BUDGET-CONSTRAINED BAYESIAN OPTIMIZATION

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Eric Hans Lee

December 2020

BUDGET-CONSTRAINED BAYESIAN OPTIMIZATION

Eric Hans Lee, Ph.D.

Cornell University 2020

Global optimization, which seeks to identify a maximal or minimal point over a domain Ω, is a ubiquitous and well-studied problem in applied mathematics, computer science, statistics, operations research, and many other fields. The resulting body of global optimization research is vast, ranging from heuristic and metaheuristic-driven approaches such as evolutionary search to application-driven systems such as multi-level, multi-fidelity optimization of physical simulations. Global optimization's inherent hardness underlies this sheer variety of different methods; absent any additional assumptions, obtaining an efficient certificate of global optimality is not possible. Consequently, there are no agreed-upon methods that exhibit robust, all-around performance like there are in local optimization.

Data-driven algorithms and models, spurred by recent advances in cheap computing and flexible, open-source software, have been growing in popularity over recent years. Bayesian optimization (BO) is one such instance of this trend in global optimization. Using its past evaluations, BO builds a probabilistic model of the objective function to guide optimization, and selects the next iterate through an acquisition function, which scores each point in the optimization domain based on its potential to decrease the objective function. BO has been observed to converge faster than competing classes of global optimization algorithms. This *sample efficiency* is BO's key strength, and makes it ideal for optimizing objective functions that are expensive to evaluate and potentially

contaminated with noise. Key BO applications that meet these criteria include optimizing machine learning hyperparameters, calibrating physical simulations, and designing engineering systems

BO's performance is heavily influenced by its acquisition function, which must effectively balance exploration and exploitation to converge quickly. Default acquisition functions such as expected improvement are greedy in the sense that they ignore how the current iteration will affect future ones. Typically, the BO exploration-exploitation trade-off is expressed in the context of a one-step optimal process: for the next iteration, choose the point that balances information quantity and quality. However, if we possess a pre-specified iteration budget h, we might instead choose the point that balances information quantity and quality over the next h steps. This *non-myopic* approach is aware of the remaining iterations and can balance the exploration-exploitation trade-off correspondingly.

Non-myopic BO is the primary topic of this dissertation; we hope that making decisions according to a known iteration budget will improve upon the performance of classic BO, which is budget-agnostic. In Chapter 3, we draw from the rich literature of sequential decision making under uncertainty to reframe BO as a finite-horizon Markov decision process (MDP), in which one tries to maximize a cumulative reward over a fixed number of iterations. In Chapter 4, we develop efficient methods to solve finite-horizon MDPs in the context of BO through a combination of rollout and policy search, with supporting variance reduction techniques to further drive down their computational overhead. In Chapter 5, we extend our non-myopic acquisition functions to the cost-constrained BO setting, in which the objective function has non-uniform cost. Based off the work in Chapter 5, we develop a promising heuristic called CArBO in Chapter 6, which uses an early exploration, late exploitation strategy to perform much better than

competing methods in the batch BO setting. Finally, Chapter 7 covers BO with gradient information.

BIOGRAPHICAL SKETCH

Eric Hans Lee was born in Stockholm, Sweden on August 3rd, 1993 to Wanchen Lee and Lijun Lin, and emigrated with his parents to the States when he was very young. Swayed by frequent lobbying from his parents, who had very strong opinions on what a young Chinese immigrant ought to study, Eric had originally decided on medicine, until he took introductory calculus at the local community college and liked it. He attended University of California, Berkeley, starting in 2011 and studied applied mathematics, graduating in 2015 magna cum laude. While at Berkeley, Eric became interested in numerical methods, both from a summer REU at the Center for Discrete Mathematics and Theoretical Computer Science, and from a numerical linear algebra course taught by Professor James Demmel at Berkeley. Eric applied to and was accepted by Cornell University, where he started his Ph.D. in Computer Science in the fall of 2015. Eric immediately started working with his advisor, Professor David Bindel, first on applications of optimal control in electrical networks, and then on Bayesian optimization. While at Cornell, Eric did research internships at Cray, SigOpt, and then Amazon in Minneapolis, San Francisco, and Berlin, respectively.

Eric enjoys traveling, reading and writing fiction, and eating new things. He will join SigOpt post-graduation, where he hopes to continue contributing to the Bayesian optimization community at large.

To Michele, who mentored me when I needed it the most.

ACKNOWLEDGEMENTS

I owe my deepest gratitude to my advisor David Bindel, who allowed me great freedom to pursue my own research topics whilst being patient and understanding as I developed from a naive undergraduate to a somewhat-matured researcher. The immense breadth of knowledge I gained under his tutelage, from optimal control and model reduction to numerical linear algebra and numerical optimization, is a legacy I will never forget. I am also grateful to my committee members Ken Birman and Peter Frazier. To Ken for always providing advice, guidance, and humorous anecdotes, and to Peter for his frequent research feedback. My academic experience at Cornell was significantly shaped by my fellow advisees, Kun Dong and David Eriksson, in whom I found life-long friendships; by my collaborators Mike McCourt and Harvey Cheng at SigOpt, with whom I enjoyed discussing both optimization and basketball; and by my collaborators at Amazon Berlin, including Valerio Peronne, Huibin Shen, Matthias Seeger, and Cédric Archambeau, with whom I tackled many important research problems.

I want to thank my friends at Cornell outside my immediate academic family, including John Paul Ryan, Sebastian Ament, Aman Argawal, Edwin Peguero, Ted Yin, Michael Roberts, and Xinran Zhu. I want to especially thank my office-mates, including Weijia Song, with whom I shared memorable biking adventures; Kevin Sekniqi, with whom I tried —and failed— to start a company; and Danny Adams, who kept me grounded in reality whenever I was stressed. I also want to thank other faulty members at Cornell, including Mike Stillman, with whom I enjoyed many morning chats next to Gimme coffee, Emin Gün Sirer, who treated me like a true friend, and John Hopcroft, whose class I TAed for three years and learned much from in the process. Of course, no Cornell CS dissertation would be complete without acknowledging Becky Stewart, who supported me with all

sorts of logistics, and without whom I suspect the entire CS graduate student population would be rendered helpless.

My close college friends helped me stay sane throughout this entire process, and I am especially grateful to Robbie Zheng, Michael Lee, Jon San Miguel, Ayman Bin Kamruddin, and Charlie Woo Young Choi —whom I am especially grateful to, as he hosted me in Seoul and lent me a measure of peace to finish my thesis in during the Covid-19 outbreak. I could never forget about my high school friends Kang Fu, Michael Liu, Arvid Ali, Sean Min, Derek Hui, Eric Zhang, and Crystal Liu, with whom I have maintained a constant connection with despite many years and many differences.

No dissertation is completed without the constant support of loved ones. In particular, I am forever grateful to have had the opportunity to study at Cornell simultaneously with my younger sister, Jessica Alice Lee (Class of 2019). I was able to watch her grow from an impressionable teenager, to a stressed-out college student, to an accomplished and wonderful young woman. Though I sometimes wish she listened more to my —often unsolicited but dutifully given— advice, such is always the way with younger siblings! Finally, I thank Michele Lesmeister, who was my mentor throughout high school. She taught a confused and lost Chinese-American adolescent to love literature, to think critically, and to embrace his sense of wit and humor. The debt I owe her is immeasurable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

Bayesian optimization acquisition functions

EIpu($\mathbf{x}$) Expected improvement per unit cost acquisition function, page 62

$\Lambda(\mathbf{x})$ A generic Bayesian optimization acquisition function, page 13

EI($\mathbf{x}$) Expected improvement acquisition function, page 15

KG($\mathbf{x}$) Knowledge gradient acquisition function, page 16

PI($\mathbf{x}$) Probability of improvement acquisition function, page 14

UCB($\mathbf{x}$) Lower/upper confidence bound acquisition function, page 16

EI-cool($\mathbf{x}$) The cost-cooled expected improvement acquisition function, page 84

Constants

ℓ Kernel lengthscale, page 8

σ^2 Noise variance, page 9

τ_{init} Cost-effective initial design budget, page 83

b Bayesian optimization batch size or budget, page 16

h Markov decision process horizon, page 25

s Kernel scaling factor, page 8

General functions and Distributions

$\mathbb{E}$ Expectation, page 14

$\mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ A Gaussian Process, page 8

$\mathcal{L}_1$ One-norm loss function, page 96

$\mathcal{L}_2$ Two-norm loss function, page 96

$\mathcal{L}_{huber}$ Huber loss function, page 96

$\mathcal{L}_{tukey}$ Tukey loss function, page 96

$\mathcal{N}(\mu_X, K_{XX})$ A normal distribution with mean μ_X, covariance K_{XX}, page 9

$\mu(\mathbf{x})$ A Gaussian Process mean function, page 8

$\mu^{\nabla}(\mathbf{x})$ A Gaussian Process mean function with derivatives, page 12

$\Phi(z)$ Normal distribution cumulative distribution function, page 13

$\phi(z)$ Normal distribution probability density function, page 13

π Markov decision process policy, page 25

π^* The optimal Markov decision process policy, page 27

π_t Markov decision process decision rule, page 25

π_{ps} Best policy found in Π_{ps} during policy search, page 46

τ The cost constraint in a constrained MDP, and the cost budget in cost-constrained BO, page 63

$c(\mathbf{x})$ Objective cost model, page 62

$C(s, a, s')$ Cost function in a constrained MDP, page 63

$C_h^{\pi}(s_0)$ The expected cost of a policy starting from state s_0, page 64

f Continuous function defined on Ω, page 13

$k(\mathbf{x}, \mathbf{x}')$ A Gaussian Process kernel function, page 8

$k^{\nabla}(\mathbf{x}, \mathbf{x}')$ A Gaussian Process kernel function with derivatives, page 12

$p(\theta)$ Prior over hyperparameters, page 11

$V_h^{\pi}(s_0)$ Markov decision process value function, page 25

fill($\mathbf{X}$) The fill distance of a set $\mathbf{X}$, page 83

Other

$\mathbb{A}$ Markov decision process action space, page 25

$\mathbb{S}$ Markov decision process state space, page 25

Ω Bayesian optimization domain, assumed to be simple, page 13

Π_{ps} Paramaterized policy set used in MDP policy search, page 46

F The set of all possible trajectories in an MDP, page 65

G The set of feasible trajectories in a constrained MDP implicitly defined by its cost function, page 65

T Markov decision process epoch set, page 25

Matrices and Vectors

$\mathbf{x}$ A vector in $\mathbb{R}^d$, page 8

$\hat{K}_{XX}$ Kernel matrix for regression i.e., $\hat{K}_{XX} = K_{XX} + \sigma^2 \mathcal{I}$, page 9

θ Vector of all kernel hyperparamters, page 8

y_X A vector of function observations in GP regression, page 9

CHAPTER 1

INTRODUCTION

Global optimization seeks to identify an maximal or minimal point over a domain Ω. In the derivative-free setting, global optimization is a problem with many competing classes of methods, including, but certainly not limited to Nelder-Mead [Singer and Nelder, 2009], heuristic and metaheuristic-driven evolutionary search [Back, 1996, Deb, 2001], model-based methods [Rios and Sahinidis, 2013], and the classic branch-and-bound algorithm [Zinzen et al., 2009]. This dissertation concerns itself with the development of novel global optimization algorithms. We shall assume without loss of generality that the global optimization problem we are trying to solve is the minimization of a continuous, *black-box* objective function $f(\mathbf{x})$ over a closed domain $\Omega \subset \mathbb{R}^d$ that is relatively simple, such as the unit hypercube:

$$\min_{\mathbf{x}\in\Omega} f(\mathbf{x}).$$

A *black-box* function is one in which we can only query or evaluate, and for which we possess neither an analytical form nor derivative information. We also assume no additional structural information that might be taken advantage of to develop high-quality optimization algorithms, such as convexity or upper and lower bounds. By it's very nature, a black-box function is challenging to minimize; due to the limited information available, there exists no certificate of even local optimality, let alone global optimality. Absent any additional assumptions, a necessary and sufficient condition for any optimizer is that it search exhaustively —that is, that it searches every point in the optimization domain. This is clearly infeasible for continuous domains, and the seminal work of [Solis and Wets, 1981] proves that any optimizer which densely samples will eventually converge to within ϵ of the global minimum of $f(\mathbf{x})$ under minimal regularity conditions (e.g., continuity). Thus, even random selection of points in Ω will eventually locate the global minimum. However, convergence rates of different global optimization

methods remain elusive without further assumptions on the problem e.g., a known Lipshitz constant [Bull, 2011, Kawaguchi et al., 2015, Rudolph, 1996].

Bayesian optimization (BO) is a class of stochastic, model-based, global optimizers that empirically converge faster than many competing methods such as evolutionary search [Jones et al., 1998a, Jones et al., 1998b]. This key strength —the *sample efficiency* of BO— has led to its widespread adoption in certain fields such as machine learning. Achieving sample-efficiency demands that BO balance exploration and exploitation, a classic trade-off in global optimization. Too much exploration slows convergence, because the optimizer might over-evaluate regions of the domain with a low probability of containing the global minimum. Too much exploitation slows convergence, because the optimizer might repeatedly evaluate near a local minimum. BO expends significant computational effort to maintain this balance, resulting in high optimization overhead. In its basic form, BO has an $O(n^4)$ complexity, where n is the number of optimization iterations. This makes it is suitable when its overhead is minimal compared to that of $f(\mathbf{x})$, and when n is relatively small (≤ 1000). Consequently, BO is frequently used for optimization problems in which $f(\mathbf{x})$ is very costly to evaluate in terms of wall clock time, money, or human effort. Key applications of BO include robotic gait control, neural network hyperparameter tuning, and sensor set selection [Shahriari et al., 2016, Snoek et al., 2012, Frazier, 2018a].

While we provide a more formal description of BO in Chapter 2, we find it helpful to provide a rough outline below. Each iteration of BO executes the following two steps.

1. Build a probabilistic surrogate model of the objective.
2. Determine the next evaluation $\mathbf{x}$ via an acquisition function and evaluate it.

The acquisition function balances exploration and exploitation by scoring each point in Ω based on its potential to decrease the objective function. Different scoring criteria lead to different acquisition functions, and a well-chosen acquisition function enables BO to converge much faster. Unforunately, standard acquisition functions such as expected improvement (EI) are too greedy and perform little exploration. As a result, they perform poorly on multimodal problems [Hernández-Lobato et al., 2014] and have provably sub-optimal performance in certain settings, e.g., bandit problems [Srinivas et al., 2010]. A key research goal in BO is developing less greedy acquisition functions [Shahriari et al., 2016]. Examples include predictive entropy search (PES) [Hernández-Lobato et al., 2014] or knowledge gradient (KG) [Frazier et al., 2008].

In this dissertation, we introduce novel methodologies to further improve the performance of BO. Many of our methodologies are concerned with BO under a finite budget. We hope that by taking into account a finite budget, we can better decide where to evaluate $f(\mathbf{x})$, which will then improve upon the performance of classic BO. The research in this thesis primarily builds upon the following prior work in non-myopic Bayesian optimization: [Osborne et al., 2009, Lam et al., 2016, Lam and Willcox, 2017, Abdolshah et al., 2019], and has been published in the following first-author papers: [Lee et al., 2020a, Lee et al., 2020b], for which the author performed the large majority of the research. Additional research in this thesis concerning efficient Gaussian process regression with derivative information has been published in [Eriksson et al., 2018], for which the author was a second author. We structure this dissertation as follows:

- In Chapter 2, we cover background material, including Gaussian process

regression, BO acquisition functions, and different variants of BO including batch BO and cost-constrained BO.

- In Chapter 3, we frame the exploration-exploitation trade-off as a principled balance between immediate and future rewards in a continuous state and action space Markov decision process (MDP), in which one tries to maximize a cumulative reward over a fixed number of iterations.
- In Chapter 4, we devise efficient strategies to solve Bayesian optimization MDPs through a popular technique known as rollout, and discuss variance reduction techniques to speed up rollout.
- In Chapter 5, we extend rollout to the *cost-constrained* BO setting in which evaluating the objective has variable cost.
- In Chapter 6, we discuss simple heuristics that perform very well in the cost-constrained BO setting. We combine these heuristics and extend them to the batch setting, and show that they yield significant improvements across the board.
- In Chapter 7, we extend Gaussian process regression to the setting with derivatives, and use this as a building block to perform BO with derivative information. We also discuss dimensionality reduction techniques.

