α_0	$1e-2$	-10,316	-2,922	109	294	176	159	467
	$1e-3$	-1,383	-2,636	28	445	**492**	**485**	
	$1e-4$	-931	-153	31	220	133	124	
	$1e-5$	-4,732	-117	125	112	116	121	
	$1e-6$	-47,222	-3,005	137	113	39	12	
	$1e-7$	-774	-22	111	113	2	0	

Table 5.2: Performance of HOOF with different values of the KL constraint (ϵ parameter). The results show that the performance is relatively robust to the setting of ϵ.

KL constraint (ϵ)	0.01	0.02	0.03	0.04	0.05	0.06	0.07
HalfCheetah	1,203	1,451	1,524	1,325	1,388	1301	1504
Hopper	359	358	350	362	359	370	365
Ant	916	942	952	957	971	963	969
Walker	466	415	467	475	456	402	457

5.2.2.2 Robustness to the Choice of KL Constraint

To show that HOOF's performance is robust to ϵ, its own hyperparameter quantifying the KL constraint, we repeated our experiments with different values of ϵ. The results presented in Table 5.2 show that HOOF's performance is stable across different values of this parameter. This is not surprising – the sole purpose of the constraint is to ensure that the WIS estimates remain viable.

5.2.2.3 Robustness to the Choice of Optimiser

Our implementation of A2C is based on the OpenAI baselines implementation, which uses RMSProp as the default optimiser. To check how robust HOOF's performance is to the choice of the optimiser, we ran both Baseline A2C and HOOF with SGD instead. The learning curves presented in Figure 5.4 shows that in this case HOOF's performance is far better than that of Baseline A2C which fails to learn at all.

5.2.2.4 Performance of HOOF without a KL Constraint

Figure 5.5 shows that without a KL constraint HOOF does not converge, which confirms that we need to constrain policy updates so that WIS estimates remain sound.

Table 5.3: Comparison of sample efficiency of HOOF over grid search with subsampled grids of different sizes.

	HOOF	Max return over subsampled grid of size			
	Returns	1	2	5	10
HalfCheetah	702	-558	-241	113	354
Hopper	321	109	165	240	287
Ant	675	-7561	-272	177	476
Walker	175	99	153	224	279

5.2.2.5 Sample Efficiency vs. Grid Search

Finally, to ascertain the sample efficiency of HOOF relative to grid search, we perform a benchmarking exercise. We used HOOF to learn both the learning rate and the entropy coefficient (c_2 in (5.7)). We split the search bounds for these across a grid with 11x11 points and ran A2C for each setting on the grid. For computational reasons we set the budget for each training run to 1 million timesteps. Given a budget of n training runs, we randomly subsample n points from the grid (without replacement) and note the best return. We repeat this 1000 times to get an estimate of the expected best return of the grid search with a budget of n training runs. The results presented in Table 5.3 compares the returns of HOOF to that of the expected best return for grid search with different training budgets. The performance of grid search is much worse than that of HOOF with the same budget (i.e., only 1 training run). The results show that grid search can take more than 10 times as many samples to match HOOF's performance.

5.2.2.6 Implementation Details

Our codebase for the A2C experiments is based on OpenAI Baselines (Dhariwal et al., 2017) implementation of A2C and uses their default hyperparameters. Experiments involving HOOF use the same hyperparameters apart from those that are learnt by HOOF. All hyperparameters are presented in Table 5.4.

For HOOF, α_{UB}, the upper bound of the search space for α was dynamically

Table 5.4: Hyperparameters used for A2C experiments.

Hyperparameter	Value
Number of environments (num_envs)	40
Timesteps per worker (nsteps)	5
Total environment steps	5e6
Discounting γ	0.99
Max gradient norm	0.5
Optimiser	RMSProp
– α	0.99
– ϵ	1e-5
Policy	MLP
– Number of fully connected layers	2
– Number of units per layer	64
– Activation	tanh
Default settings for Baseline A2C (and HOOF and meta gradients if not learnt)	
– Initial learning rate	7e-4
– Learning rate schedule	linear annealing
– Value function cost weight (c_1)	0.5
– Entropy cost weight (c_2)	0.01
Grid search over (α, c_2) with A2C	
– Grid settings for α	$0.01 \times 10^{-0.5 \times \{0,1,..,10\}}$
– Grid settings for c_2	$0.05 \times \{0.0, 0.1, .., 1.0\}$
HOOF specific hyperparameters	
– Initial search bounds for α	[0,1e-2]
– Number of random samples for α	100
– Search bounds for c_2 (for grid search experiment)	[0,0.2]
– Number of random samples for c_2 (for grid search experiment)	50

updated at each iteration based on the following: if no candidates violate the KL constraint, $\alpha_{UB} \leftarrow 1.25\alpha_{UB}$. If more than 80% of the candidates violate the KL constraint, $\alpha_{UB} \leftarrow \alpha_{UB}/1.25$.

5.2.3 HOOF with TNPG

TRPO has been shown to perform better than TNPG in continuous control tasks (Schulman et al., 2015), a result attributed to stricter enforcement of the KL constraint. However, in this section, we show that stricter enforcement of the KL constraint

becomes unnecessary once we properly adapt TNPG's learning rate. To do so, we apply HOOF to learn $(\delta, \gamma, \lambda)$ of TNPG ('HOOF-TNPG'), and compare it with TRPO with the OpenAI Baselines default settings of $(\epsilon = 0.01, \gamma = 0.99, \lambda = 0.98)$ ('Baseline TRPO').

Figure 5.6 shows the learning curves of HOOF-TNPG and the Baseline TRPO. HOOF-TNPG learns much faster, and outperforms Baseline TRPO in all environments except for Walker where there's no significant difference. Figure 5.7 presents the learnt $(\delta, \gamma, \lambda)$. The results show that different KL constraints and GAE hyperparameters are needed for different domains. We could not compare with meta-gradients as the objective function is not differentiable, as discussed in Section 6.4. We also could not perform a comparison against grid search similar to the one in Section 5.2.2 as the computational burden of performing a grid search over three hyperparameters was too large.

5.2.3.1 Importance of Learning (γ, λ) Conditional Value Functions

In Figure 5.8 we compare the performance of HOOF ('HOOF-TNPG') with that of HOOF without (γ, λ) conditional value functions ('HOOF-no-(γ, λ)'). Clearly the conditioning is key to good performance. This is because the value is highly dependent on (γ, λ) which changes throughout training.