CHAPTER 2

BACKGROUND

In this chapter, we include the necessary background to understand BO and the research in this dissertation. While we hope this chapter is largely self-contained, the reader will need some prior knowledge of probability theory, linear algebra, and calculus.

2.1 Multivariate Gaussian

The multivariate Gaussian, also known as the multivariate normal, generalizes the univariate Gaussian to $\mathbb{R}^n$. The vector-valued random variable $y = [y_1, y_2, \ldots, y_n]^T \in \mathbb{R}^n$ is said to follow a multivariate normal distribution with mean vector $\mu \in \mathbb{R}^n$ and positive definite covariance matrix $K \in \mathbb{R}^{n \times n}$ if its probability density function (pdf) is given by:

$$p(y \mid \mu, K) = \frac{1}{(2\pi)^{d/2}|K|^{1/2}} \exp\Big(-\frac{1}{2}(y-\mu)^T K (y-\mu)\Big).$$

We can write this more succinctly as $y \sim N(\mu, K)$. Note that $|K|$ is the determinant of matrix K i.e., the product of its eigenvalues. $p(y \mid \mu, K)$ is a proper distribution in that it is always positive and integrates to 1. Among the multivariate normal's fundamental properties is *self-conjugation*; it's conditional distribution is also a multivariate normal. Assume we have sub-divided y into two random variables y_1 and y_2 of dimension n_1 and n_2, respectively, and done the same with μ and K:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \sim \mathcal{N}\Bigg(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} \\ K_{12}^T & K_{22} \end{bmatrix} \Bigg),$$

where $y_1, \mu_1 \in \mathbb{R}^{n_1}$, $y_2, \mu_2 \in \mathbb{R}^{n_2}$, $K_{11} \in \mathbb{R}^{n_1 \times n_1}$, $K_{12} \in \mathbb{R}^{n_1 \times n_2}$ and $K_{22} \in \mathbb{R}^{n_2 \times n_2}$. The distribution of y_1 conditioned on y_2 is given by:

$$
\begin{aligned}
y_1 \mid y_2 &\sim \mathcal{N}\left(\hat{\mu}, \hat{K}\right), \\
\hat{\mu} &= \mu_2 + K_{12}^T [K_{11}]^{-1} (y_1 - \mu_1), \\
\hat{K} &= K_{22} - K_{12}^T [K_{11}]^{-1} K_{12}.
\end{aligned}
$$

In Bayesian statistics, $\hat{\mu}$ is known as posterior mean and $\hat{K}$ is known as the posterior covariance[1], given that $\mathcal{N}(\mu, K)$ is the prior. This terminology will become important when we generalize the multivariate normal to their infinite-dimensional counterparts, Gaussian processes, in the next section.

2.2 Gaussian processes and Gaussian process regression

One can think of a Gaussian process (GP) as an infinite-dimensional generalization of a multivariate Gaussian. To be more precise, a GP is a collection of random variables, any finite number of which are jointly Gaussian [Rasmussen and Williams, 2006]. It defines a distribution over functions on $\mathbb{R}^d$, which we write as:

$$
f \sim \mathcal{GP}(\mu(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')),
$$

where $\mu(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}$ is a mean field and $k(\mathbf{x}, \mathbf{x}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a symmetric and positive (semi)-definite covariance kernel. A kernel $k(\mathbf{x}, \mathbf{x}')$ is a bivariate function that correlates its two arguments. A kernel dependent on $r = \| \mathbf{x} - \mathbf{x}' \|_2$ i.e., $k(\mathbf{x}, \mathbf{x}') = \phi(r)$ is a radial basis function (RBF) kernel, and typically contains length-scale and scale factor hyperparameters ℓ and s, respectively. We list a few popular

[1] $\hat{K}$ is also the Schur complement of K in numerical linear algebra.

Kernel	Formula
Squared Exponential (SE)	$s^2\exp(-\frac{r^2}{2\ell^2})$
Matérn 3/2	$s^2(1+\frac{\sqrt{3}r}{\ell})\exp(-\frac{\sqrt{3}r}{\ell})$
Matérn 5/2	$s^2(1+\frac{\sqrt{5}r}{\ell}+\frac{\sqrt{5}r}{3\ell^3})\exp(-\frac{\sqrt{5}r}{\ell})$

Figure 2.1: A few popular RBF kernels.

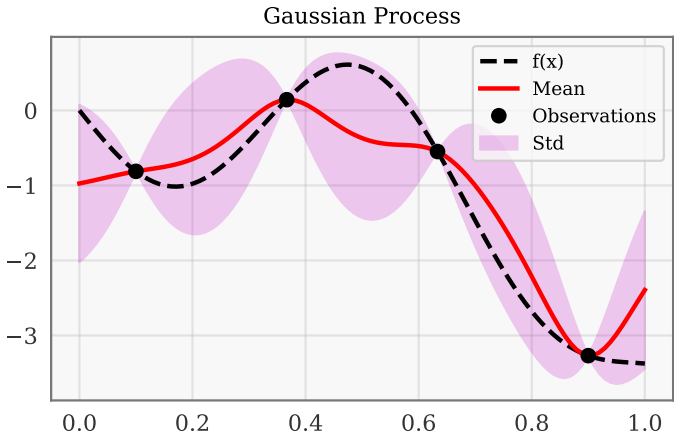


Figure 2.2: A Gaussian process (GP) is a distribution over functions. Here, we show the posterior mean (red) and 95 percent confidence interval (magenta) of a GP given four observations of a ground-truth function $f(\mathbf{x})$ (black). The GP uses the Matérn 5/2 kernel, and its hyperparameters are calculated through maximum likelihood estimation.

radial kernels in Table 2.1. A Gaussian process prior means the following. At any set of points $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n\times d}$, the values $f_X = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]^T$ are jointly distributed with respect to the following multivariate normal:

$$f_X \sim \mathcal{N}(\mu_X, K_{XX}),$$

where $\mu_X \in \mathbb{R}^n$ such that $(\mu_X)_i = \mu(\mathbf{x}_i)$ and $K_{XX} \in \mathbb{R}^{n\times n}$ such that $(K_{XX})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The mean and kernel functions are often chosen to reflect prior knowledge in the data e.g., a periodic kernel for periodic data.

GP regression is equivalent to computing a posterior mean at a desired point, and works as follows. Let us assume that we have a Gaussian process prior over a set of locations $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]^T \in \mathbb{R}^{n\times d}$ with a mean function μ and a kernel

$k(\mathbf{x}, \mathbf{x}')$. Let us assume that we've also observed values $y_X = [y_1, \ldots, y_n]^T$ at $\mathbf{X}$, where $y_i = f(\mathbf{x}_i)$. Because the multivariate Gaussian is a conjugate prior, the posterior distribution at $\mathbf{x}$ is given by the standard conditional distribution in the prior section:

$$\mathcal{N}(\hat{\mu}, \hat{K}_{xx}),$$

where $\hat{\mu}$ and $\hat{K}_{xx}$ are the posterior mean and covariance:

$$\hat{\mu}_x = \mu_x + K_{Xx}^T [K_{XX}]^{-1} (y_X - \mu_X).$$
$$\hat{K}_{xx} = K_{xx} - K_{Xx}^T [K_{XX}]^{-1} K_{Xx}.$$

It is usually standard in GP regression to assume y_X is contaminated by independent Gaussian noise with variance σ^2. This is equivalent to using the regularized kernel matrix $\widetilde{K}_{XX}$ instead of K_{XX} during regression:

$$\widetilde{K}_{XX} = K_{XX} + \sigma^2 \mathcal{I}.$$

Kernel matrices are known to be often poorly conditioned [Wendland, 2004], and adding a relatively small diagonal factor σ alleviates poor conditioning to a certain extent. It can also be learned jointly with the kernel hyperparameters. An simple example of GP regression can be seen in Figure 2.2, in which we've plotted the posterior mean and square root of the variance (the standard deviation) of a GP with four observations in 1D.

2.2.1 Kernel hyperparameters

The quality of a GP model depends not only on the choice of kernel function, but also on the kernel function's hyperparameters. The choice of kernel is

usually not automated[2], and often requires external knowledge e.g., periodicity of the data. However, once a kernel has been chosen, there are a few ways of automatically selecting its hyperparameters, which we denote by the vector θ. The first is maximum likelihood estimation (MLE). To be concise, we suppress the dependence of K_{XX} matrices on θ in our notation. Under a Gaussian process prior depending on the covariance hyperparameters θ, the log marginal likelihood of the observations y_X given $\mathbf{X}$ and θ is:

$$\mathcal{LML}(y_X \mid \mathbf{X}, \theta) = -\frac{1}{2}\left[(y_X - \mu_X)^T \alpha + \log|\tilde{K}_{XX}| + n\log 2\pi\right].$$
$$\frac{\partial}{\partial \theta_j}\mathcal{LML}(y_X \mid \mathbf{X}, \theta) = -\frac{1}{2}\mathrm{tr}\left((\alpha\alpha^T - \tilde{K}_{XX}^{-1})\frac{\partial \tilde{K}_{XX}}{\partial \theta_j}\right).$$

where $\alpha = \tilde{K}_{XX}^{-1}(y_X - \mu_X)$ and $\tilde{K}_{XX} = K_{XX} + \sigma^2 I$. The standard direct method to pre-compute the Cholesky factorization of $\tilde{K}_{XX}$, and use it to compute α and its log-determinant. A Cholesky factorization has complexity $O(n^3)$, which is too expensive for inference and learning beyond even just a few thousand points. Recent work exploits the *data-sparsity* of certain kernel matrices to solve with $\tilde{K}_{XX}$ using conjugate gradient [Saad, 2003] and to calculate the log-determinant $\log|\tilde{K}_{XX}|$ with stochastic trace estimation [Dong et al., 2017]. These scale better than the direct approach because $\tilde{K}_{XX}$ is structured, and can be applied to a vector in under $O(n^2)$ time.

A Bayesian treatment of kernel hyperparameters requires a prior on the hyperparameters $p(\theta)$. Given this prior, one may opt to either perform a maximum-a-posteriori (MAP) estimate or infer a full posterior distribution $p(\theta|\mathbf{X}, Y)$, which is then integrated out. The MAP will maximize the product of the likelihood and the prior, or equivalently, the sum of the log-likelihood and the log-prior.

[2]It is worth noting that there is significant research devoted to kernel selection, which we will not discuss in this dissertation.

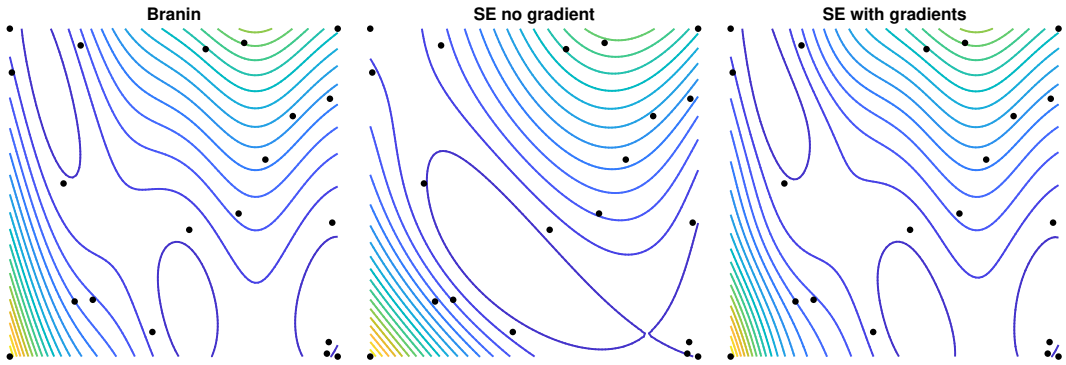


Figure 2.3: An example where gradient information pays off; the true function is on the left. Compare the regular GP without derivatives (middle) to the GP with derivatives (right). Unlike the former, the latter is able to accurately capture critical points of the function.

Inferring and integrating out a full posterior distribution $p(\theta|\,\mathbf{X}, Y)$ is typically done via Markov chain Monte Carlo (MCMC) [Rasmussen and Williams, 2006]. A full Bayesian treatment of hyperparameters is the most expensive, but has been shown to exhibit greater robustness and prediction accuracy in the low-data regine [Snoek et al., 2012], who recommends a uniform prior for the kernel lengthscales and a log-normal prior for the kernel scaling parameter.

2.2.2 Derivative information

The GP model can easily be extended to incorporate derivative information, which is especially valuable in higher dimensions, but comes at a cost: the kernel matrix augmented with derivatives, K_{XX}^{∇}, is of size $n(d+1)$-by-$n(d+1)$. This makes scalability an even larger issue as training and prediction become $O(n^3d^3)$ and $O(nd)$ respectively [Rasmussen and Williams, 2006, Eriksson et al., 2018]. Figure 2.3 illustrates the value of derivative information; fitting with derivatives is evidently much more accurate than fitting function values alone.

We define a multi-output GP that allows us to both predict derivatives and make inference based on derivative information. The multi-output GP model takes the form

$$\mu^{\nabla}(\mathbf{x}) = \begin{bmatrix} \mu(\mathbf{x}) \\ \partial_{\mathbf{x}}\mu(\mathbf{x}) \end{bmatrix}, \qquad k^{\nabla}(\mathbf{x}, \mathbf{x}') = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}') & (\partial_{\mathbf{x}'}k(\mathbf{x}, \mathbf{x}'))^T \\ \partial_{\mathbf{x}}k(\mathbf{x}, \mathbf{x}') & \partial^2 k(\mathbf{x}, \mathbf{x}') \end{bmatrix},$$

where $\partial_{\mathbf{x}}k(\mathbf{x}, \mathbf{x}')$ and $\partial^2 k(\mathbf{x}, \mathbf{x}')$ represent the column vector of (scaled) partial derivatives in $\mathbf{x}$ and the matrix of (scaled) second partials in $\mathbf{x}$ and $\mathbf{x}'$, respectively. As in the scalar GP case, we model measurements of the function as contaminated by independent Gaussian noise, but we can use different noise variances for the function values and each partial derivative.

2.3 Bayesian optimization

Algorithm 1 Bayesian Optimization

0: **input:** budget b, acquisition function $\Lambda(\mathbf{x})$
0: Place a GP prior on f
1: **for** $k = 0, \ldots, b-1$ **do**
2: Learn GP hyperparameters, update posterior
3: Determine next evaluation point $\mathbf{x}^* = \arg\max_{\Omega} \Lambda(\mathbf{x})$
4: Sample value y_{k+1} at point $\mathbf{x}_{k+1}$ according to π^*
5: Update observation set with $\mathbf{x}_{k+1}$ and y_{k+1}
6: **end for**
7: **Return:** $\mathbf{x}_i$ for which the minimum y_i value was observed

Consider minimizing a continuous, black-box function $f(\mathbf{x}) : \Omega \to \mathbb{R}$ over a simple domain $\Omega \subset \mathbb{R}^d$. By black-box, we mean that the analytic form and gradients are unavailable, and the function can only be queried through potentially noisy evaluations. Bayesian optimization (BO) is a well-established class of methods to address this problem. Using its past observations, BO builds an probabilistic model of the objective function to guide optimization, and selects

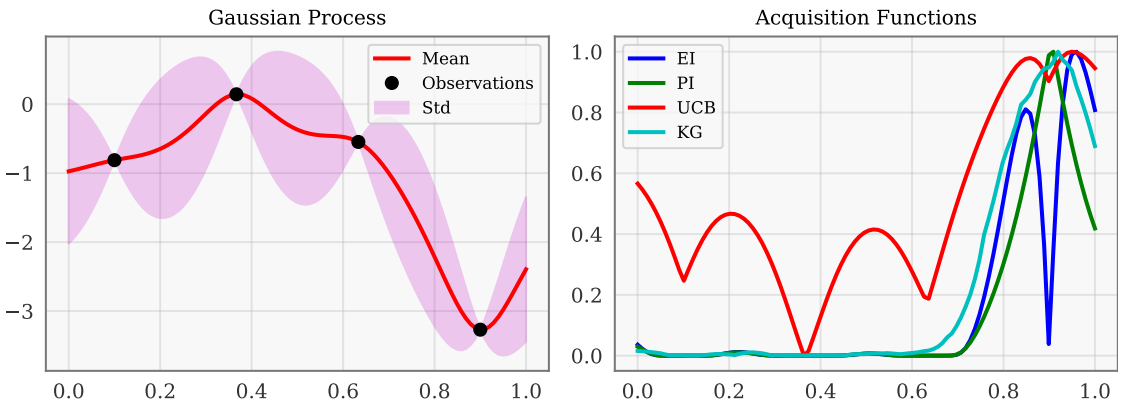


Figure 2.4: In Bayesian optimization iteration proceeds by first building a GP (left plot) from observations. Optimization of the acquisition function (right plot) gives a new sample point. Left: the GP model. Right: the expected improvement, probability of improvement, upper confidence bound ($\kappa = 1$), and knowledge gradient acquisition functions are plotted.

the next iterate through an acquisition function, which scores each point in the optimization domain based on its potential to decrease the objective function.

We outline BO in Algorithm 1, where $\Lambda(\mathbf{x})$ represents an arbitrary acquisition function. Below, we list some popular acquisition functions. Many are expressed as the expectation of a random variable, and only a subset of these have closed forms, which we provide when available. We take $\phi(z)$ to be the pdf of the Normal distribution and $\Phi(z)$ to be its cdf, where $z = \frac{y^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}$. We denote the current known minimum as $y^* := f(\mathbf{x}_{min})$.

2.3.1 Probability of improvement

Probability of improvement (PI) is one of the simplest acquisition functions, and is defined as the probability that the point $\mathbf{x}$ will yield an increase in the objective function. PI can be written as:

$$\text{PI}(\mathbf{x}) = \mathbb{E}[\gamma(y(\mathbf{x}))] = \Phi\left(\frac{y^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right).$$

$$\gamma(y(\mathbf{x})) = \begin{cases} 1\,, & y < y^* \\ 0\,, & y \geq y^*. \end{cases}$$

PI is not a popular acquisition function due to its empirical poor performance [Jones et al., 1998b], as it tends to be overly exploratory. However, we include it here because of its usefulness as a control variate when estimating other acquisition functions.

2.3.2 Expected improvement

Expected improvement (EI), introduced by [Mockus et al., 1978] and then reintroduced by [Jones et al., 1998a], is arguably the de-facto standard BO acquisition function. EI measures the expected reduction in the objective:

$$\mathrm{EI}(\mathbf{x}) = \mathbb{E}[\gamma(y(\mathbf{x}))] = (y^* - \mu(\mathbf{x}))\Phi\left(\frac{y^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x})\phi\left(\frac{y^* - \mu(\mathbf{x})}{\sigma(\mathbf{x})}\right).$$

$$\gamma(y(\mathbf{x})) = \begin{cases} y^* - y\,, & y < y^* \\ 0\,, & y \geq y^*. \end{cases}$$

EI is known to be too greedy and perform little exploration [Qin et al., 2017]. As a result, it performs poorly on multimodal problems [Hernández-Lobato et al., 2014] and has provably sub-optimal performance in certain settings, e.g., bandit problems [Srinivas et al., 2010].

2.3.3 Lower and upper confidence bound

In the BO setting, the lower confidence bound acquisition is used when minimizing and the upper confidence bound is used when maximizing

[Srinivas et al., 2010]. Without loss of generality, we will overload UCB as acronym for both. In terms of minimization, UCB is defined as:

$$\text{UCB}(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\sigma(\mathbf{x}),$$

where κ is an input parameter representing the tradeoff between exploration and exploitation. UCB provides regret bounds and thus gives certain convergence guarantees, but its performance is dependent upon κ.

2.3.4 Knowledge gradient

The knowledge gradient (KG) acquisition function [Frazier et al., 2008, Frazier, 2018b] reconsiders EI's need to only return a previously evaluated point as the final BO solution. KG instead is willing to return a final solution that has some uncertainty attached to it, and assumes this to be the expected minimum of the GP's posterior mean. Let $\mu^*_{n+1}(y(\mathbf{x}))$ be the minimum of the posterior GP mean assuming $y(\mathbf{x})$ has been evaluated. Then KG is defined as:

$$\text{KG}(\mathbf{x}) = \mathbb{E}[\gamma(y(\mathbf{x}))].$$

$$\gamma(f|x) = \mu^*_{n+1}(y(\mathbf{x})).$$

Note that KG does not have an analytic form, unlike the other acquisition functions, and therefore must be approximated numerically.

2.4 Batch Bayesian optimization

Standard BO is sequential but can be extended to the batch setting. In this setting, b candidates $\mathbf{x}^1, \ldots, \mathbf{x}^b$ are selected by a batch acquisition function and

then evaluated in parallel. This parallel evaluation is especially suited to settings in which multiple computers or resources are available to perform function evaluations, which can occur either synchronously or asynchronously. A large body of work has been devoted to batch BO, including (but not limited to) work by [González et al., 2016, Azimi et al., 2010, Wu and Frazier, 2016, Shah and Ghahramani, 2015, Wang et al., 2017, Snoek et al., 2012]. Batch BO is inherently less sample efficient than sequential BO, which uses information the most optimally. Consequently, the challenge in the batch setting is candidate *diversity*, in which batch candidates are sufficiently different from each other, but also sufficiently exploitative, so that sample efficiency is largely preserved. In the ideal world, a batch size of b yields a corresponding b-times linear speed-up in convergence rate.

2.4.1 Batch acquisition functions

A batch acquisition function is maximized over over Ω^b to return b different candidates $\mathbf{x}^1, \ldots, \mathbf{x}^b$. These are evaluated in parallel. Below are batch extensions to EI and KG, which are both expressed as an expectation over the GP's joint posterior distribution:

$$\Lambda_b(\mathbf{x}_1 \ldots \mathbf{x}_b) = \mathbb{E}[\gamma(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_b))].$$

For example, EI can be extended to the batch EI criterion by modifying the associated random variable to take the minimum value of the batch:

$$\gamma(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_b)) = \begin{cases} y^* - \min_{1 \le i \le b} y(\mathbf{x}_i) \ , & \min_{1 \le i \le b} y(\mathbf{x}_i) < y^* \\ 0 \ , & \min_{1 \le i \le b} y(\mathbf{x}_i) \ge y^*. \end{cases}$$

KG can also be extended to the batch KG criterion in the same way:

$$\gamma(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_b)) = \mu^*_{n+b}(y(\mathbf{x}_1), \ldots, y(\mathbf{x}_b)).$$

These expectations are taken with respect to the GP's joint posterior distribution at $\mathbf{x}^1, \ldots, \mathbf{x}^b$. Both batch EI and batch KG do not have analytic forms, and their values and gradients must be approximated. This makes them unwieldy to maximize, and this especially true with large batch sizes.

2.4.2 Batch fantasizing

An alternative to batch acquisition functions is batch fantasizing, also known as the Kriging believer [Snoek et al., 2012]. Instead of defining a batch criterion used to select b simultaneous points, batch fantasizing executes a sequential simulation using a sequential acquisition function, and determines a batch through this simulation. The sequential simulation predicts n_f different future evaluation trajectories, known as "fantasies"[Wilson et al., 2018]. These n_f fantasies are aggregated to identify a sequence of evaluation points, which is then evaluated in parallel. Batch fantasizing is described in Algorithm 2.

We make heavy use of batch fantasizing in Chapter 6, and describe Algorithm 2 assuming EI as the acquisition function without loss of generality. First, let $\mathbf{x}_1$ be the the argmax of EI, and sample n_f values from the posterior at $\mathbf{x}_1$. These n_f "fantasy" values represent different future evaluation trajectories. We maintain n_f different GPs, and update them each with a different fantasy value (GP hyperparameters are kept constant). Second, we maximize a new acquisition that is the average of EI acquisition functions over each different fantasy. The argmax of this averaged acquisition is set to be $\mathbf{x}_2$. We repeat these steps until a

Algorithm 2 Batch Fantasizing with EI

Input: batch b, fantasies n_f, data $\mathbf{X}, Y$
for $i = 1, \ldots, n_f$ **do**
  $Y^{(i)} \leftarrow \texttt{copy}(Y)$
end for
$\mathbf{x}_1 \leftarrow \texttt{argmax}_{\mathbf{x} \in \Omega}\ \text{EI}(\mathbf{x} \mid \mathbf{X}, Y)$
for $j = 2, \ldots, b$ **do**
  $\mathbf{x} \leftarrow \mathbf{x} \cup \{\mathbf{x}_j\}$
  for $i = 1, \ldots, n_f$ **do**
    $y_k^{(i)} \leftarrow \texttt{sample\_posterior}(\mathbf{x} \mid \mathbf{X}, Y^{(i)})$
    $Y^{(i)} \leftarrow Y^{(i)} \cup \{y^{(i)}\}$
  end for
  $\mathbf{x}_j \leftarrow \texttt{argmax}_{\mathbf{x} \in \Omega} \frac{1}{n_f} \sum_{i=1}^{n_f} \text{EI}(\mathbf{x} \mid \mathbf{X}, Y^{(i)})$
end for
return $\mathbf{x}_b = \{\mathbf{x}_1, \ldots, \mathbf{x}_b\}$

batch of b points is obtained.

2.5 Bayesian optimization with different cost metrics

Progress in BO is implicitly measured with iterations. This assumption is inherent in the EI, KG, and UCB acquisition functions we have covered, which only take into account the distribution of the objective function and not variability in the objective function's cost. In fact, many of the applications that BO is used in have variable cost, which can often be measured. In robotic gait control, different control actions take different times to accomplish. In hyperparameter optimization, different hyperparameters result in different training times. In materials design, constituent components have different monetary costs [Abdolshah et al., 2019]. The setting in which the cost of the objective varies is known as *cost-aware* BO. When we are cost-aware, and also have an a-priori known budget, then we say that we are *cost-constrained*.

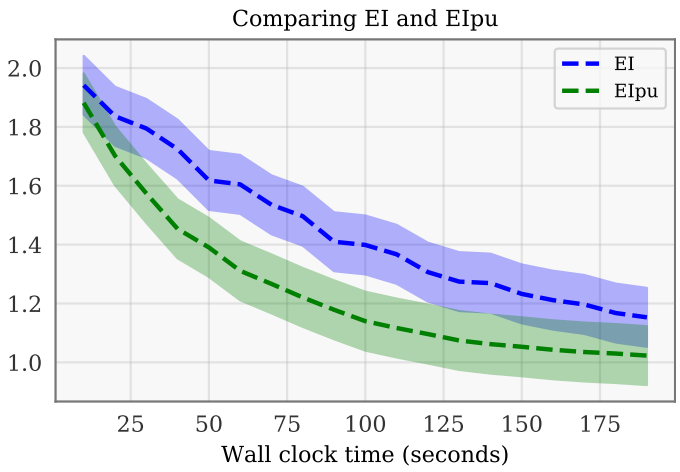


Figure 2.5: We compare EI (blue) and EIpu (green) on a synthetic function with synthetic wall clock time.

Cost-aware and cost-constrained BO are largely understudied at the time of this dissertation's writing, and indeed, we argue in Chapter 5 that the *large majority* of practical applications are cost-constrained; a large portion of this dissertation is dedicated to developing cost-constrained BO methods. Typical approaches in this setting include *multi-fidelity* BO, in which fidelity parameters $s \in [0, 1]^m$, such as iteration count or grid size, are assumed to be a noisy proxy for high-fidelity evaluations [Forrester et al., 2007, Kandasamy et al., 2017, Poloczek et al., 2017, Wu and Frazier, 2019]. Increasing s decreases noise at the expense of cost. Not all applications may be formulated as multi-fidelity, and we develop more general methods in Chapter 5 and Chapter 6.

The de-facto heuristic in cost-constrained BO is to divide an acquisition function by a cost model $c(\mathbf{x})$. The cost model $c(\mathbf{x})$ models the cost of evaluating $f(\mathbf{x})$. In particular, [Snoek et al., 2012] uses the expected improvement per unit acquisition function:

$$\text{EIpu}(\mathbf{x}) = \frac{\text{EI}(\mathbf{x})}{c(\mathbf{x})}$$

In Figure 2.5, we see that EIpu beats EI on a synthetic function, whose wall clock

time also follows a synthetic function. However, as we will show in Chapter 5, dividing the acquisition function by the cost is a rather weak heuristic that only works well if the global optimum is relatively cheap to evaluate. We improve upon EIpu in Chapters 5 and 6.

CHAPTER 3

MARKOV DECISION PROCESSES AND BAYESIAN OPTIMIZATION

3.1 Introduction

BO acquisition functions such as EI are greedy in the sense that they ignore how the current iteration will affect future ones. Typically, the BO exploration-exploitation trade-off is expressed in the context of a one-step optimal process: for the next iteration, choose the point that balances information quantity and quality. Unfortunately, many acquisition criteria are reliant on additional parameters, which affect BO performance in rather poorly understood ways. A classic example is the κ parameter in UCB-κ, which represents the amount of exploration to be done, and largely dictates performance of the underlying BO routine.

A *look ahead* approach, also known as *non-myopia*, is aware of the remaining iterations and can balance the exploration-exploitation trade-off correspondingly by accounting for the impact of future evaluations. The characterization of the exploration-exploitation trade-off in BO as balance between immediate and future rewards is an important one. A key research goal in BO is developing less greedy acquisition functions [Shahriari et al., 2016, Frazier, 2018a], and accounting for future rewards is a more principled way of doing so than existing state-of-the-art acquisition functions. Examples of these include predictive entropy search (PES) [Hernández-Lobato et al., 2014] or knowledge gradient (KG) [Frazier et al., 2008], and generally rely on acquisition criteria that have been empirically observed to work well.

To be non-myopic, we must reason about Bayesian optimization as a sequence of decisions under known epistemic uncertainty. In Figure 3.1, we depict such a sequence. At iteration k, we decide on a new evaluation $\mathbf{x}_{k+1}$, which in turn yields a reward, or observation y_{k+1}. We continue this process for h steps, where

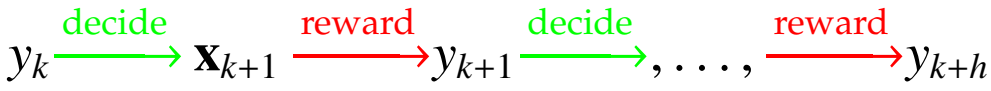


Figure 3.1: Modeling h steps of BO, where y_k is known. At each step, we make a decision (an evaluation $\mathbf{x}_k$) and receive a reward (an observation y_k). We might assume for all $k+1 \leq t \leq k+h$ that y_t is drawn from a posterior distribution conditioned on all prior decisions and rewards.

h is the total budget. We assume that for all $k+1 \leq t \leq k+h$ that y_t is drawn from a posterior distribution conditioned on all prior decisions and rewards; because y_t is modeled as a GP, we can do this in a reasonably straightforward manner. Having established distributions for $y_{k+1}, \ldots, y_{k+h}$, we might reasonably ask for an algorithm to compute the optimal sequence of decisions, which in this case are the evaluation points $\mathbf{x}_{k+1}, \ldots, \mathbf{x}_{k+1}$, so that the smallest of the observations is minimized in expectation. As it turns out, this is a difficult problem, and one that this chapter expands upon at length!

There is a rich literature on sequential optimization and control of discrete-time systems, starting from the seminal work of [Bellman, 1952, Bellman, 1961] and continuing to this day, including recent treatments by [Bertsekas, 1995], [Powell, 2007], and [Puterman, 2014]. The classic framework in this space is the Markov decision process (MDP), which models a stochastic control process in which the next state depends only on the current one. MDPs are widely studied and are ubiquitous in areas such as optimal control of dynamical systems, econometrics, or robotics. A quintessential example of an MDP is the grid-world, in which an agent living in a finite grid can either walk up, down, left, or right, and whose goal is to reach a terminal grid point in the minimal amount of time. More generally, an MDP formalizes the states an agent can reach, the actions an agent can take, and the goal an agent is trying to achieve. As it turns out, stochastic global optimization can be formalized in the same way.

As we have already stated earlier, we frame the exploration-exploitation trade-off as a balance between immediate and future rewards in a continuous state and action space Markov decision process (MDP). In this framework, *non-myopic* acquisition functions are optimal MDP policies, and promise better performance by considering the impact of future evaluations up to a given BO budget (also referred to as the *horizon*).

3.2 The Markov decision process

In this section, we briefly introduce discrete-time MDPs. There is a wealth of research and accompanying tutorials on MDPs, and we opt to only briefly review the basics for the sake of parsimony. Our presentation concerns *finite-horizon* MDPs, and we ignore algorithms concerning their infinite-horizon counterparts.