5.2.4 Implementation Details

Our codebase for the TNPG experiments is based on OpenAI Baselines (Dhariwal et al., 2017) implementation of TRPO and uses their default hyperparameters. Experiments involving HOOF use the same hyperparameters apart from those that are learnt by HOOF. All hyperparameters are presented in Table 5.5.

Table 5.5: Hyperparameters used for TNPG experiments.

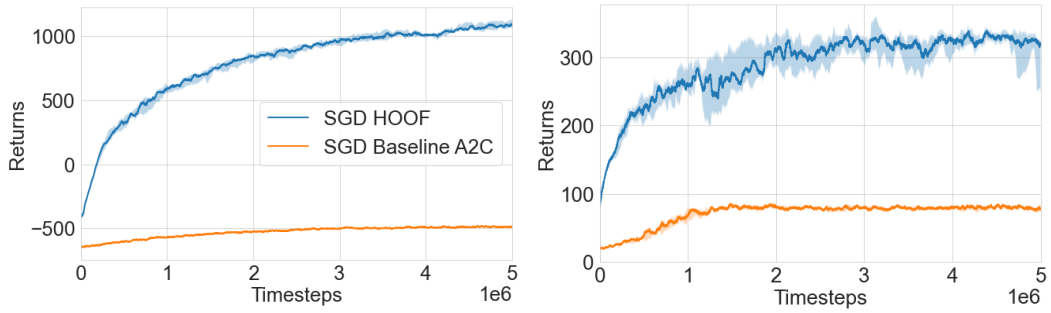
Total environment steps	5e6
Timesteps per iteration	10,000
Policy	MLP
– Number of fully connected layers	2
– Number of units per layer	64
– Activation	tanh
Baseline TRPO	
– KL constraint	0.01
– Discounting γ	0.99
– GAE-λ	0.98
HOOF specific hyperparameters	
– Search bounds for ϵ	[0.001, 0.1]
– Search bounds for (γ)	[0.85, 1]
– Search bounds for (λ)	[0.85, 1]
– Number of random samples for ϵ	50
– Number of random samples for (γ, λ)	50

5.3 Discussion

The performance of a policy gradient method is highly dependent on its hyperparameters. However, methods typically used to tune these hyperparameters are sample inefficient, computationally expensive, and learn only a fixed setting of the hyperparameters. In this chapter we presented HOOF, a sample efficient method that automatically learns a schedule for the learning rate and GAE hyperparameters of policy gradient methods without requiring multiple training runs. We believe that this, combined with its simplicity and ease of implementation, makes HOOF a compelling method for optimising policy gradient hyperparameters.

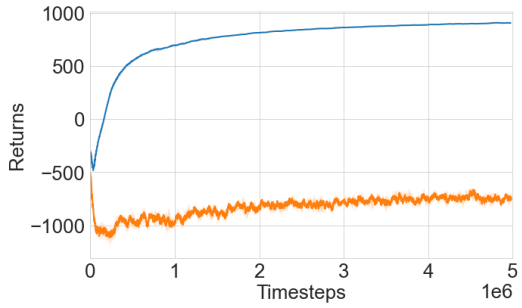
While we have presented HOOF as a method to learn the hyperparameters of a policy gradient algorithm, the underlying principles are far more general. For example, one could compute a distribution for the gradient and generate candidate policies by sampling from that distribution, instead of just using the point estimate of the

gradient. It has also been hypothesised that state/action dependent discount factors might help speed up learning (White, 2017; Fedus et al., 2019). This could be achieved by using HOOF to learn the parameters of a function that maps the states/actions to the discount factors.

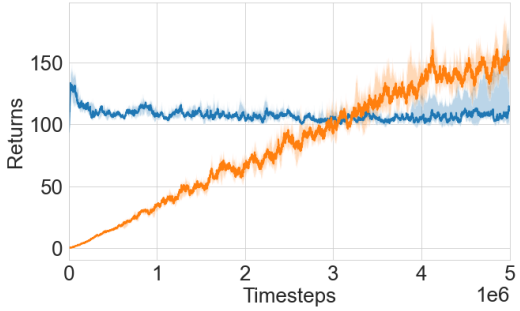


(a) HalfCheetah

(b) Hopper

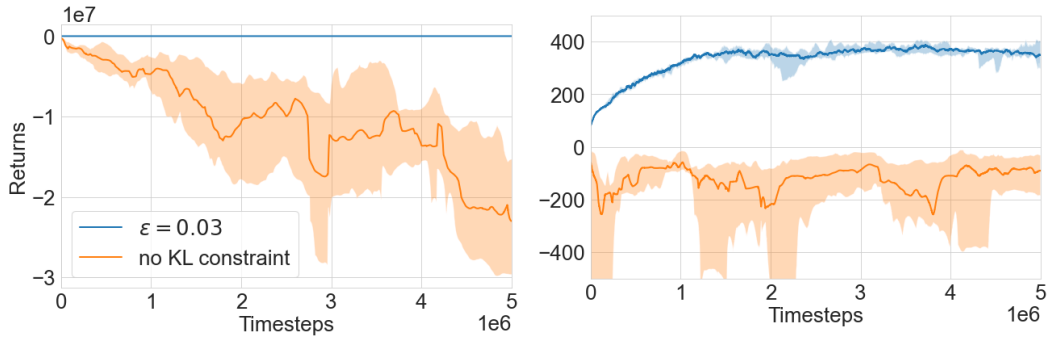


(c) Ant



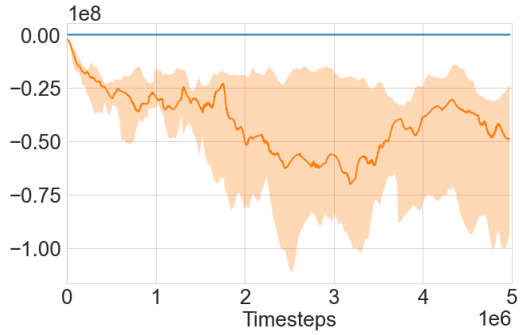
(d) Walker2d

Figure 5.4: Comparison of the performance of HOOF with $\epsilon = 0.03$ and Baseline A2C where the optimiser is SGD for both (instead of RMSProp).

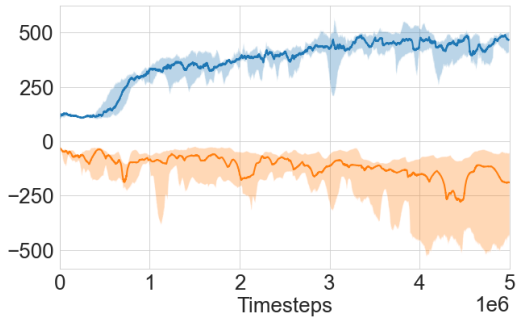


(a) HalfCheetah

(b) Hopper

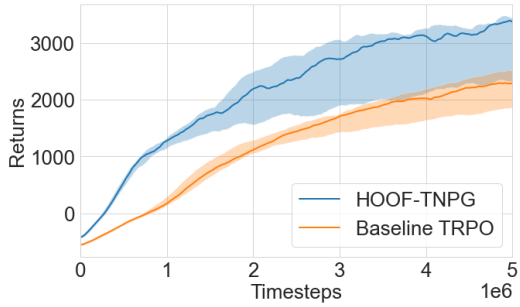


(c) Ant

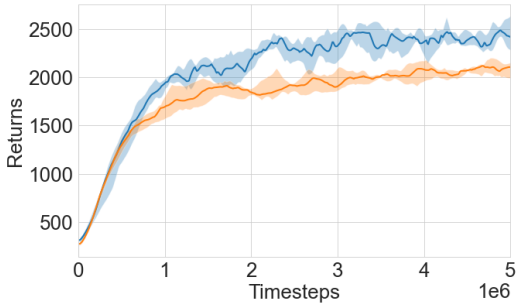


(d) Walker2d

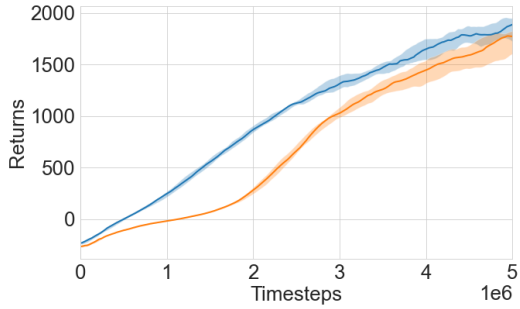
Figure 5.5: Comparison of the performance of HOOF with $\epsilon = 0.03$ and HOOF without any KL constraint.



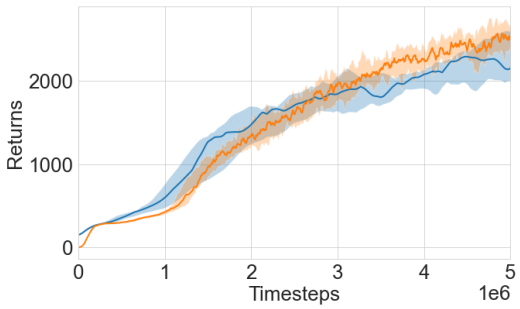
(a) HalfCheetah



(b) Hopper

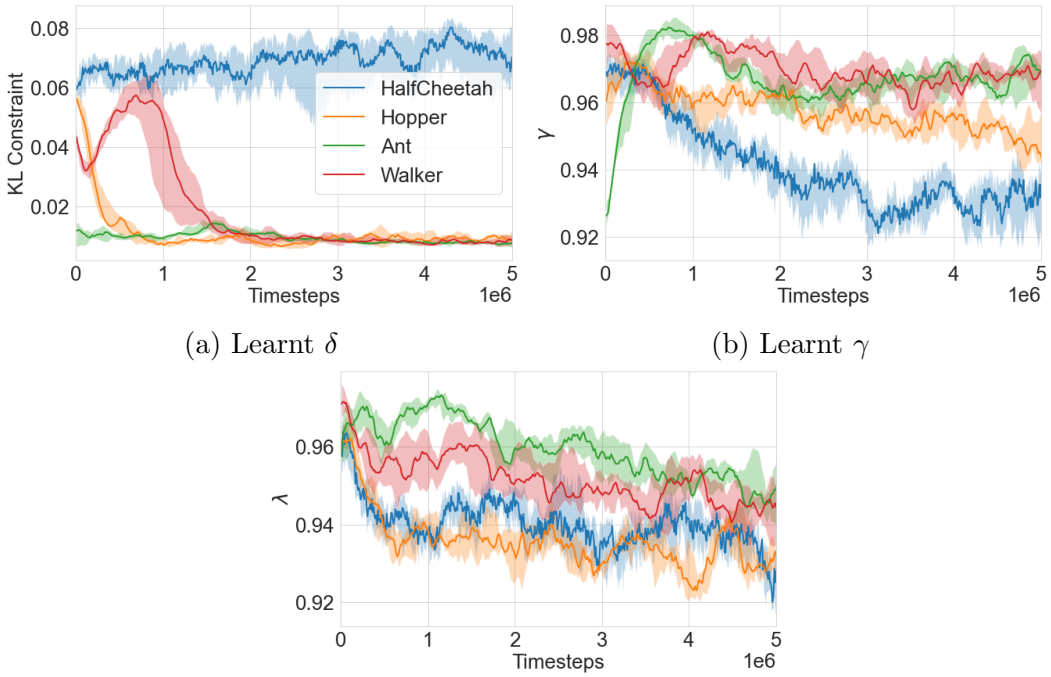


(c) Ant



(d) Walker

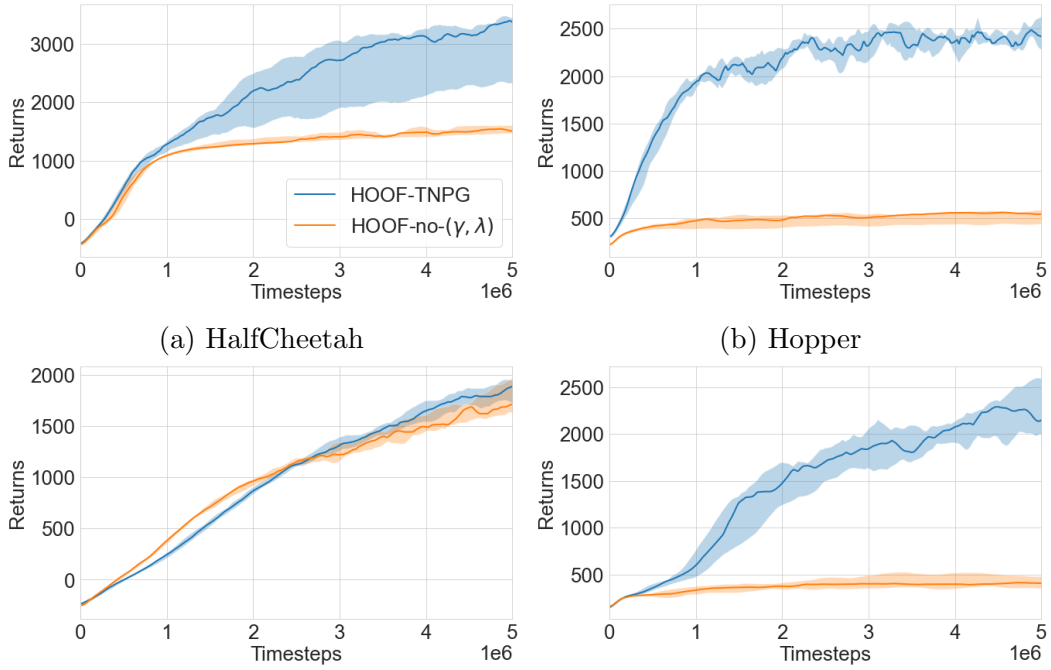
Figure 5.6: Performance of HOOF-TNPG vs TRPO baselines.