We follow the generally-accepted notation from [Puterman, 2014]; we express an MDP as the collection $< T, \mathbb{S}, \mathbb{A}, P, R >$. T is the set of decision epochs and for our problem, we only consider the finite horizon setting, where $T = \{0, 1, \ldots, h-1\}$, for some integer $h < \infty$. The state space, $\mathbb{S}$, encapsulates all the information needed to model the system from time $t \in T$. $\mathbb{A}$ is the action space. Given a state $s \in \mathbb{S}$ and an action $a \in \mathbb{A}$, $P(s'|s, a) : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0, 1]$ is the transition probability of the next state being s'. $R(s, a, s') : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}^+$ is the reward received for choosing action a from state s, and ending in state s'.

A decision rule, $\pi_t : \mathbb{S} \to \mathbb{A}$, maps states to actions at time t. A policy π is a series of decision rules $\pi = (\pi_0, \pi_1, \ldots, \pi_{h-1})$, one at each decision epoch. Given a policy π, a starting state s_0, and horizon h, we can define the expected total

reward of π as $V_h^\pi(s_0)$:

$$V_h^\pi(s_0) = \mathop{\mathbb{E}}_{s_0, s_2, \ldots, s_{h-1}} \Big[\sum_{t=0}^{h-1} R(s_t, \pi_t(s_t), s_{t+1}) \Big].$$

The above expectation is taken with respect to the transition probabilities of states at each epoch, and we will omit the subscript below the expectation for the sake of neat notation for the rest of this dissertation. $V_h^\pi(s_0)$ is also known as the *value function*, and should be thought of as the fitness measure of any particular policy. Only in very specific MDPs, such as linear control systems using a linear quadratic regulator, is the analytic form of $V_h^\pi(s_0)$ available [Bemporad et al., 2002]. Absent a known analytic form, the value function must computed through numerical integration. Assuming we can draw from the transition probability $P(s, a, s')$, we can estimate the expected total reward for a policy π with Monte Carlo (MC) integration [Sutton and Barto, 1998]:

$$V_h^\pi(s_0) \approx \frac{1}{N} \sum_{i=1}^{N} \Big[\sum_{t=0}^{h-1} R(s_t^i, \pi_t(s_t^i), s_{t+1}^i) \Big].$$

More intuitively, we approximate $V_h^\pi(s_0)$ by averaging rewards accrued over N independent sample paths from the MDP following policy π starting with information s_0:

$$(\pi(s_0^i), s_1^i, \pi(s_1^i), s_2^i, \pi(s_2^i), \ldots, s_h^i), \ 1 \le i \le N \ ,$$

where superscript i indexes the sample number and subscript h indexes the horizon. Unsurprisingly, computing the value function is also known as *policy evaluation* in the MDP literature.

Note that we have presented a simplified version of MDPs. Typical formulations also include a discount factor γ and a terminal reward $R(s)$. We have omitted these for notational brevity, as they are not used in this dissertation.

3.3 Solving MDPs

In phrasing a sequence of decisions as an MDP, our goal is to find the optimal policy π^* that maximizes the expected total reward:

$$\pi^* = \arg\sup_{\pi \in \Pi} V_h^{\pi}(s_0),$$

where Π is the space of all admissible policies. Note that in the case of a finite state and action space, the number of admissible policies is $d_a^h d_s^h$, where d_a is the size of the action space and d_h is the size of the state space. The exponential increase of both the action and state spaces in h means that most real world MDPs are quite difficult to solve exactly. However, we find it instructive to review exact solutions so that our approximate solutions may be better motivated.

3.3.1 Optimality conditions

An optimal MDP policy always exists, though it may not be unique. Let the optimal policy be denoted as:

$$\pi^* = (\pi_0^*, \pi_1^*, \ldots, \pi_{h-1}^*).$$

Let us consider the following MDP subproblem, in which we are attempting to minimize the following value function starting at time k:

$$\mathbb{E}\Big[\sum_{t=k}^{h-1} R(s_t, \pi_t(s_t), s_{t+1})\Big],$$

for any intermediate state s_k. This MDP subproblem also known as the k-tail subproblem. *Bellman's principle of optimality* [Bellman, 1952, Powell, 2007] states that the optimal policy for the k-tail subproblem is in fact:

$$\pi^* = (\pi_k^*, \pi_{k+1}^*, \ldots, \pi_{h-1}^*).$$

In other words, the optimal policy for the $k-$tail subproblem is completely independent of the first k optimal decision rules $(\pi_0^*, \pi_1^*, \ldots, \pi_{k-1}^*)$! Because the optimal policy of the future does not depend on the past, a straightforward algorithm exists to simultaneously calculate both the optimal policy and its value function. We discuss it in the following subsection.

3.3.2 Dynamic programming

Bellman's principle of optimality allows us to construct the following high level algorithm to recover the optimal policy. First, for every final state s_{h-1}, we compute the optimal policy to maximize the tail problem for $k = h - 1$. This gives us (π_{h-1}^*). We then compute the optimal policy to maximize the tail problem for $k = h - 2$. This gives us $(\pi_{h-2}^*, \pi_{h-1}^*)$. We continue backwards in time until we reach $k = 0$, which then gives us all optimal decision rules and thus the optimal policy:

$$\pi^* = (\pi_0^*, \pi_1^*, \ldots, \pi_{h-1}^*).$$

This dynamic programming algorithm can be recursively expressed as a sequence of nested optimization problems starting from $k = h - 1$ and working in reverse to $k = 0$:

$$\begin{aligned} V_h(s^*) &= 0, \\ V_k(s) &= \max_{a \in \mathbb{A}} \mathbb{E}_{s'}[R(s, a, s') + V_{k+1}(s')] \quad (k = h-1, h-2, \ldots, 0). \end{aligned}$$

We start with zero reward at a terminal state s^*, and iterate backwards in time. At each step, we perform a maximization problem to determine the optimal decision rule that maximizes the k-tail subproblem. At the end of iteration, $V_0(s_0)$ is precisely the value function of the optimal policy i.e., $V_0(s_0) = V_h^{\pi^*}(s_0)$. Note

that in many typical MDP formulations, the reward at a terminal zero state may be determined by a nonzero terminal reward function, which we have omitted above.

3.4 BO as an MDP

As we mentioned in the beginning of this chapter, we might think of BO as a sequential process in which one determines the next iteration $\mathbf{x}_{next}$, evaluates $f(\mathbf{x}_{next})$, observes some value, and collects some reward associated with this value. If these values are drawn from a distribution conditioned on past data, then BO —and indeed, many related, model-driven sequential optimization problems under uncertainty— can be formulated as an MDP. The dynamics of this MDP are driven by the inherent stochasticity in the GP, whose posterior distributions provide a framework for expressing how optimization might proceed in the future. Having covered the MDP fundamentals, we now map them to their analogous counterparts in BO.

Given a GP prior over an observation set $\mathcal{D}_k$ with mean $\mu^{(k)}$ and kernel $K^{(k)}$, we model h steps of BO as an MDP. The k superscript indicates that k evaluations of the objective have already been performed, so that $\mathcal{D}_k \in \Omega^k \times \mathbb{R}^k$.

- **Horizon:** our horizon is h, which can be the full or restricted BO budget.
- **State space:** The state space of BO is all possible observation sets reachable from starting state $\mathcal{D}_k$ with h steps of BO.
- **Action space:** The action space of BO is Ω; actions correspond to sampling a point in Ω. Its transition probability and reward function are defined

recursively, as follows.

- **Transition probability:** The transition probabilities in BO from state $\mathcal{D}_t$ to state $\mathcal{D}_{t+1}$, where $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\mathbf{x}_{t+1}, y_{t+1})\}$, given an action $\mathbf{x}_{t+1}$, is defined as:

$$P(\mathcal{D}_t, \mathbf{x}_{t+1}, \mathcal{D}_{t+1}) \sim \mathcal{N}(\mu^{(t)}(\mathbf{x}_{t+1}; \mathcal{D}_t), K^{(t)}(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}; \mathcal{D}_t)). \tag{3.1}$$

  In other words, the probability of transitioning from $\mathcal{D}_t$ to $\mathcal{D}_{t+1}$ is the probability of sampling y_{t+1} from the posterior of $\mathcal{GP}(\mu^{(t)}, K^{(t)})$ at $\mathbf{x}_{t+1}$.

- **Reward:** The reward function in BO we consider is derived from the the EI criterion [Jones et al., 1998b]. Let y_t^* be the minimum observed value in the observed set $\mathcal{D}_t$, i.e., $y_t^* = \min\{y_0, \ldots, y_t\}$. Then our reward is expressed as:

$$R(\mathcal{D}_t, \mathbf{x}_{t+1}, \mathcal{D}_{t+1}) = (y_t^* - y_{t+1})^+ \equiv \max(y_t^* - y_{t+1}, 0). \tag{3.2}$$

In the BO setting, a policy is a sequence of decision rules, each of which selects one of the next h BO evaluations. The value function of a policy can be expressed as the total reduction in the objective function over h steps.

$$\begin{aligned} V_h^\pi(\mathcal{D}_k) &= \mathbb{E}\Big[\sum_{t=k}^{k+h-1} R(\mathcal{D}_t, \pi_t(\mathcal{D}_t), \mathcal{D}_{t+1})\Big] \\ &= \mathbb{E}\Big[\sum_{t=k}^{k+h-1} (y_t^* - y_{t+1})^+\Big] \\ &= \mathbb{E}\Big[(y_k^* - y_{k+h})^+\Big]. \end{aligned} \tag{3.3}$$

The optimal policy is then the sequence of decision rules that yields the highest reduction in the objective function over h steps. In the case when $h = 1$, the optimal policy is determined by maximizing the expected improvement acquisition function:

$$\pi_0^* = \arg\max_{\Omega} \mathrm{EI}(\mathbf{x}),$$

Algorithm 3 Non-myopic, MDP Bayesian Optimization

input: horizon h, budget b, initial observation set $\mathcal{D}_{b_i}$ of size b_i
Place a GP prior on f, update posterior with $\mathcal{D}_{b_i}$
for $k = b_i, b_i + 1, \ldots, b - 1$ **do**
    Learn GP hyperparameters, update posterior
    Calculate optimal policy $\pi^* = \arg\max_\pi V^\pi_{\min(h,b-k)}(\mathcal{D}_k)$
    Sample value y_{k+1} at point $\mathbf{x}_{k+1}$ according to π^*
    Update observation set $\mathcal{D}_{k+1} = \mathcal{D}_k \cup \{(\mathbf{x}_{k+1}, y_{k+1})\}$
end for
Return: $\mathbf{x}_i$ for which the minimum y_i value was observed

This can be shown by expanding the value function, its reward function, and its transition probability for horizon 1:

$$\begin{aligned}
\pi_{EI} &= \arg\max_\pi V_1^\pi(\mathcal{D}_k) \\
&= \arg\max_{\mathbf{x}\in\Omega} \mathbb{E}\left[(y_k^* - y_\mathbf{x})^+\right] \\
&= \arg\max_{\mathbf{x}\in\Omega} \mathrm{EI}(\mathbf{x} \mid \mathcal{D}_k),
\end{aligned}$$

where the starting state is $\mathcal{D}_k$. More generally, the EI acquisition function at any point $\mathbf{x}$ is equivalent to the value function of the policy $\pi(\mathcal{D}_k) = \mathbf{x}$. EI is a greedy policy, which may in part explain why it has empirically greedy behavior; it is unaware of potential reductions in the objective function down the road, and thus heavily favors exploitation over exploration. In the next section of this chapter, we attempt to remedy this.

3.5 Non-myopic BO

We say that BO is non-myopic if it uses the optimal policy, or an approximation thereof, for $h > 1$. We formalize this notion in Algorithm 3, in which an optimal policy is computed for the remaining budget (itself an input argument),

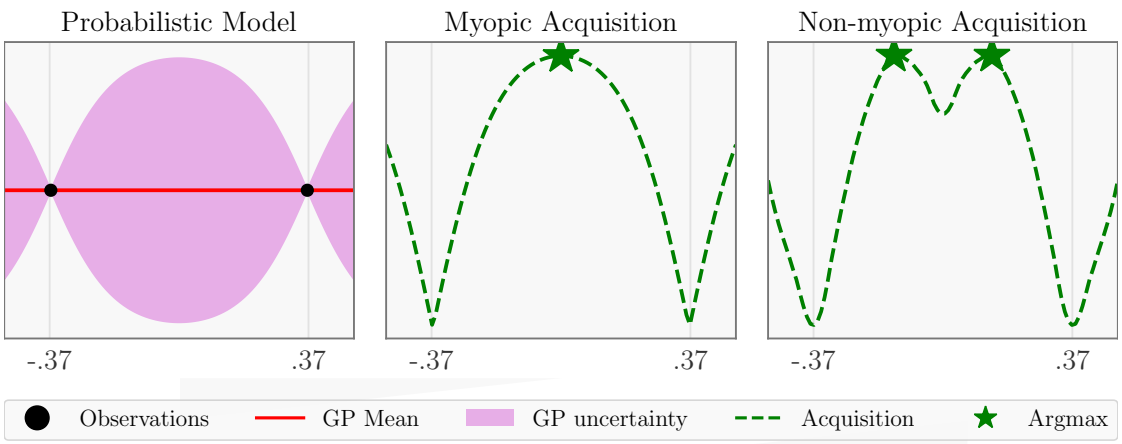


Figure 3.2: Left: the GP prior using a constant mean and Matérn 5/2 kernel with $\ell = 0.2$. Middle: the myopic acquisition function prefers sampling at the origin. Right: the non-myopic acquisition function prefers sampling away from the origin.

and used to determine the next BO evaluation.

In Figure 3.2, we illustrate the strength of non-myopic BO on a carefully chosen toy problem. We consider an objective function with two samples evenly spaced from the origin, whose sampled values (plotted in black) are both zero. Any GP model fitting the observed data therefore must have a mean of zero (plotted in red) with an uncertainty region (plotted in magenta) maximized around the origin. Because the mean is zero, the value of the uncertainty region is directly correlated with the value of the expected improvement acquisition function; maximizing EI is thus equivalent to maximizing the variance.

If we have a budget one one left, we sample according to the argmax of EI, which is the optimal policy for $h = 1$. This leads us to sample at the origin. This is shown in the middle image of 3.2, in which the EI acquisition function is traced out in green and its argmax is marked with a green star. If we have a budget of two left, sampling according to the argmax of EI is no longer the correct thing to do. This can be seen in the right image of 3.2, in which we trace

out the value function of for $h = 2$ in green and mark the optimal policies with two green stars. The behavior of the optimal policy is clearly different from EI's and prefers sampling at evenly spaced points in expectation. This leads to higher expected reduction in the objective function over two BO steps. Note that due to the symmetry of the problem, the optimal policy is not unique. In practice, a tiebreaking mechanism would be required to pick a consistent optimal policy should there be multiple.

3.6 Conclusion

We have presented BO as a classic MDP and shown a small example of how optimal MDP policies empirically improve BO performance. There is much work that can still be done in just this MDP formulation alone. We might extend it to cover the batch, multi-objective, and cost-constrained variants of BO. With the exception of the cost-constrained variant, which we devote Chapter 5 and some of Chapter 6 to, we haven't investigated either the batch or multi-objective setting in detail. The batch setting is rather straightforward, and requires an extension of the action space from Ω to Ω^b, where b is the batch size. The resulting MDP is otherwise unchanged, and therefore Bellman's optimality condition still holds. The multi-objective setting (in which the objective is vector-valued) is a more interesting, because the reward function, and therefore the value function, are be vector valued as well. The resulting MDP is known as a multi-objective MDP (MOMDP), and is a somewhat well-studied object [Roijers et al., 2013]. The general idea, as with any multi-objective optimization problem, is to enumerate the *Pareto* frontier of optimal policies. Algorithms for doing so can be found in [Chatterjee et al., 2006].

In general, our MDP formulation, while principled, is difficult to solve exactly for all but the smallest problems; computing the optimal policy for BO though the classic dynamic programming algorithm is intractable. This is because the BO state and action spaces are uncountably infinite. Naively discretizing the state and action spaces is also intractable, due to the high dimensionality of said spaces. There are other exact methods to solve finite-horizon MDPs, such as *Value Iteration* or *Policy Iteration* [Bertsekas, 1995], which are similarly intractable—we do not cover them in this dissertation for brevity's sake. In Chapter 4, we discuss efficient strategies to approximate optimal policies which lead to good performance in practice.

CHAPTER 4

EFFICIENT STRATEGIES FOR NON-MYOPIC BAYESIAN OPTIMIZATION

4.1 Introduction

Calculating the optimal policy for BO is difficult, and requires solving an infinite-dimensional dynamic program [Powell, 2007, Frazier, 2018a]. Rollout [Bertsekas, 1995, Bertsekas, 2005, Bertsekas, 2010] is a popular method of approximating an optimal policy. In the context of BO, rollout simulates future BO realizations and their corresponding values using the GP model, and then averages them. This average defines a rollout acquisition function, which is considered the state-of-the-art in non-myopic BO [Frazier, 2018a, Wu and Frazier, 2019]. While more practical than the calculating the optimal policy, rollout acquisition functions are still computationally expensive to the extent that suggesting the next evaluation can take several hours [Wu and Frazier, 2019]. In this chapter, we present efficient methods to compute these rollout acquisition functions:

- We compute rollout acquisition functions via Monte Carlo integration, and use variance reduction techniques —quasi-Monte Carlo, common random numbers, and control variates— to decrease the estimation error by several orders of magnitude.
- We provide experimental results, which suggest rollout acquisition functions perform better on multimodal problems.
- We examine the impact of model mis-specification on performance and show that increasing the horizon also increases sensitivity to model error.

Non-myopic BO frames the exploration-exploitation trade-off as a balance of immediate and future rewards. The strength of this approach is demonstrated in Figure 4.1, in which we compare EI and KG to a non-myopic acquisition function on a carefully chosen objective. The GP has a region of uncertainty on the left

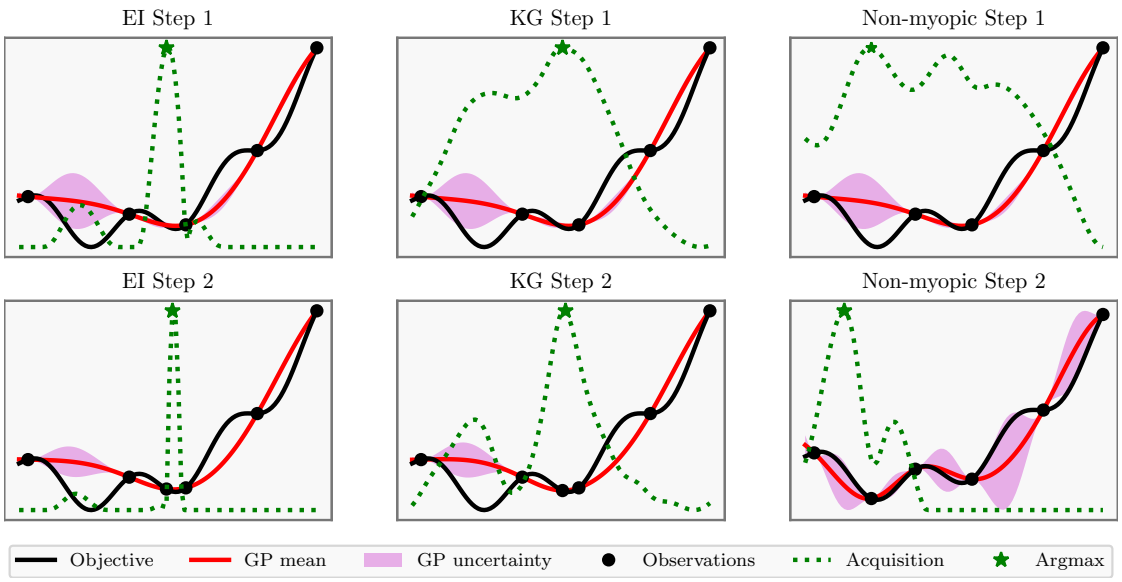


Figure 4.1: Comparing EI (*left*), KG (*middle*), and a rollout acquisition function (*right*) on a carefully chosen objective for two steps of BO. We observe five values of $f(x) = \sin(20x) + 20(x - 0.3)^2$. For each acquisition function, we perform two steps of BO. EI will ignore the left region and instead greedily evaluate twice in a sub-optimal location. KG is less greedy, but will nonetheless evaluate similarly to EI. The rollout acquisition function will evaluate the region of high uncertainty and thus identify the global minimum.

containing the global minimum and a local minimum at the origin. We run two steps of BO, updating the posterior each step. EI and KG behave greedily by sampling twice near the origin, while the non-myopic approach uses one evaluation to explore the uncertain region, subsequently identifying the global minimum and converging faster than either EI or KG.

4.2 Rollout policies

In the context of BO, rollout policies [Wu and Frazier, 2019], which are sub-optimal but yield promising results, are a tractable alternative to optimal policies. Our explanation of rollout for BO follows. For a given current state $\mathcal{D}_k$, we

denote our base policy $\tilde{\pi} = (\tilde{\pi}_0, \tilde{\pi}_1, \ldots, \tilde{\pi}_{h-1})$. We introduce the notation $\mathcal{D}_{k,0} \equiv \mathcal{D}_k$ to define the initial state of our MDP and $\mathcal{D}_{k,t}$ for $1 \leq t \leq h$ to denote the random variable that is the state at each decision epoch. Each individual decision rule $\tilde{\pi}_t$ consists of maximizing the base acquisition function Λ given the current state $s_t = \mathcal{D}_{k,t}$,

$$\tilde{\pi}_t = \arg\max_{\mathbf{x} \in \Omega} \Lambda(\mathbf{x} \mid \mathcal{D}_{k,t}).$$

Using this policy, we define the *rollout acquisition function* $\Lambda_h(\mathbf{x})$ as the rollout of $\tilde{\pi}$ to horizon h i.e., the expected reward of $\tilde{\pi}$ starting with the action $\tilde{\pi}(\mathcal{D}_k) = \mathbf{x}$:

$$\Lambda_h(\mathbf{x}_{k+1}) := \mathbb{E}\Big[V_h^{\tilde{\pi}}(\mathcal{D}_k \cup \{(\mathbf{x}_{k+1}, y_{k+1})\})\Big],$$

where y_{k+1} is the noisy observed value of f at $\mathbf{x}_{k+1}$. Λ_h is better than Λ in expectation for a correctly specified GP prior and for any acquisition function. This follows from standard results in the MDP literature.

Definition 4.2.1 *[Bertsekas, 1995] A policy π is sequentially consistent if, for every trajectory it generates starting at s_0:*

$$(s_0, a_0), (s_1, a_1), \ldots, (s_{h-1}, a_{h-1}),$$

it generates the following trajectory starting at s_1:

$$(s_1, a_1), (s_2, a_2) \ldots, (s_{h-1}, a_{h-1}).$$

In other words, sequential consistency requires the trajectory generated from applying π starting at s_i for horizon $h - i$ to be a sub-trajectory of the trajectory generated from applying π starting at s_0 for horizon h. Sequential consistency guarantees that rollout does better than its base heuristic.

Theorem 4.2.1 *[Bertsekas, 1995] A rollout policy π_{roll} does no worse than its base policy $\tilde{\pi}$ in expectation if $\tilde{\pi}$ is sequentially consistent i.e.,*

$$V_h^{\pi_{roll}}(s_0) \geq V_h^{\tilde{\pi}}(s_0).$$

That is to say, the value function of a rollout policy is always greater than or equal to the value function of the base policy.

Any acquisition function is sequentially consistent, so long as we consistently break ties if the acquisition function has multiple maxima —though this will not occur for generic problems. Thus, Theorem 4.2.1 implies that rolling out any acquisition function will do better in expectation than the acquisition function itself [Yue and Kontar, 2019]. We note that sequential consistency of the base policy is a sufficient condition for $V_h^{\pi_{roll}}(s_0) \geq V_h^{\tilde{\pi}}(s_0)$, but in fact is stronger than we need. A necessary condition is that of *sequential improvement*, which we do not cover here (sequential consistency implies sequential improvement) [Bertsekas, 2010, Bertsekas, 1995].

Thus, rolling out any acquisition function will do at least as well in expectation as the acquisition function itself. The next BO evaluation is determined by π_{roll}:

$$\mathbf{x}_{k+1} = \pi_{roll}(\mathcal{D}_k) = \arg\max_{\Omega} \Lambda_h(\mathbf{x}).$$

4.3 Computational methods for rollout

To rollout π once, we must do h steps of BO with Λ. Many such rollouts must then be averaged to reasonably estimate Λ_h, which is an h-dimensional integral. Estimation can be done either through explicit quadrature or MC integration, and is the primary computational bottleneck of rollout. We opt for MC.

In the context of rollout, MC estimates the expected reduction over h steps of BO using heuristic policy $\tilde{\pi}$:

$$\begin{aligned}\Lambda_h(\mathbf{x}_{k+1}) &= \mathbb{E}\left[V_h^{\tilde{\pi}}(\mathcal{D}_k \cup \{(\mathbf{x}_{k+1}, y_{k+1})\})\right] \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=k}^{k+h-1}(y_t^* - y_{t+1})^+.\end{aligned}$$

The distribution for y_{t+1} is a normal distribution whose mean and variance are determined by rolling out $\tilde{\pi}$ to horizon h and examining the posterior GP:

$$\begin{aligned}y_{t+1} &\sim \mathcal{N}(\mu^{(t)}(\mathbf{x}_{t+1}; \mathcal{D}_t), K^{(t)}(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}; \mathcal{D}_t)), \\ \mathbf{x}_{t+1} &= \pi(\mathcal{D}_t) = \arg\max_{\mathbf{x}\in\Omega} \Lambda(\mathbf{x} \mid \mathcal{D}_t).\end{aligned} \tag{4.1}$$

A sample path or trajectory in this context may be represented as the sequence $(\mathbf{x}_k, y_k), (\mathbf{x}_{k+1}, y_{k+1}), \ldots, (\mathbf{x}_{k+1}, y_{k+h})$ produced by Equation 4.1. We parameterize the vector of y values, $\mathbf{y}$, with an h-dimensional vector $\mathbf{z}$ drawn from $\mathcal{N}(0, \mathsf{I}_h)$. y_{t+1} is distributed according to $\mathcal{N}(\mu^{(t)}(\mathbf{x}_{t+1}; \mathcal{D}_t), K^{(t)}(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}; \mathcal{D}_t))$, so we map $\mathbf{z}_{t+1}$ to y_{t+1} by a simple scale-and-shift. This map is done sequentially from time step one to time step h. MC integration thus involves sampling N times from $\mathcal{N}(0, \mathsf{I}_h)$, mapping each of the samples, and averaging. The mapping step is equivalent to applying our rollout policy $\tilde{\pi}$, and is the dominant cost of integration.

Compared to other quadrature schemes, MC is well-suited to high-dimensional integration. MC converges at a rate of $\sigma/\sqrt{N}$, the standard deviation of the MC estimator, where σ^2 is the population variance and N is the total number of samples. MC's primary drawback is slow convergence. Increasing precision by an order of magnitude requires two orders of magnitude more samples. If σ is high, many samples may be required to converge. In this section, we focus on two strategies that significantly decrease the overhead of rollout: variance reduction and policy search.

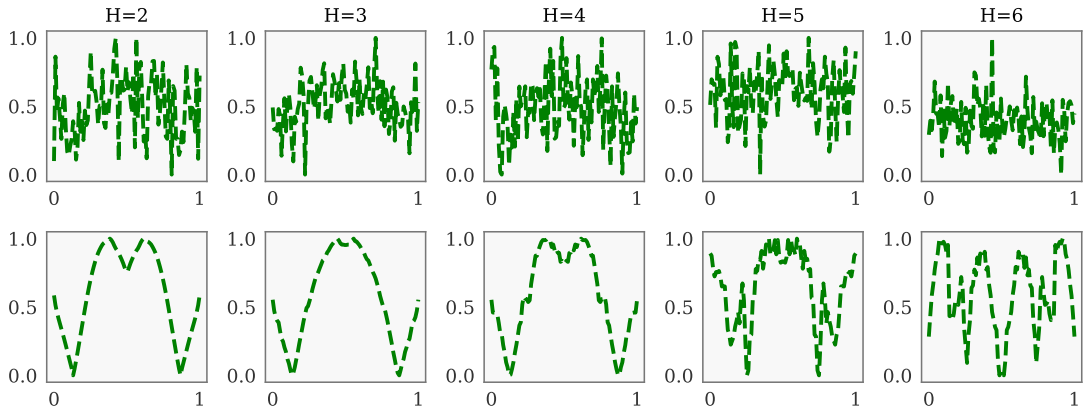


Figure 4.2: We calculate of a rollout acquisition function in 1D with different values of h using only $50h$ samples. (*Top*) The results of a standard MC estimator. (*Bottom*) The results of our estimator look far less noisy.

4.4 Efficient rollout via variance reduction

Unfortunately, while rollout is tractable and conceptually straightforward, it is still computationally demanding. Variance reduction is a class of methods that improve convergence by decreasing the variance of the estimator. Effective variance reduction methods can reduce σ by orders of magnitude [Owen, 2009]. We develop a new estimator that uses a combination of quasi-Monte Carlo, common random numbers, and control variates to significantly reduce the number of MC samples needed. The effectiveness of this estimator is seen in Figure 4.2, in which we compare the standard MC estimator (top row) to the one we developed (bottom row) for different horizons. Our estimator is clearly far more practical than the standard MC estimator, which is far too noisy to be of any practical use.

4.4.1 Quasi-Monte Carlo (QMC)

Quasi-Monte Carlo (QMC) integration may be seen as a middle ground between regular MC and quadrature. In MC, points are sampled with respect to the underlying probability distribution, and tend to cluster even when the distribution is uniform [Morokoff and Caflisch, 1994]. This means that they may not cover the domain of integration particularly efficiently. Conversely, quadrature covers the domain as thoroughly as possible, but as a result scales poorly with dimension. QMC attempts to cover the domain of integration efficiently but not thoroughly. Generally speaking, QMC is suitable for medium dimension problems, which [Caflisch, 1998] suggests are problems whose dimensions are between 4 and 300. This makes it well-suited for rollout, as dense quadrature is no longer possible as the horizon h increases.

Instead of sampling directly from the probability distribution, QMC instead uses a low-discrepancy sequence as its sample set [Caflisch, 1998, Morokoff and Caflisch, 1995].

Theorem 4.4.1 *[Caflisch, 1998]: QMC converges at a rate bounded above by* $\log(N)^h/N$*, where N is the number of samples and h is the integral's dimension.*

This bound stems from the well-known Koksma-Hlawka inequality [Caflisch, 1998], and is roughly linear for large N. In practice, this bound is often loose and convergence proceeds faster [Papageorgiou, 2003]. QMC is inapplicable if a low-discrepancy sequence does not exist for the target distribution.

In our case, the distributions we integrate over are normal, for which low-discrepancy sequences do exist. We generate low-discrepancy Sobolev sequences

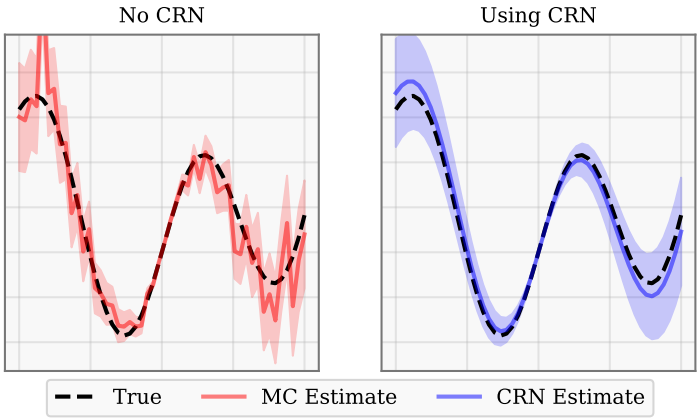


Figure 4.3: We estimate a function via MC. (*Left*) Standard MC without using CRN is noisy, and the estimate's argmin is not the function's. (*Right*) Using CRN makes a significant difference. The estimate is not only much smoother, but its argmin is also the same as the function's.

in the h-dimensional uniform distribution $\mathcal{U}[0,1]^h$ and map them to the unit multivariate Gaussian via the Box-Muller transform [Goodman, 2020]. This yields a low-discrepancy sequence for $\mathcal{N}(0, \mathrm{I}_h)$. Recall the parameterization of samples from $\mathcal{N}(0, \mathrm{I}_h)$ to sample rollout trajectories in Equation 4.1. We apply QMC by replacing the unit multivariate Gaussian samples with our low-discrepancy sequence.