(a) Learnt δ (b) Learnt γ

(c) Learnt λ

Figure 5.7: Hyperparameters learnt by HOOF-TNPG for HalfCheetah, Hopper, and Walker.



(a) HalfCheetah (b) Hopper

(c) Ant (d) Walker

Figure 5.8: Performance of HOOF without (γ, λ) conditioned value functions: Not learning (γ, λ) conditioned value functions leads to significant reduction in performance in all environments except Ant.

Chapter 6

Related Work

In this chapter we present some of the methods proposed in the literature for optimising problems with environment variables, learning policies robust to SREs (related to ALOQ and FPO in Chapters 3 and 4), learning policies for physical systems by bridging the reality gap (related to TALOQ in Section 3.3), and hyperparameter optimisation for reinforcement learning (related to HOOF in Chapter 5).

6.1 Optimising in the presence of environment variables

Williams et al. (2000) consider a problem setting they call the *design of computer experiments* that is similar to our setting, but does not specifically consider SREs. Their proposed GP-based approach marginalises out the environment variable by alternating between BO and BQ. However, unlike ALOQ, their method is based on the EI acquisition function, which makes it computationally expensive for reasons discussed in Section 3.2, and is applicable only to discrete environment variables. We include their method as a baseline in our experiments. Our results (in Section 3.5.2) show that their method is unsuitable for settings with SREs. Further, their method fails even to outperform a baseline that randomly samples the environment variable at each step.

Toscano-Palmerin and Frazier (2018) propose a BO based method for the same setting. They show that their method, Bayesian Quadrature Optimisation, which uses a value of information criterion to select points for evaluation, is more efficient than that of Williams et al. (2000). However, like Williams et al. (2000) they do not explicitly consider SREs, and their method likely to fail without the benefit of the machinery developed in ALOQ (discussed in detail in Section 3.2) to specifically address the issues raised by SREs.

Krause and Ong (2011) also address optimising performance in the presence of environment variables. However, they address a fundamentally different contextual bandit setting in which the learned policy conditions on the observed environment variable. Similarly, as discussed in 4.1, Kupcsik et al. (2013) and Fabisch and Metzen (2014) consider the problem of learning policies given a context observable to the agent. While the setting of the environment variable can be considered to be a context, in our setting the environment variable is unobservable to the agent, and our objective is to learn a policy that performs optimally under a specified distribution of the environment variable rather than for some observed context.

6.2 Learning Policies Robust to SREs

Frank et al. (2008) also consider the problems posed by SREs. In particular, they propose an approach based on importance sampling (IS) for efficiently evaluating policies whose expected value may be substantially affected by rare events. While their approach is based on *temporal difference* (TD) methods, we take a BO-based policy search approach. Unlike TD methods, BO is well suited to settings in which sample efficiency is paramount and/or where assumptions (e.g., the Markov property) that underlie TD methods cannot be verified. More importantly, they assume prior knowledge of the SREs, such that they can directly alter the probability of such events during policy evaluation. By contrast, a key strength of ALOQ is that it requires

only that a set of environment variables can be controlled in the simulator, without assuming any prior knowledge about whether SREs exist, or about the settings of the environment variables that might trigger them.

Ciosek and Whiteson (2017) also proposed an IS-based algorithm, OFFER, where the setting of the environment variable is gradually changed based on observed trials. Since OFFER, as a policy gradient method, relies on properties of the Markov Decision Process such as the Markov property, it suffers from all the disadvantages mentioned earlier. Also, it can lead to unstable IS estimates if the environment is modified in other ways than by changing the initial state.

Rajeswaran et al. (2017a) propose EPOpt for finding robust policies by training on different versions of a simulator. First, multiple instances of the simulator are generated by drawing a random sample of the simulator parameter settings. Trajectories are then sampled from each of these instances and used by a batch policy optimisation algorithm (e.g., TRPO (Schulman et al., 2015)). While ALOQ finds a risk-neutral solution, EPOpt finds a risk-averse one that maximises the *conditional value at risk* (CVaR) by feeding the policy optimisation only the sampled trajectories whose returns are lower than the CVaR. In a risk-neutral setting, EPOpt reduces to the underlying policy optimisation algorithm with trajectories randomly sampled from different instances of the simulator. This approach will not see SREs often enough to learn an appropriate response, as we demonstrate in our experiments.

Pinto et al. (2017) also suggest a method to address the problem of finding robust policies. Their method learns a policy by training in a simulator that is adversarial in nature, i.e., the simulator settings are dynamically chosen to minimise the returns of the policy. This method requires significant prior knowledge to be able to set the simulator settings such that it provides just the right amount of challenge to the policy. Furthermore, it does not consider any settings with SREs.

6.3 Learning policies for physical systems

The methods suggested for learning policies for physical systems address the need for sample efficiency by a variety of means: they can use be model based, use sample efficient optimisation techniques like BO, and incorporate the use of simulators in the learning loop. Here we discuss some of these methods.

A key challenge in working with mobile robots is designing policies for locomotion. Even if a good parametrisation of the locomotion controller is available, learning the optimal parameters in a sample efficient manner can be difficult due to the presence of local optima. This issue was studied for a Sony AIBO, a quadruped robot used in the RoboCup Legged League by Lizotte et al. (2007), where they used BO to perform gait optimization. Furthermore, Calandra et al. (2015) showed that BO can be used to efficiently find good gait parameters for bipedal robots.