4.4.2 Common random numbers (CRN)

CRN is used when estimating a quantity to be optimized over parameter $\mathbf{x}$, and is implemented by using the same random number stream for all values of $\mathbf{x}$. CRN does not decrease the point-wise variance of an estimate, but rather decreases the covariance between two neighboring estimates, which smooths out the function. Consider estimating $\mathbb{E}_y[f(y, \mathbf{x}_1)]$ and $\mathbb{E}_y[f(y, \mathbf{x}_2)]$ for two points $\mathbf{x}_1$ and $\mathbf{x}_2$ using N samples, with variances σ_1/N and σ_2/N respectively. We define the differences δ

and $\hat{\delta}$ as:

$$\begin{aligned}
\delta &= \mathbb{E}_y[f(y, \mathbf{x}_1)] - \mathbb{E}_y[f(y, \mathbf{x}_2)], \\
\hat{\delta} &= \mathbb{E}_y[f(y, \mathbf{x}_1) - f(y, \mathbf{x}_2)], \\
\text{Var}[\delta] &= (\sigma_1^2 + \sigma_2^2)/N, \\
\text{Var}[\hat{\delta}] &= (\sigma_1^2 + \sigma_2^2 - 2\,\text{Cov}[f(y, \mathbf{x}_1), f(y, \mathbf{x}_2)])/N.
\end{aligned}$$

Here, $\hat{\delta}$ uses the same number stream for both $\mathbf{x}_1$ and $\mathbf{x}_2$. If $\mathbf{x}_1$ and $\mathbf{x}_2$ are close and f is continuous in $\mathbf{x}$, $\text{Cov}[f(y, \mathbf{x}_1), f(y, \mathbf{x}_2)]) > 0$ which implies $\hat{\delta} < \delta$. The smoothing effect of common random numbers can be seen in Figure 4.3, in which we estimate the mean of a toy function without and with common random numbers, in red and blue respectively. the blue curve is early far smoother than the red. We've also shaded variance estimates; note that the pointwise variance is unchanged.

4.4.3 Control variates

The general idea behind control variates is to find a covariate $g(y)$ with the same distribution as $f(y)$ and a known mean. The quantity $c(y) = f(y) - \beta g(y)$, known as a regression control variate (RCV), is estimated, and then de-biased afterwards.

Theorem 4.4.2 *Consider the estimator* $\mathbb{E}[c(y)] = \mathbb{E}[f(y) - \beta g(y)]$. *Let* $Var[f(y)], Var[g(y)] = \sigma_f, \sigma_g$. $g(y)$ *is sufficiently correlated with* $f(y)$ *if:*

$$\beta^2 \sigma_g^2 - 2\beta Cov[f(y), g(y)] < 0.$$

If $g(y)$ *is sufficiently correlated with* $f(y)$*, then the estimator* $\mathbb{E}[c(y)]$ *is strictly more accurate than* $\mathbb{E}[f(y)]$ *i.e.,* $Var[c(y)] < Var[f(y)]$.

The optimal value that minimizes the variance of $c(y)$ is $\beta = \text{Cov}[f(y), g(y)]/\sigma_h^2$. In practice, both $\text{Cov}[f(y), g(y)]$ and σ_h^2 must be estimated.

In the case of $k > 1$ control variates, we consider a vector of control variates $\mathbf{g}(y) = [g_1(y), g_2, \ldots, g_k(y)]^T$. Our estimator will have the form $c(y) = f(y) - \beta^T \mathbf{g}(y)$, where β is an length k vector of constants of the form:

$$\beta = \Sigma_{\mathbf{g}}^{-1} \sigma_{\mathbf{g},f},$$

where $\Sigma_{\mathbf{g}}$ is the covariance matrix of $\mathbf{g}(y)$ and $\sigma_{\mathbf{g},f}$ is the vector of covariances between each variate and $f(y)$

In the context of BO, we opt to use covariates derived from existing acquisition functions with known means. EI and PI are straightforward options. We expect their value to be at least somewhat correlated with the value of the rollout acquisition function; a promising candidate point should ideally score highly among all acquisition functions, and vice-versa.

4.5 Fast policy search

While we have lowered the cost of evaluating the rollout acquisition function, there still remains the problem of its optimization. [Wu and Frazier, 2019] use the reparameterization trick to estimate the gradient of EI for horizon two and use stochastic gradient descent to maximize it. However, their method does not immediately extend to horizons larger than two.

Policy search is an alternative method for approximately solving MDPs that is often faster than rollout, in which a best performing policy is chosen out of a (possibly infinite) set of policies $\Pi = \{\pi_1, \pi_2, \ldots\}$ [Bertsekas, 1995]. It is

performed either by computing the expected reward for each policy in the set, or using a gradient-based method to maximize the expected reward given a parameterization of the policy set. In this paper, we use a finite policy set:

$$\Pi_{ps} = \{\pi(\mathcal{D}_k) \mid \pi(\mathcal{D}_k) := \arg\max \Lambda(\mathbf{x}|\mathcal{D}_k) \in A\},$$

where A is any arbitrary set of acquisition functions. We then select the best-performing policy:

$$\pi_{ps} = \arg\max_{\pi_i \in \Pi_{ps}} V_h^{\pi_i}(\mathcal{D}_k).$$

A less formal explanation follows: at every step of BO, we roll out each acquisition function in A on its argmax, and use the one with the highest h-step reward. A related approach by [Hoffman et al., 2011] employs a bandit strategy to switch between different acquisition functions. Our policy search method does not maximize the rollout acquisition function, and is thus significantly faster, though it likely reduces performance.

4.6 Experiments

Unless otherwise stated, we use a GP with the Matérn 5/2 ARD kernel [Snoek et al., 2012] and learn its hyperparameters via maximum likelihood estimation [Rasmussen and Williams, 2006]. When rolling out acquisition functions, we maximize them with L-BFGS-B using five restarts, selected by evaluating the acquisition on a Latin hypercube of 10d points and picking the five best. We use EI as the base policy. All synthetic functions are found in [Surjanovic and Bingham, 2020].

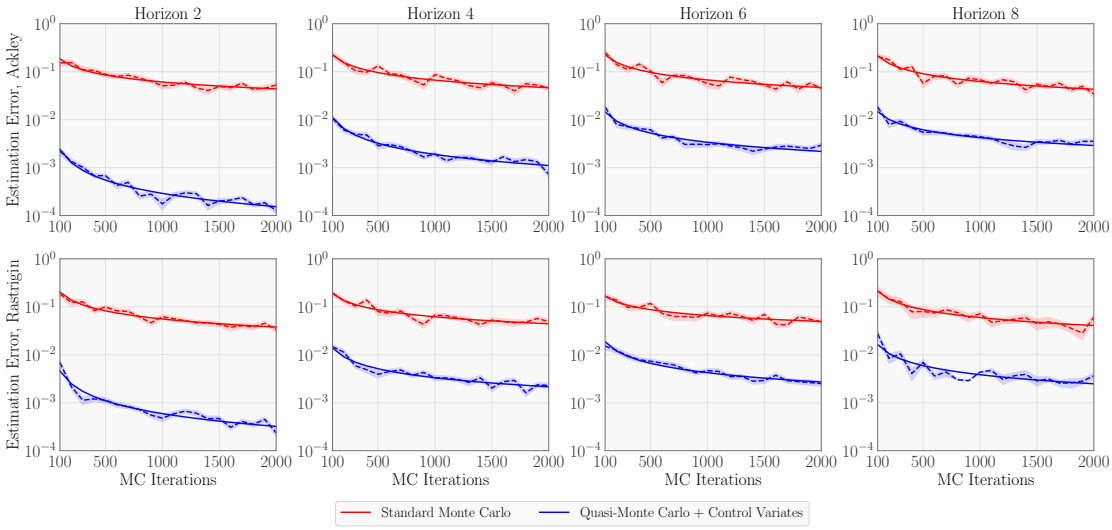


Figure 4.4: The estimation errors of MC (*red*) and our reduced-variance estimator (*blue*).

4.6.1 Variance reduction experiments:

We compare the estimation error and convergence rate between the standard MC estimator and our estimator. We take 2d random points in the domain and evaluate the Ackley and Rastrigin functions in 2D and 4D, respectively. We roll out EI for horizons 2, 4, 6, and 8, and calculate the variance of both estimators for MC sample sizes in $[100, 200, 300, \ldots, 2000]$, using 50 trials each. We take the ground truth to be estimation with 10^4 samples. The mean error of the standard MC estimator (*red*) and our reduced-variance estimator (*blue*) are plotted with dotted lines in Figure 4.4, with standard error shaded above and below. We also compute a best-fit line to each mean error, which is plotted with a solid line.

Table 4.1 summarizes our experimental results, and includes our estimates for the convergence rate of both estimators and the relative reduction in estimation error $\sigma/\hat{\sigma}$. Our estimator has significantly lower estimation error —the maximum reduction in estimation error we achieve is a factor of 410. Standard MC

Objective	Horizon	MC Rate	Our Rate	$\sigma/\hat{\sigma}$
Ackley	2	0.53	0.95	410
Ackley	4	0.52	0.82	63
Ackley	6	0.55	0.64	28
Ackley	8	0.53	0.54	26
Rastrigin	2	0.56	0.90	150
Rastrigin	4	0.48	0.63	31
Rastrigin	6	0.47	0.68	30
Rastrigin	8	0.42	0.64	25

Table 4.1: We estimate the convergence rate and error reduction $\sigma/\hat{\sigma}$ for the standard MC estimator and our estimator, for horizons 2, 4, 6, and 8 on the Ackley (2D) and Rastrigin (4D) synthetic functions.

clearly converges at a $N^{-1/2}$ rate. Our estimator converges like N^{-1} for smaller horizons, but its convergence rate drops as h increases. This is due to QMC's $\log(N)^h/N$ convergence. N is not large enough for longer horizons to exhibit N^{-1} convergence; increasing it past 2000 should yield N^{-1} convergence.

Another trend is the increase in estimation error as the horizon increases, which is expected given that the dimensionality of the underlying integral increases. Fortunately, the error seems to increase only linearly —and by a small constant— rather than exponentially, suggesting that MC samples proportional to h is sufficient to achieve a high quality of approximation. Finally, the reduction in estimation error levels off to around a factor of 25, suggesting that the correlation between the rollout acquisition function and our control variates decreases when h increases. A factor of 25 error reduction is still significant; the standard MC estimator would need a minimum of 600 times more samples to achieve comparable accuracy.

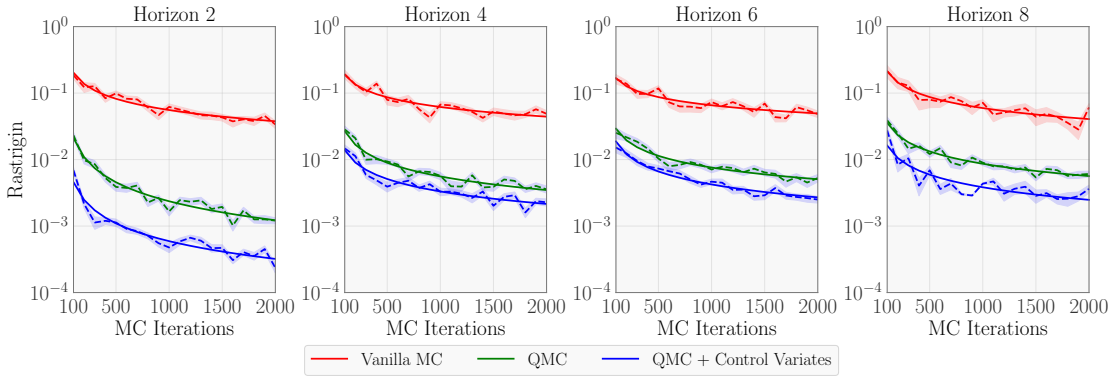


Figure 4.5: The estimation errors of MC (red), QMC (green), and QMC combined with control variates (blue). Generally speaking QMC contributes to a greater drop in variance. Note that the y-axis is logarithmically scaled.

Horizon	$\sigma_1/\hat{\sigma}$	$\sigma_2/\hat{\sigma}$
2	129	410
4	51	63
6	19	28
8	19	26

Table 4.2: We compare the reduction in variance between QMC and MC, denoted by $\sigma_1/\hat{\sigma}$, and QMC + control variates and MC, denoted by $\sigma_2/\hat{\sigma}$. While control variates contributes the greater reduction in variance for $h = 2$, QMC contributes the greater reduction in variance for $h > 2$. This is likely because our control variates, which are myopic acquisition functions, more closely resemble the rollout acquisition functionsi for small horizons.

4.6.2 Variance reduction ablation study

Recall that we combine QMC and control variates to achieve high levels of variance reduction in the resulting Monte Carlo estimator.

In Figure 4.5, we empirically measure the individual impact of QMC and control variates on the Ackley function. We roll out EI for horizons 2, 4, 6, and 8, and calculate the variance of estimators for MC sample sizes in $[100, 200, 300, \ldots, 2000]$, using 50 trials each. We compare the Vanilla MC estimator, a QMC estimator, and a QMC estimator that also uses control variates.

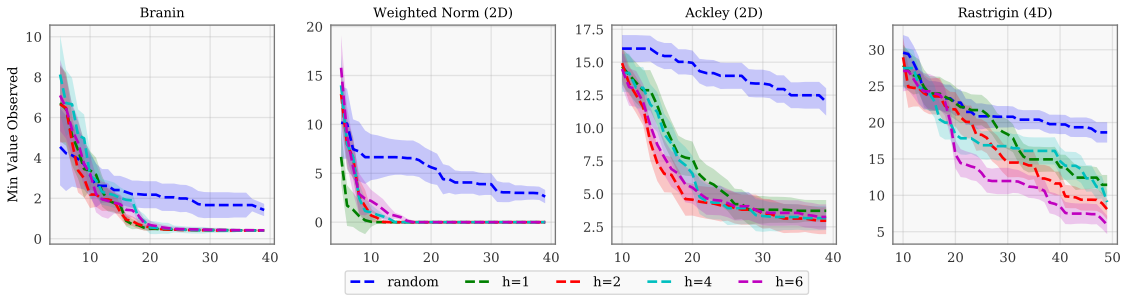


Figure 4.6: Empirically, looking longer horizons only seems to help on multimodal functions. On unimodal functions (not necessarily convex), there is little to no performance gain.

The underlying function is the Rastrigin function. As we mentioned before, the effectiveness of our control variates, which consist of myopic acquisition functions, are less effective as h increases. As a whole, QMC contributes to a greater drop in variance for horizons greater than 2, as seen in Table 4.2.

4.6.3 Full rollout on synthetic functions:

We roll out EI for $h = 2, 4$, and 6 on the Branin, weighted-two-norm (2D), Ackley (2D), and Rastrigin (4D) functions in Figure 4.6 using $200h$ MC samples, and compare to both standard EI and random search. To optimize the acquisition functions quickly, we employ the following strategy: we evaluate the acquisition function on a Sobol sequence of size 10d, as well as an additional point which is the argmax of EI. We then use this as an initial design and run BO for 50d more iterations. We run 50 iterations for each horizon and provide random search as a baseline. The mean results and the standard error are plotted in Figure 4.6.

On the Branin function, all horizons performed comparably and converge in 20 iterations. Rollout performed best on the Ackley and Rastrigin functions, which are multimodal. On the weighted norm function, which is strongly convex,

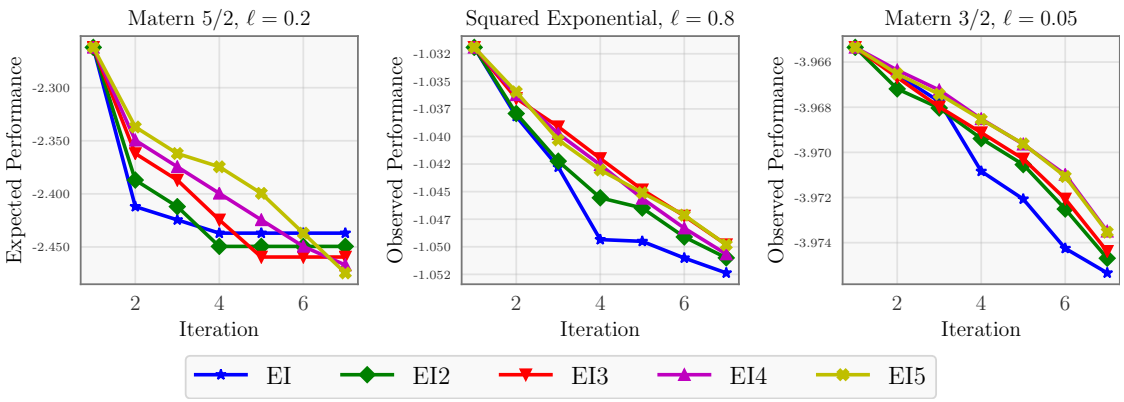


Figure 4.7: (*Left*) The expected performance of EI-based rollout for $h = 1, 2, 3, 4,$ and 5. (*Middle, Right*) The observed performance of rollout, given model error in the form of a smoother and less smooth kernel, respectively. When the model has large error, the resulting performance of non-myopic policies can be reversed from the expected performance.