Levine et al. (2016) and Chebotar et al. (2017) use *guided policy search* (GPS) (Levine and Koltun, 2013) to learn a deep neural net based visuomotor policy for various object manipulation tasks. Yahya et al. (2017) propose a distributed, asynchronous version of GPS to accelerate training and improve generalisation. Rusu et al. (2016) use progressive networks to transfer policies for a Jaco robot arm learnt in simulation. Viereck et al. (2017) show that simulated depth images can be used to learn policies for grasping objects that transfer well to the physical robot. Tobin et al. (2017) and James et al. (2017) ensure transfer of policies to reality by generating a diverse set of training data in simulation by randomising the visual characteristics of objects. While GPS has proved to be highly sample efficient, it are unsuitable for our setting. A straightforward application of it ignores the setting of the environment variable, and treats its effect on the transition and rewards as noise. As such the underlying linear-quadratic assumptions of the transition dynamics and rewards are going to be grossly violated as these effects can be very large due to the presence of SREs.

Within evolutionary robotics, one approach to bridge the reality gap is to minimise

the gap between the simulator and reality. Miglino et al. (1996) evolve controllers for a Khepera robot in simulation by learning an accurate model of the responses of its infra red sensors to real world objects. Moeckel et al. (2013) learn the parameters of an ODE-based simulator to maximise the *similarity ratio* of gaits observed in simulation and on the robot. Zagal and Ruiz-del Solar (2007) present the *back to reality* algorithm that co-evolves a simulator and a controller to minimise the difference in fitness observed in simulation and reality. The transferability approach Koos et al. (2013) follows an alternative idea: instead of correcting the simulator predictions, the algorithm learns a model of the limits of the simulator, that is, when the predictions are accurate and when they are are not; the optimization/search algorithm can then use this model as a soft constraint to keep the policy search algorithm focused on policies for which the simulator is accurate.

Another approach is to learn a controller in simulation and then fine tune it on the real robot. For example, Lipson and Pollack (2000) learn a design of a robot in simulation, which is then 3D-printed and evolved for a few more steps for fine tuning. Cully et al. (2015) learn a low dimensional behaviour map of diverse controllers in simulation and then perform BO to find the optimal controller on the robot. Inspired by the kernel structure proposed in Poloczek et al. (2017), Marco et al. (2017) develop an entropy search based BO algorithm that together with selecting a policy for evaluation, also selects whether to evaluate the policy on the simulator or the physical system at each iteration.

PILCO (Deisenroth and Rasmussen, 2011) uses data from trials on the physical system to build a model of the environment, and has been shown to achieve remarkable sample efficiency in robot control. Kamthe and Deisenroth (2018) propose a similar approach that maintains data efficiency, while being less computationally intensive and better able to handle state and control constraints. A similar approach, Black-DROPS (Chatzilygeroudis et al., 2017), is a purely black-box model-based policy

search algorithm for robotics and can take advantage of parallel computations. Black-DROPS achieves similar data-efficiency to PILCO. However, like PILCO it scales poorly as the dimensionality of the state/action space increases. A recent extension of Black-DROPS (Chatzilygeroudis and Mouret, 2018) combines nonparametric model learning (using GPs) and model identification in order to scale up to high-dimensional systems.

These methods superficially resemble ALOQ in its use of GPs but the key difference is that they are model based approaches where the GP models the transition dynamics, while ALOQ is model free with the GP modelling the returns as a function of the policy and environment variable. These methods are fundamentally ill suited to our setting. First, they assume that the transition dynamics are Gaussian and can be learned with a few hundred observed transitions, which is often infeasible in more complex environments (i.e., they scale poorly as the dimensionality of the state/action space increases). Second, during training they would be unable to learn a good transition model that conditions on the environment variable since SREs might not be observed often enough under random sampling. Even if they were to learn such a transition model, they would still have to learn a policy that marginalises out the environment variable since it is not observable outside of training. In this case, propagating the uncertainties through the trajectory during training would lead to major violations of the Gaussian assumption when the environment variables can cause SREs.

6.4 Hyperparameter Optimisation for Reinforcement Learning

Most hyperparameter search methods can be broadly classified into sequential search, parallel search, and gradient based methods.

Sequential search methods perform a training run with some candidate hyperparameters, and use the results to inform the choice of the next set of hyperparameters

for evaluation. BO is a sample efficient global optimisation framework that models performance as a function of the hyperparameters, and is especially suited for sequential search as each training run is expensive. After each training run BO uses the observed performance to update the model in a Bayesian way, which then informs the choice of the next set of hyperparameters for evaluation. Several modifications have been suggested to further reduce the number of evaluations required: input warping (Snoek et al., 2014) to address non-stationary fitness landscapes; freeze-thaw BO (Swersky et al., 2014) to decide whether a new training run should be started and the current one discontinued based on interim performance; transferring knowledge about hyperparameters across similar tasks (Swersky et al., 2013); and modelling training time as a function of dataset size (Klein et al., 2016). To further speed up the wall clock time, some BO based methods use a hybrid mode wherein batches of hyperparameter settings are evaluated in parallel (Contal et al., 2013; Desautels et al., 2014; Shah and Ghahramani, 2015; Wang et al., 2016; Kandasamy et al., 2018).

By contrast, parallel search methods like grid search and random search run multiple training runs with different hyperparameter settings in parallel to reduce wall clock time, but require more parallel computational resources. These methods are easy to implement, and have been shown to perform well (Bergstra et al., 2011; Bergstra and Bengio, 2012).

Both sequential and parallel search suffer from two key disadvantages. First, they require performing multiple training runs to identify good hyperparameters. Not only is this computationally inefficient, but when applied to RL, also sample inefficient as each run requires fresh interactions with the environment. Second, these methods learn fixed values for the hyperparameters that are used throughout training instead of a schedule, which can lead to suboptimal performance (Luketina et al., 2016; Jaderberg et al., 2017; Xu et al., 2018).

PBT (Jaderberg et al., 2017) is a hybrid of random and sequential search, with

the added benefit of adapting hyperparameters during training. It starts by training a population of hyperparameters which are then updated periodically to further explore promising hyperparameter settings. However, by requiring multiple training runs, it inherits the sample inefficiency of random search.

HOOF is much more sample efficient because it requires no more interactions with the environment than those gathered by the underlying policy gradient method for one training run. Consequently, it is also far more computationally efficient. However, while HOOF can only optimise hyperparameters that directly affect the policy update, these methods can tune other hyperparameters, e.g., policy architecture, and batch size. Combining these complementary strengths in an interesting topic for future work.

Gradient based methods (Sutton, 1992; Bengio, 2000; Luketina et al., 2016; Pedregosa, 2016; Xu et al., 2018) adapt the hyperparameters by performing gradient descent on the policy gradient update function with respect to the hyperparameters. This raises the fundamental problem that the update function needs to be differentiable. For example, the update function for TRPO uses conjugate gradient to approximate $I(\pi)^{-1}g$, performs a backtracking line search to enforce the KL constraint, and introduces a surrogate improvement constraint, which introduce discontinuities in the update and makes it non-differentiable.

A second major disadvantage of these methods is that they introduce their own set of hyperparameters, which can make them sample inefficient if they require tuning. For example, the meta-gradient estimates can have high variance, which in turn significantly affects performance. To address this, the objective function of meta-gradients introduces reference (γ', λ') hyperparameters to trade off bias and variance. As a result, its performance can be sensitive to these, as the experimental results of Xu et al. (2018) show. Furthermore, gradient based methods tend to be highly sensitive to the setting of the learning rate, and these methods introduce their own

learning rate hyperparameter for the meta learner which requires tuning, as we show in our experiments. As a gradient-free method, HOOF does not require a differentiable objective and, while it introduces a few hyperparameters of its own, these do not affect sample efficiency, as mentioned in Section 5.1.3.

Other work on non-gradient based methods includes that of Kearns and Singh (2000), who derive a theoretical schedule for the TD(λ) hyperparameter that they show is better than any fixed value. Downey et al. (2010) learn a schedule for TD(λ) using a Bayesian approach. White and White (2016) greedily adapt the TD(λ) hyperparameter as a function of state. Unlike HOOF, these methods can only be applied to TD(λ) and, in the case of Kearns and Singh (2000), are not compatible with function approximation.

Chapter 7

Conclusion

In the last few years the field of RL has seen many new algorithms being proposed that have led to significant success in a number of challenging tasks. However, there still remains significant challenges in applying RL to solve real world problems, perhaps the biggest of which is sample efficiency. In this thesis, we focussed on two challenges faced by existing RL algorithms - their reliance on Monte Carlo sampling makes them highly sample inefficient when applied to environments with SREs, and even in environments without SREs, their performance can be highly sensitive to the setting of certain hyperparameters.

In Chapters 3 and 4 we addressed the problem of sample efficient learning in the presence of SREs. Our first algorithm, ALOQ, is a direct policy search algorithm that uses BO and BQ to actively search over settings of the environment variable. We presented an extension, TALOQ, that can be used to leverage an imperfect simulator to enhance the sample efficiency while learning, while at the same time bridging the reality gap that can appear when learning in simulation. The biggest drawback of ALOQ is that it cannot be used to learn high dimensional policies, for example, large neural nets. Our algorithm, FPO, addressed this by using a policy gradient method for the policy updates, while still performing active search using BO. While this made FPO quite flexible in terms of the choice underlying policy gradient method, it also brought with it the disadvantages of these methods, namely, sensitivity to

their hyperparameters, which in turn has a significant impact on sample efficiency. We addressed this in Chapter 5 through HOOF - an algorithm that can learn the hyperparameters with unparalleled sample efficiency, while being computationally efficient and simple to implement.

While the algorithms presented in this thesis go a long way towards addressing the robustness of RL algorithms to SREs and their own hyperparameters, they also open up a number of promising paths to be explored for future work.

7.1 Alternative Formulation for Settings with SREs

Both ALOQ and FPO view the problem of SREs through the lens of single objective RL, i.e. these rare events have some (significantly different) reward and our goal is to find a policy that maximises the expected return. While this has the advantage of learning a policy that automatically performs a trades off between the reward and probability of experiencing a rare event, it requires that a lot of care be taken in the design of the reward function, which is often difficult to determine a priori. For example, when learning to fly a helicopter, a reward function that is based on the squared distance between the path and the current position seems to be a sensible choice. However, it is not immediately apparent what the reward function for a crash (which is an SRE) should be.

An alternative formulation would be frame this as a multi-objective RL problem. In this case, rather than carefully crafting a reward function for SREs, we would require the policy to have a probability of experiencing SREs associated with it, together with its return. Our goal would be find a set of policies that lie on the Pareto frontier of returns vs probability of experiencing SREs. However, being able to accurately determine the probability of experiencing SREs for any given policy is going to be a challenging task by itself, and further work needs to be done to explore the viability of this formulation.

7.2 Expanding the Scope of Hyperparameter Optimisation Algorithms

While we have shown HOOF to be a highly sample efficient algorithm for learning the hyperparameters of policy gradient algorithms, it has certain limitations. First, it cannot be used to learn hyperparameters that do not directly affect the gradient at each iteration. For example, it cannot learn the batch size hyperparameter, and value based methods such as DQN are entirely out of scope. Second, the off-policy evaluation using importance sampling makes it only applicable to stochastic policies. Finally, HOOF is rather myopic since it selects the hyperparameters at each iteration based on a one step improvement objective. While our experiments show that this works well in practice, there is no reason to believe that this is optimal.

An interesting research direction would involve designing hyperparameter optimisation algorithms that are more widely applicable. Furthermore, algorithms that incorporate non-myopic objectives could lead to further performance gains, however this would require developing radically different design principles compared to HOOF.

Bibliography

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*.

Bartz-Beielstein, T., Lasarczyk, C. W. G., and Preuss, M. (2005). Sequential parameter optimization. In *IEEE Congress on Evolutionary Computation*.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*.

Bengio, Y. (2000). Gradient-based optimization of hyperparameters. *Neural computation*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*.

Bochner, S. (2003). *Lectures on Fourier Integrals: With an Author's Supplement on Monotonic Functions, Stieltjes Integrals, and Harmonic Analysis*.

Briol, F.-X., Oates, C. J., Girolami, M., Osborne, M. A., and Sejdinovic, D. (2015). Probabilistic Integration: A Role for Statisticians in Numerical Analysis? *ArXiv e-prints*.

Brochu, E., Cora, V. M., and de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv.org.

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

Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. (2015). Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*.

Chatzilygeroudis, K. and Mouret, J.-B. (2018). Using Parameterized Black-Box Priors to Scale Up Model-Based Policy Search for Robotics. In *International Conference on Robotics and Automation*.

Chatzilygeroudis, K., Rama, R., Kaushik, R., Goepp, D., Vassiliades, V., and Mouret, J.-B. (2017). Black-Box Data-efficient Policy Search for Robotics. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Chebotar, Y., Kalakrishnan, M., Yahya, A., Li, A., Schaal, S., and Levine, S. (2017). Path integral guided policy search. In *IEEE International Conference on Robotics and Automation*.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. (2018). Bayesian optimization in alphago. *CoRR*, abs/1812.06855.