EI converges within 10 iterations, and looking ahead further yielded poorer results. These results suggest that more exploratory acquisitions are needed for a multimodal objective, whereas more exploitative acquisitions suffice for reasonably simple objective functions.

4.6.4 The impact of model mis-specification:

We believe that any probabilistic model only supports a limited amount of look-ahead due to the effects of model error. Errors in the GP model result in errors to the MDP transition probabilities, which grow as they are propagated towards longer horizons. This likely renders long-horizon rollout ineffectual. We support this hypothesis by comparing the performance of policies in the MDP setting they were designed in with their observed performance on objectives drawn from a different MDP.

We do this by drawing objective functions from a GP with fixed kernel every

step of BO. More concretely, evaluating the objective function at any point $\mathbf{x}$ is performed by sampling from the GP posterior distribution at $\mathbf{x}$. Because policies are designed to maximize this MDP's reward for a fixed horizon and because the objective is drawn from the MDP itself, policies looking further ahead perform better by definition. We then draw objectives from a GP using a *different* kernel with a *different* lengthscale, and check if policies looking further ahead still perform better.

We rollout EI in 1D with a budget of seven and we model our objective with a GP using the Matérn 5/2 kernel with $\ell = 0.2$. In Figure 4.7, the left panel depicts expected performance of rollout for $h = 1, 2, 3, 4$, and 5. The middle panel and right panels depict observed performance of rollout when the we sample objectives from a GP that has a far smoother kernel (SE with $\ell = 0.8$) and far less smooth kernel (Matérn 3/2 with $\ell = 0.05$), respectively. All plots use 2000 independent BO trials with a fixed initial design.

The result is perhaps unsurprising; the ranking of the observed performance of policies completely reversed from the expected performance. Myopic BO performed the best; more generally, policies with shorter horizons outperformed those policies with longer horizons. This demonstrated sensitivity to model error suggests non-myopic BO must carefully strike a balance between model accuracy and look ahead horizon, and justifies use of modest horizons over the full BO budget. This confirms experimental results by [Yue and Kontar, 2019], who suggest looking ahead to modest horizons is preferable to longer horizons in practice.

4.6.5 Policy search: synthetic

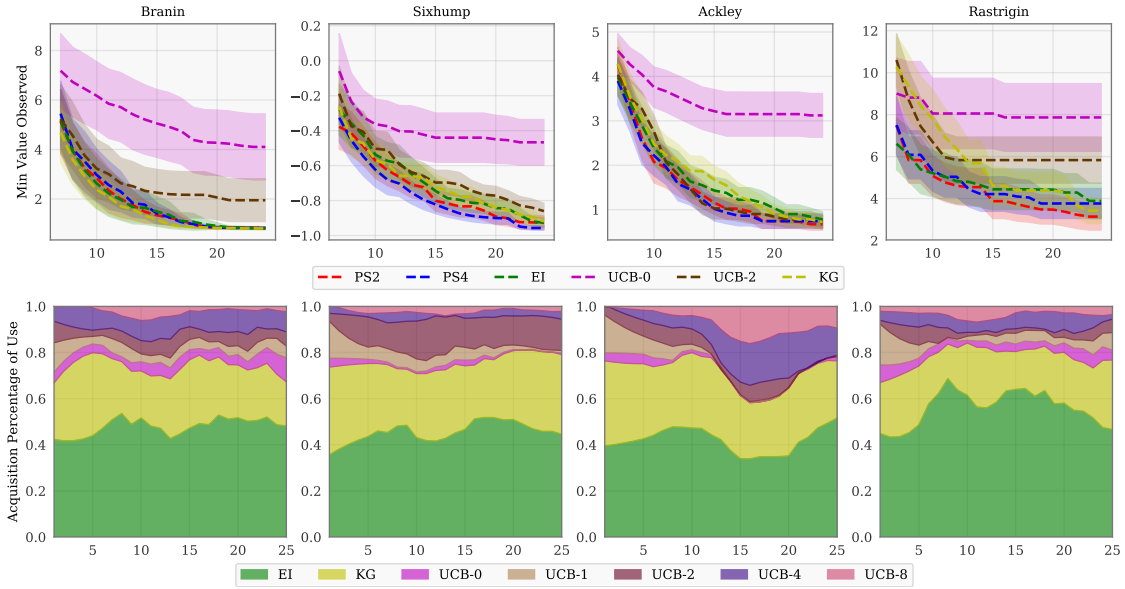


Figure 4.8: (*Top*) Policy search performs at least as well as the best acquisition function, if not better. (*Bottom*) For each of the corresponding objectives, we plot the percentage of use of each acquisition function per iteration for PS4. EI and KG are chosen more often than any of the UCB acquisition functions. The worst-performing acquisition, UCB-0, is chosen the least, suggesting correlation between an acquisition's performance and its percentage of use.

Policy search: We consider policy search (PS) with an acquisition set of EI, KG, and Upper Confidence Bound (UCB–κ) for $\kappa \in \{0, 1, 2, 4, 8\}$ [Snoek et al., 2012], which contains acquisitions that tend towards both exploitation and exploration. We run policy search for horizons 2 and 4 on the Branin, Sixhump, Ackley, and Rastrigin synthetic functions, all in 2D, using $200h$ MC samples. Experiments suggest that our policy search method performs at least as well as the best-performing acquisition in Π_{ps}.

We apply the variance reduction techniques —QMC and control variates— of Chapter 4 to the rollouts to significantly reduce the number of samples needed to evaluate their value function. All acquisition functions are maximized via L-BFGS-B with five random restarts, except for KG, which uses grid search of size

900. The mean results and standard error over 50 trials are plotted in Figure 4.8, in which policy search for horizons 2 and 4, labeled PS2 and PS4 respectively, do better or on par with the best-performing acquisition function. This robustness is a key strength of policy search, as the performance of each acquisition function is often problem-dependent.

We also examine the choice of acquisition function as a function of iteration. The percentage of use of each acquisition function is shown for its corresponding objective, and for plotting purposes we smooth the percentage with a box filter of size five. EI and KG are chosen more often than the other acquisition functions. UCB-0, the worst performing method representing a pure exploitation policy, is chosen significantly less than others, while UCB-2 was chosen the most often out of the UCB family. Of particular interest is the Ackley function (third column, Figure 4.8): when UCB-2 starts to outperform the other acquisitions functions, a clear spike in its percentage of use is seen in the corresponding histogram.

4.6.6 Policy search: NAS benchmark

The NAS benchmark is a tabular benchmark containing all possible hyperparameter configurations evaluated for a two-layer multi-layer perceptron on different datasets [Klein and Hutter, 2019]. We run policy search on the NAS benchmarks, optimizing the perceptrons' layer sizes, batch size, and training epochs.

The exact search space we consider is:

- Batch size in $\{8, 16, 32, 64\}$.
- Epochs in $\{10, 20, 30, 40, 50, 60, 70, 80, 90, 100\}$.

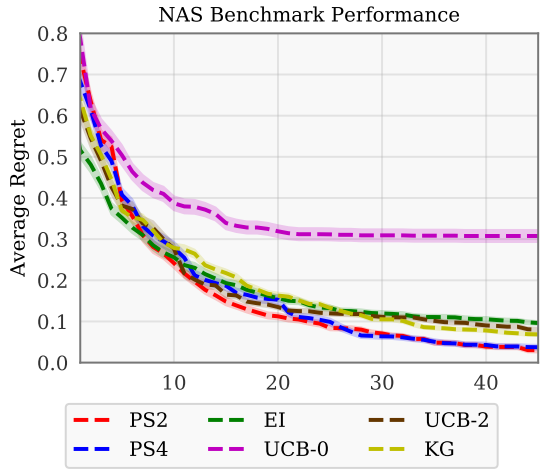


Figure 4.9: We compare the performance of policy search for horizons 2 and 4 in red and blue, respectively, with that of EI, UCB0, UCB2, and KG. PS2 and PS4 outperform the others after about 20 iterations.

- Layer 1 width in $\{16, 32, 64, 128, 256, 512\}$.
- Layer 2 width in $\{16, 32, 64, 128, 256, 512\}$.

The resulting search space is four-dimensional. We optimize over the unit hypercube $[0, 1]^4$ and scale and round evaluation points to the corresponding NAS search space entry. Note that the NAS benchmark contains other hyperparameters as well, which we set to the default. These include the activation functions (default: tanh), the dropout (default: 0), the learning rate (default: 0.005), and the learning rate schedule (default: cosine).

The datasets in the NAS benchmark are all classification tasks taken from the UC Irvine repository for machine learning datasets. We run our method on all four in the NAS benchmark: *Naval*, *Tele*, *Protein*, and *Splice*, for 60 BO iterations. The achievable classification error for each dataset is different, so we compare methods by regret, which is defined as:

$$\frac{y_{init} - y_{best}}{y_{init}},$$

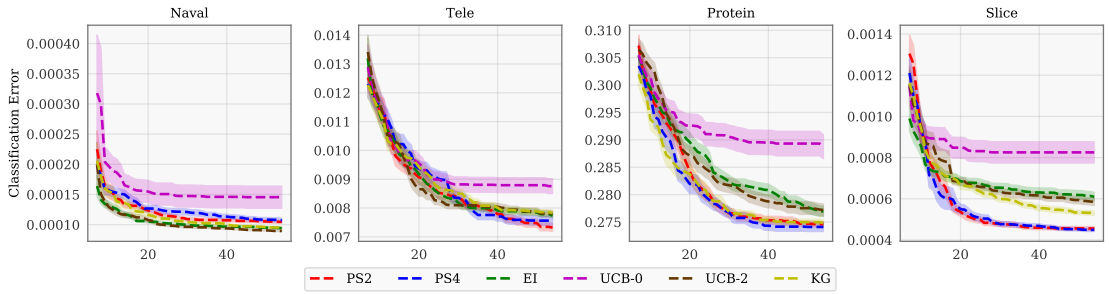


Figure 4.10: The classification error achieved by PS2 and PS4 is largely on par with, if not better than, the performance of EI, KG, and UCB variants. The only exception is the *Naval* dataset.

where y_{init} is the starting value during optimization and y_{best} is the best observed value during iteration so far. For each dataset, we run BO using EI, KG, UCB0, UCB2, and our policy search methods for horizons 2 and 4, labeled PS2 and PS4 respectively. We replicate BO runs 50 times. In Figure 4.9, we plot the average regret among the four datasets, and PS2 and PS4 beat the competing methods. In Figure 4.10, we plot the the individual classification errors for further clarity. We find the performance of both PS2 and PS4 performance are largely on par with, if not better than, the performance of EI, KG, and UCB variants.

4.7 Conclusion

We have shown that a combination of quasi-Monte Carlo, control variates, and common random numbers significantly lowers the overhead of rollout in BO. We have introduced a policy search which further decreases computational cost by removing the need to maximize the rollout acquisition function. Finally, we have illustrated the penalties incurred by using inaccurate GP models in the non-myopic setting.

This work raises several interesting research directions. Decreasing the variance of our estimator may be possible with additional variance reduction methods such as stratified or antithetic sampling. Developing a more comprehensive policy search space, such as a parameterized set of all convex combinations of acquisition functions, may further strengthen the policy search performance.

CHAPTER 5

NON-MYOPIC, COST-CONSTRAINED BAYESIAN OPTIMIZATION

5.1 Introduction

While BO budgets are typically given in iterations, this implicitly measures convergence in terms of iteration count and assumes each evaluation has identical cost. For many BO applications in practice, evaluation costs may vary in different regions of the search space. In hyperparameter optimization, the time spent on neural network training increases quadratically with layer size; in clinical trials, the monetary cost of drug compounds vary; and in optimal control, controllers have differing complexities. In these cases, standard BO is often insufficient. *Cost-constrained BO* measures convergence with alternative cost metrics such as time, money, or energy, for which standard BO methods are unsuited. In this chapter, we extend non-myopic BO to handle the cost-constrained setting. We loosely represent cost-constrained BO as the following (ill-defined) constrained optimization problem:

$$\begin{aligned} &\min_{\mathbf{x}\in\Omega} f(\mathbf{x}), \\ &\text{subject to } \sum_{i=1}^{n_{\mathbf{x}}} c(\mathbf{x}_i) \leq \tau. \end{aligned} \tag{5.1}$$

Just as in standard BO, $f(\mathbf{x})$ is a black box function that can be queried at points $\mathbf{x}$. However, in the cost-constrained setting, $f(\mathbf{x})$ outputs not only its value, but also the cost associated with evaluation, as determined by a black box cost function $c(\mathbf{x})$. The cost function $c(\mathbf{x})$ itself is also assumed to be continuous. Our goal is to minimize $f(\mathbf{x})$ subject to the constraint that the total cost of all $n_{\mathbf{x}}$ evaluations for some integer does not exceed a given budget τ.

Equation 5.1 is an ill-defined optimization problem, and is only meant to provide intuition. We will formalize it later in this chapter as an instance of a *constrained Markov decision process* (CMDP), which likely comes as no surprise

for the reader. We then similarly extend notions of rollout from Chapter 4 to the cost-constrained setting. Finally, we demonstrate its superior performance on a set of practical hyperparameter optimization problems. To our knowledge, there has been no work as of this dissertation's writing in the area of non-myopic, cost-constrained BO.

5.2 Motivation and Related Work

BO's sample efficiency leads to fast convergence only if evaluations cost the same, an assumption that is often not true in practice. Motivated by this, in this chapter we develop cost-constrained BO acquisition functions, with the overall aim of improving convergence when measured by cost. This cost may be time, energy, or money, and the goal is to minimize the objective given a known cost budget. Cost-constrained BO is an important problem, and we argue that the large majority of BO problems are in fact, cost-constrained. Figure 5.1 illustrates this by randomly evaluating 5000 hyperparameter configurations for five popular hyperparameter optimization problems. Unsurprisingly, resulting evaluation times vary, often by an order of magnitude or more. Moreover, the bulk of each problem's search space tends to be cheap, suggesting significant cost savings may be achieved by using a cost efficient rather than a sample efficient optimizer.

Most prior approaches to cost-constrained BO occur in the *grey-box* setting, in which additional information about the objective is available. *Multi-fidelity* BO is one such widely studied approach in which fidelity parameters $s \in [0, 1]^m$, such as iteration count or grid size, are assumed to be a noisy proxy for high-fidelity

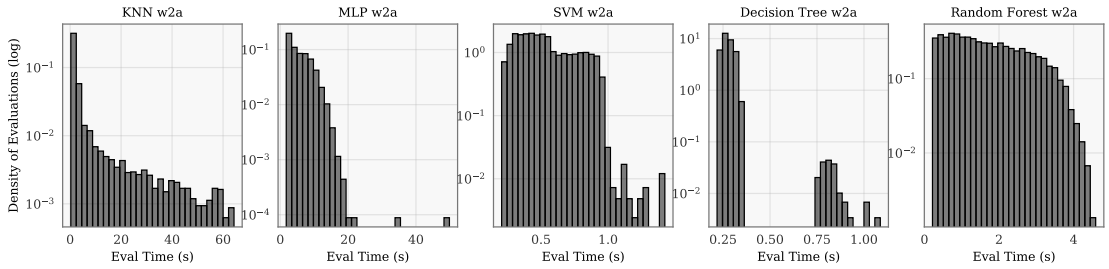


Figure 5.1: Runtime distribution, log-scaled, of 5000 randomly selected points for the K-nearest-neighbors (KNN), Multi-layer Perceptron (MLP), Support Vector Machine (SVM), Decision Tree (DT), and Random Forest (RF) hyperparameter optimization problems, each trained on the w2a dataset. The runtimes vary, often by an order of magnitude or more.

evaluations [Forrester et al., 2007, Kandasamy et al., 2017, Poloczek et al., 2017, Wu and Frazier, 2019]. Increasing s decreases noise at the expense of runtime. Multi-fidelity methods are often application-specific. For example, Hyperband [Li et al., 2017] and its BO variants [Falkner et al., 2018, Klein et al., 2016, Klein et al., 2017] cheaply train many neural network configurations for only a few epochs, and then train a selected subset for further epochs. In *multi-task* BO, hyperparameter optimization is run on cheaper training sets before more expensive ones. [Swersky et al., 2013] introduce a cost-constrained, multi-task variant of entropy search to speed-up optimization of logistic regression and latent Dirichlet allocation. Cost information is input as a set of cost preferences (e.g., parameter $\mathbf{x}_1$ is more expensive than parameter $\mathbf{x}_2$) by [Abdolshah et al., 2019], who develop a multi-objective, constrained BO method that evaluates cheap points before expensive ones, as determined by the cost preferences, to find feasible, low-cost solutions.