Ciosek, K. and Whiteson, S. (2017). Offer: Off-environment reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Contal, E., Buffoni, D., Robicquet, A., and Vayatis, N. (2013). Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Cox, D. D. and John, S. (1992). A statistical method for global optimization. In *IEEE International Conference on Systems, Man and Cybernetics*.

Cox, D. D. and John, S. (1997). SDO: A statistical method for global optimization. In *in Multidisciplinary Design Optimization: State-of-the-Art*.

Cully, A., Clune, J., Tarapore, D., and Mouret, J.-B. (2015). Robots that can adapt like animals. *Nature*.

Cully, A. and Mouret, J.-B. (2015). Evolving a behavioral repertoire for a walking robot. *Evolutionary Computation*.

Deisenroth, M. P. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning*.

Desautels, T., Krause, A., and Burdick, J. W. (2014). Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*.

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

Downey, C., Sanner, S., et al. (2010). Temporal difference bayesian model averaging: A bayesian perspective on adapting lambda. In *International Conference on Machine Learning*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016a). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016b). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*.

Fabisch, A. and Metzen, J. H. (2014). Active contextual policy search. *The Journal of Machine Learning Research*.

Farquhar, G., Rocktaschel, T., Igl, M., and Whiteson, S. (2018). Treeqn and atreec: Differentiable tree-structured models for deep reinforcement learning. In *International Conference on Learning Representations*.

Fedus, W., Gelada, C., Bengio, Y., Bellemare, M. G., and Larochelle, H. (2019). Hyperbolic discounting and learning over multiple horizons. *arXiv preprint arXiv:1902.06865*.

Foerster, J., Nardelli, N., Farquhar, G., Torr, P., Kohli, P., and Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*.

François-Lavet, V., Fonteneau, R., and Ernst, D. (2015). How to discount deep reinforcement learning: Towards new dynamic strategies. In *NIPS 2015 Workshop on Deep Reinforcement Learning*.

Frank, J., Mannor, S., and Precup, D. (2008). Reinforcement learning in the presence of rare events. In *International Conference on Machine Learning*.

Gamble, C. and Gao, J. (2018). Safety-first ai for autonomous data centre cooling and industrial control.

Glynn, P. W. (1990). Likelihood ratio gradient estimation for stochastic systems. In *Communications of the ACM*.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018a). Deep reinforcement learning that matters. In *AAAI*.

Henderson, P., Romoff, J., and Pineau, J. (2018b). Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *CoRR*, abs/1810.02525.

Hennig, P., Osborne, M. A., and Girolami, M. (2015). Probabilistic numerics and uncertainty in computations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*.

Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*.

Hutter, F., Hoos, H. H., Leyton-Brown, K., and Murphy, K. P. (2009). An experimental investigation of model-based parameter optimisation: Spo and beyond. In *Conference on Genetic and Evolutionary Computation*.

Igl, M., Zintgraf, L. M., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A.,

Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*.

Jakobi, N., Husbands, P., and Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In Morán, F., Moreno, A., Merelo, J. J., and Chacón, P., editors, *Advances in Artificial Life*.

James, S., Davison, A. J., and Johns, E. (2017). Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task. In *Conference on Robot Learning*.

Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*.

Jones, D. R., Schonlau, M., and Welch, W. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*.

Kakade, S. (2001). A natural policy gradient. In *Neural Information Processing Systems*.

Kamthe, S. and Deisenroth, M. (2018). Data-efficient reinforcement learning with probabilistic model predictive control. In *International Conference on Artificial Intelligence and Statistics*.

Kanagawa, M., Sriperumbudur, B. K., and Fukumizu, K. (2016). Convergence guarantees for kernel-based quadrature rules in misspecified settings. *Neural Information Processing Systems*.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. (2018). Parallelised

bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*.

Kearns, M. J. and Singh, S. P. (2000). Bias-variance error bounds for temporal difference updates. In *Conference on Learning Theory*.

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J., Lam, V., Bewley, A., and Shah, A. (2018). Learning to drive in a day. *CoRR*, abs/1807.00412.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2016). Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*.

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

Koos, S., Mouret, J.-B., and Doncieux, S. (2013). The transferability approach: Crossing the reality gap in evolutionary robotics. *IEEE Transactions on Evolutionary Computation*.

Krause, A. and Ong, C. S. (2011). Contextual gaussian process bandit optimization. In *Neural Information Processing Systems*.

Kupcsik, A. G., Deisenroth, M. P., Peters, J., Neumann, G., et al. (2013). Data-efficient generalization of robot skills with contextual policy search. In *AAAI Conference on Artificial Intelligence*.

Lee, J. et al. (2018). DART: Dynamic Animation and Robotics Toolkit. *The Journal of Open Source Software*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*.

Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on International Conference on Machine Learning*.

Lipson, H. and Pollack, J. B. (2000). Automatic design and manufacture of robotic lifeforms. *Nature*.

Lizotte, D. J., Wang, T., Bowling, M., and Schuurmans, D. (2007). Automatic gait optimization with gaussian process regression. In *International Joint Conference on Artificial Intelligence*.

Luketina, J., Raiko, T., Berglund, M., and Greff, K. (2016). Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning*.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning*.

Malkomes, G., Schaff, C., and Garnett, R. (2016). Bayesian optimization for automated model selection. In *Neural Information Processing Systems*.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. In *Advances in Neural Information Processing Systems*.

Marco, A., Berkenkamp, F., Hennig, P., Schoellig, A. P., Krause, A., Schaal, S., and Trimpe, S. (2017). Virtual vs. real: Trading off simulations and physical experiments

in reinforcement learning with Bayesian optimization. In *International Conference on Robotics and Automation*.

Martinez-Cantin, R., de Freitas, N., Brochu, E., Castellanos, J., and Doucet, A. (2009). A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*.

Martinez-Cantin, R., de Freitas, N., Doucet, A., and Castellanos, J. (2007). Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*.

Miglino, O., Lund, H. H., and Nolfi, S. (1996). Evolving mobile robots in simulated and real environments. *Artificial Life*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

Močkus, J. (1975). On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*.

Moeckel, R., Perov, Y. N., Nguyen, A. T., Vespignani, M., Bonardi, S., Pouya, S., Sproewitz, A., van den Kieboom, J., Wilhelm, F., and Ijspeert, A. J. (2013). Gait optimization for roombots modular robots - matching simulation and reality. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. (2020). Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research*.

Mordatch, I., Lowrey, K., Andrew, G., Popovic, Z., and Todorov, E. V. (2015). Interactive control of diverse complex characters with neural networks. In *Neural Information Processing Systems*.

Mouret, J.-B. and Clune, J. (2015). Illuminating search spaces by mapping elites. *arxiv:1504.04909*.

Neal, R. (2000). Slice sampling. *Annals of Statistics*.

O'Hagan, A. (1987). Monte carlo is fundamentally unsound. *Journal of the Royal Statistical Society. Series D (The Statistician)*.

O'Hagan, A. (1991). Bayes-hermite quadrature. *Journal of Statistical Planning and Inference*.

Osborne, M., Garnett, R., Ghahramani, Z., Duvenaud, D. K., Roberts, S. J., and Rasmussen, C. E. (2012). Active learning of model evidence using bayesian quadrature. In *Neural Information Processing Systems*.

Owen, A. B. (2013). *Monte Carlo theory, methods and examples*.

Paul, S., Chatzilygeroudis, K., Ciosek, K., Mouret, J.-B., Osborne, M., and Whiteson, S. (2018). Alternating optimisation and quadrature for robust control. In *AAAI Conference on Artificial Intelligence*.

Paul, S., Kurin, V., and Whiteson, S. (2019a). Fast efficient hyperparameter tuning for policy gradient methods. In *Neural Information Processing Systems*.

Paul, S., Osborne, M. A., and Whiteson, S. (2019b). Fingerprint policy optimisation for robust reinforcement learning. In *International Conference on Machine Learning*.

Pautrat, R., Chatzilygeroudis, K., and Mouret, J.-B. (2018). Bayesian optimization with automatic prior selection for data-efficient direct policy search. In *IEEE International Conference on Robotics and Automation*.

Pedregosa, F. (2016). Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*.

Peters, J. and Schaal, S. (2006). Policy gradient methods for robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

Peters, J. and Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural networks*.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. (2017). Robust adversarial reinforcement learning. In *International Conference on Machine Learning*.

Poloczek, M., Wang, J., and Frazier, P. (2017). Multi-information source optimization. In *Neural Information Processing Systems*.

Pritchard, J. K., Seielstad, M. T., Perez-Lezaun, A., and Feldman, M. W. (1999). Population growth of human y chromosomes: a study of y chromosome microsatellites. *Molecular Biology and Evolution*.

Rajeswaran, A., Ghotra, S., Levine, S., and Ravindran, B. (2017a). EPOpt: Learning robust neural network policies using model ensembles. *International Conference on Learning Representations*.

Rajeswaran, A., Lowrey, K., Todorov, E. V., and Kakade, S. M. (2017b). Towards generalization and simplicity in continuous control. In *Neural Information Processing Systems*.

Rasmussen, C. E. and Ghahramani, Z. (2003). Bayesian monte carlo. *Neural Information Processing Systems*.

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Ross, S. M. (1990). *A Course in Simulation*. Prentice Hall PTR.

Ru, B., McLeod, M., Granziol, D., and Osborne, M. A. (2017). Fast Information-theoretic Bayesian Optimisation. In *International Conference on Machine Learning*.

Rubin, D. B. (1984). Bayesianly justifiable and relevant frequency calculations for the applied statistician. *The Annals of Statistics*.

Rubinstein, R. (1999). The cross-entropy method for combinatorial and continuous optimization. *Methodology and computing in applied probability*.

Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2016). Sim-to-real robot learning from pixels with progressive nets. *CoRR*, abs/1610.04286.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International Conference on Machine Learning*.

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

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

Settles, B. (2010). Active learning literature survey. *University of Wisconsin, Madison*.

Shah, A. and Ghahramani, Z. (2015). Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*.

Shewchuk, J. R. (1994). An introduction to the conjugate gradient method that even an idiot can understand. Technical report, Carnegie-Mellon University, Department Of Computer Science.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*.

Snoek, J., Swersky, K., Zemel, R., and Adams, R. (2014). Input warping for bayesian optimization of non-stationary functions. In *International Conference on International Conference on Machine Learning*.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: no regret and experimental design. In *International Conference on Machine Learning*.

Srinivasan, A. (2020). Reinforcement learning: The business use case.

Sutton, R. S. (1992). Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *AAAI*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*.

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task bayesian optimization. In *Advances in Neural Information Processing Systems*.

Swersky, K., Snoek, J., and Adams, R. P. (2014). Freeze-thaw bayesian optimization. *arXiv preprint arXiv:1406.3896*.

Tavaré, S., Balding, D. J., Griffiths, R. C., and Donnelly, P. (1997). Inferring coalescence times from dna sequence data. *Genetics*.

Thomas, P. S. (2015). *Safe Reinforcement Learning*. PhD thesis, University of Massachusetts.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*.

Tobin, J., Fong, R., Ray, A. K., Schneider, J., Zaremba, W., and Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *International Conference on Intelligent Robots and Systems*.

Toscano-Palmerin, S. and Frazier, P. I. (2018). Bayesian Optimization with Expensive Integrands. *ArXiv e-prints*.

Viereck, U., ten Pas, A., Saenko, K., and Jr., R. P. (2017). Learning a visuomotor controller for real world robotic grasping using easily simulated depth images. *CoRR*, abs/1706.04652.

Villemonteix, J., Vazquez, E., and Walter, E. (2009). An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*.

Wang, J., Clark, S. C., Liu, E., and Frazier, P. I. (2016). Parallel bayesian global optimization of expensive functions. *arXiv preprint arXiv:1602.05149*.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and De Freitas, N. (2013). Bayesian Optimization in High Dimensions via Random Embeddings. In *IJCAI*.

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

White, M. and White, A. (2016). A greedy approach to adapting the trace parameter

for temporal difference learning. In *International Conference on Autonomous Agents & Multiagent Systems*.

Williams, B. J., Santner, T. J., and Notz, W. I. (2000). Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*.

Wilson, J., Hutter, F., and Deisenroth, M. (2018). Maximizing acquisition functions for bayesian optimization. In *Advances in Neural Information Processing Systems*.

Xu, Z., van Hasselt, H., and Silver, D. (2018). Meta-gradient reinforcement learning.

Yahya, A., Li, A., Kalakrishnan, M., Chebotar, Y., and Levine, S. (2017). Collective robot reinforcement learning with distributed asynchronous guided policy search. In *International Conference on Intelligent Robots and Systems*.

Zagal, J. C. and Ruiz-del Solar, J. (2007). Combining simulation and reality in evolutionary robotics. *Journal of Intelligent and Robotic Systems*.