These prior methods outperform their black-box counterparts by evaluating cheap proxies or cheap points before carefully selecting expensive evaluations. This *early and cheap, late and expensive* strategy is accomplished by leveraging

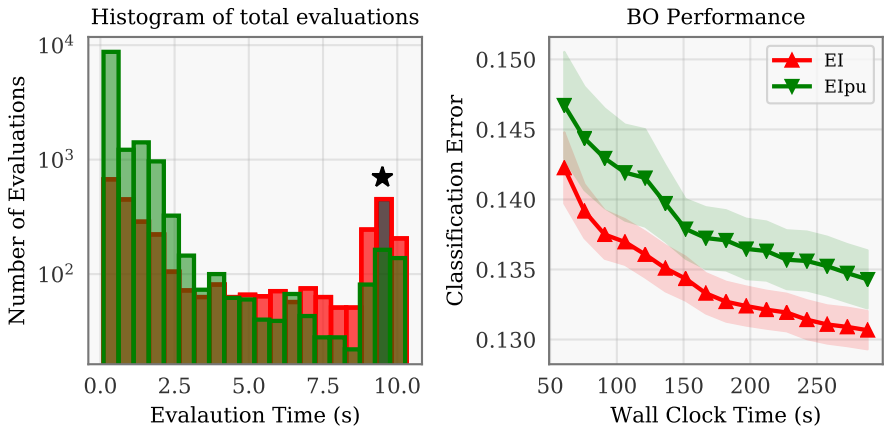


Figure 5.2: We run EI and EIpu on KNN, 51 times each. *Left:* EIpu evaluates many more cheap points than EI, which evaluates more expensive points. The optimum's cost, one of the most expensive points, is a black star. *Right:* EIpu performs poorly as a result.

additional cost information inside the optimization routine. While these methods demonstrate strong performance, they sacrifice generality and do not apply to black-box BO. To our knowledge, cost-constrained BO in the general black-box setting has not been thoroughly investigated. The de-facto standard in this setting is to normalize the acquisition by a GP cost model [Snoek et al., 2012]. This extends EI to *EI per unit cost (EIpu)*:

$$\text{EIpu}(\mathbf{x}) := \frac{\text{EI}(\mathbf{x})}{c(\mathbf{x})},$$

which is designed to balance the objective's cost and evaluation quality. [Snoek et al., 2012] showed that EIpu can boost performance on a variety of HPO problems.

EIpu is not a bad acquisition function, insofar as possessing the correct units. However, in our benchmarks, it demonstrated underwhelming performance. Out of twenty HPO problems in Chapter 6, EIpu was worse than EI on nine. We illustrate EIpu's poor performance on certain problems in Figure 5.2, in which EIpu (green) is slower than EI (red) at HPO of a K-nearest-neighbor

model. The empirical optimum, namely the best point over all trials (black star), has high cost. As a result, dividing by the cost penalizes EIpu *away* from the optimum and diminishes its performance. This is evidenced by the evaluation time histograms: EIpu evaluates far more very cheap points compared to EI, which instead evaluates fewer but more expensive points. Due to its bias towards cheap points, EIpu and, more generally, dividing acquisition by cost, is likely to only display strong results when optima are relatively cheap. This is a problem in the black-box setting as we do not know the global optima's cost a priori. Intuitively, one can adversarially increase the optimum's cost to make EIpu underperform.

We argue that a more principled cost-constrained strategy can be developed through a CMDP framework that we introduce below.

5.3 Constrained Markov decision processes

A constrained Markov decision processes (CMDP) is an MDP with an additional set of cost constraints [Altman, 1999, Piunovskiy, 2006, Bertsekas, 2005]. These costs, like MDP rewards, are accumulated through state by action until a certain horizon. A CMDP extends an MDP, and is the collection $< T, \mathbb{S}, \mathbb{A}, P, R, C, \tau >$. T is the set of decision epochs and for our problem, we only consider the finite horizon setting, where $T = \{0, 1, \ldots, h-1\}$, for some integer $h < \infty$. The state space, $\mathbb{S}$, encapsulates all the information needed to model the system from time $t \in T$. $\mathbb{A}$ is the action space. Given a state $s \in \mathbb{S}$ and an action $a \in \mathbb{A}$, $P(s'|s,a) : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to [0,1]$ is the transition probability of the next state being s'. $R(s,a,s') : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}^+$ is the reward received for choosing action a from

state s, and ending in state s'. $C(s, a, s') : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ is the cost of choosing action a from state s, and ending in state s'. τ is the cost constraint, and we assume without loss of generality that it is a positive scalar value.

A decision rule, $\pi_t : \mathbb{S} \to \mathbb{A}$, maps states to actions at time t. A policy π is a series of decision rules $\pi = (\pi_0, \pi_1, \ldots, \pi_{h-1})$, one at each decision epoch. Given a policy π, a starting state s_0, and horizon h, recall the definition of an MDP value function $V_h^\pi(s_0)$:

$$V_h^\pi(s_0) = \mathbb{E}\Big[\sum_{t=0}^{h-1} R(s_t, \pi_t(s_t), s_{t+1})\Big].$$

We can define the cost function $C_h^\pi(s_0)$ in a similar way by replacing the reward with the cost:

$$C_h^\pi(s_0) = \mathbb{E}\Big[\sum_{t=0}^{h-1} C(s_t, \pi_t(s_t), s_{t+1})\Big].$$

The cost function measures total expected cost given a policy π, starting state s_0, and horizon h. The goal in a CMDP is to find the optimal policy subject defined as [Altman, 1999]:

$$\pi^* = \arg\max_\pi V_h^\pi(s_0),$$
$$\text{subject to } \; C_h^\pi(s_0) \leq \tau.$$

In other words, we want to determine the policy that maximizes the expected reward subject to having expected cost less than τ. We refer readers to the classic treatment of CMDPs by [Altman, 1999] for more information, and note that it uses significantly different notation from ours. In their presentation, they define the value and cost constraints exclusively in terms of policy vectors and not states and actions.

5.3.1 Feasible trajectories

The notion of *feasible trajectories* [Bertsekas, 2005] is important when discussing approximate CMDP solutions. Recall in the MDP case that a trajectory, an element in the set of all possible trajectories F, is the sequence:

$$(s_0, a_0), (s_1, a_1), \ldots, (s_{h-1}, a_{h-1}).$$

We say a trajectory is *feasible* if it doesn't violate its cost constraint τ:

$$\sum_{t=0}^{\ell} C(s_t, a_t, s_{t+1}) < \tau,$$

for any non-negative integer $0 \leq \ell \leq h-1$ less than horizon h. For consistency, we extend all feasible trajectories to have length h by introducing a dummy state and action which produce zero reward and cost (the formal equivalent of "standing still"). The set of all feasible trajectories is known as $G \subseteq F$.

Instead of explicitly defining a cost constraint $C(s_t, a_t, s_{t+1})$ in the CMDP setting, it is equivalent to instead replace it with the set of feasible trajectories G implicitly defined by $C(s_t, a_t, s_{t+1})$, and take the expectation of the reward function with respect to G instead of the transition probabilities.

5.4 Cost-constrained BO as a CMDP

We can express cost-constrained BO with non-uniform cost as a CMDP, which is analogous to the extension from MDP to CMDP. At a high level, we get a value y_t and a cost c_t from the objective at time t, and want to evaluate the point with maximal reduction in the objective over h steps without exceeding a given cost budget τ.

We note that our CMDP model is somewhat similar to the Bayesian optimization with resources (BOR) framework developed by [Dolatnia et al., 2016], who consider a partially observable MDP (POMDP) framework for BO when resource consumption of the objective varies, and when there might be multiple agents (such multiple laboratories or cloud instances) that can evaluate the acquisition function in parallel. However, our formulation is much narrower in scope, and considers only sequential evaluation. We cover parallel, cost-constrained BO in Chapter 6. A more formal description, some of which is identical our treatment of BO as an MDP, follows.

Given a cost function $c(\mathbf{x}) : \Omega \to \mathbb{R}^+$, a cost budget τ, and a GP prior over the observation set $\mathcal{D}_t$ with mean $\mu^{(t)}$ and kernel $K^{(t)}$, we model h steps of cost-constrained BO as a CMDP.

- **Horizon:** the horizon we consider is h.
- **State space:** The state space of BO is all possible observation sets reachable from starting state $\mathcal{D}_t$ with h steps of BO.
- **Action space:** The action space of BO is Ω; actions correspond to sampling a point in Ω. Its transition probability and reward function are defined recursively, as follows.
- **Transition probability:** The transition probabilities in BO from state $\mathcal{D}_t$ to state $\mathcal{D}_{t+1}$, where $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\mathbf{x}_{t+1}, y_{t+1})\}$, given an action $\mathbf{x}_{t+1}$, are defined as:

  $$P(\mathcal{D}_t, \mathbf{x}_{t+1}, \mathcal{D}_{t+1}) \sim \mathcal{N}(\mu^{(t)}(\mathbf{x}_{t+1}; \mathcal{D}_t), K^{(t)}(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}; \mathcal{D}_t)). \tag{5.2}$$

  In other words, the probability of transitioning from $\mathcal{D}_t$ to $\mathcal{D}_{t+1}$ is the probability of sampling y_{t+1} from the posterior of $\mathcal{GP}(\mu^{(t)}, K^{(t)})$ at $\mathbf{x}_{t+1}$.

- **Reward:** The reward function in BO we consider is derived from the the EI criterion [Jones et al., 1998b]. Let y_t^* be the minimum observed value in the observed set $\mathcal{D}_t$, i.e., $y_t^* = \min\{y_0, \ldots, y_t\}$. Then our reward is expressed as:

$$R(\mathcal{D}_t, \mathbf{x}_{t+1}, \mathcal{D}_{t+1}) = (y_t^* - y_{t+1})^+ \equiv \max(y_t^* - y_{t+1}, 0). \tag{5.3}$$

- **Cost function:** We use $c(\mathbf{x})$ as our cost function. We assume in this presentation that our cost is deterministic and state-independent; it only depends on the action. In practice, the cost function must be learned as well, and could certainly be modeled with an additional GP. In this case, the CMDP could be modified to account for cost uncertainty as well, which we do not consider in this chapter. Finally, we note that there are interesting applications for which the cost is state-dependent as well. For example, in the sensor set selection problem [Garnett et al., 2010] with a reconfiguration cost, the cost itself depends on the current sensor configuration.
- **Constraint:** For simplicity's sake, we assume a scalar constraint τ. However, we could extend this to a vector-valued constraint. For example, in as materials design, there might be a finite amount of each constituent component, each with its own budget [Abdolshah et al., 2019].

The expected total reward of a policy π is unchanged and expressed as:

$$\begin{aligned} V_h^\pi(\mathcal{D}_k) &= \mathbb{E}\Big[\sum_{t=k}^{k+h-1} R(\mathcal{D}_t, \pi_t(\mathcal{D}_t), \mathcal{D}_{t+1}) \Big] \\ &= \mathbb{E}\Big[\sum_{t=k}^{k+h-1} (y_t^* - y_{t+1})^+ \Big]. \end{aligned}$$

The expected total cost of π is expressed as:

$$C_h^\pi(\mathcal{D}_k) = \mathbb{E}\Big[\sum_{t=k}^{k+h-1} c(\pi_t(\mathcal{D}_t)) \Big].$$

We can represent a trajectory though this CMDP as the sequence:

$$(\mathbf{x}_k, y_k), (\mathbf{x}_{k+1}, y_{k+1}), \ldots, (\mathbf{x}_{k+1}, y_{k+h}).$$

A trajectory is feasible if:

$$\sum_{i=k}^{k+h} c(\mathbf{x}_i) \leq \tau.$$

5.5 CMDP rollout

CMDPs are considered far more difficult to solve than MDPs [Altman, 1999], and the standard dynamic programming approach of Chapter 3 does not extend trivially. Most importantly, Bellman's principle of optimality no longer applies. The optimal CMDP policy depends on the starting state and may require randomization[1] —indeed, the existence of an optimal policy is not guaranteed! The standard solution is to solve a large linear program in the state and actions spaces, but this is computationally intractable for all but the smallest problems. We opt to use rollout. Rollout in the CMDP setting is conceptually straightforward and very similar to standard rollout [Bertsekas, 2005]. The only difference is that CMDP rollout averages only feasible trajectories from G instead of all possible trajectories as standard rollout will do. In practice, this means that as we roll out the base policy, we terminate either once we reach the horizon or violate the cost constraint [Bertsekas, 2005].

There remains the question of what base policy we might use; rollout is only as good as its base policy. We develop a base policy by considering the following two cases:

[1] An optimal MDP policy is always determinstic.

- If the maximum of EI has cost $c(\mathbf{x}^*) = \tau$, and the total budget is equal to τ, then the following policy π is optimal in the CMDP sense:

$$\pi(\mathcal{D}_t) = \mathbf{x}^* = \arg\max_{\mathbf{x}\in\Omega} \mathrm{EI}(\mathbf{x} \mid \mathcal{D}_t).$$

- If the maximum of EI has cost $c(\mathbf{x}^*) = \tau$, there exists a point with arbitrarily small cost $c(\mathbf{x}_\epsilon) = \epsilon$, and the total budget is greater than or equal to $\tau + \epsilon$, then $\mathbf{x}_\epsilon$ should be evaluated before $\mathbf{x}^*$. In the limit, a point that is free to evaluate should always be evaluated first.

A reasonable base policy should, at the minimum, satisfy these two cases. To deal with the first case, maximizing EI must necessarily be the last step in our base policy. To deal with the second case, we note that maximizing EIpu for the first rollout iteration will result in the desired behavior. For simplicity's sake, we extend to maximization of EIpu until the last iteration. The base rollout policy $\tilde{\pi} = (\tilde{\pi}_0, \ldots, \tilde{\pi}_{h-1})$ that we consider is therefore:

$$\tilde{\pi}_t(\mathcal{D}_t) = \begin{cases} \arg\max_{\mathbf{x}\in\Omega} \mathrm{EIpu}(\mathbf{x} \mid \mathcal{D}_t), & t < h-1, \\ \arg\max_{\mathbf{x}\in\Omega} \mathrm{EI}(\mathbf{x} \mid \mathcal{D}_t), & t = h-1. \end{cases}$$

In other words, $\tilde{\pi}$ rolls out $h-1$ steps of EIpu followed by a last step of EI. If $h = 1$, then $\tilde{\pi}$ is equivalent to EI. Furthermore, this base policy is consistent with an early exploration, late exploitation strategy, which we know to be a promising heuristic from prior work; EIpu tends to select cheaper points and EI tends to select more expensive points. Therefore, $\tilde{\pi}$ starts by trying to select cheaper points and then ends with selecting a point that is possibly more expensive.

We define the non-myopic acquisition function $\Lambda_h(\mathbf{x})$ as the rollout of $\tilde{\pi}$ to horizon h i.e., the expected reward of $\tilde{\pi}$ starting with the action $\tilde{\pi}_0 = \mathbf{x}$:

$$\Lambda_h(\mathbf{x}_{k+1}) := \mathbb{E}\Big[V_h^{\tilde{\pi}}(\mathcal{D}_k \cup \{(\mathbf{x}_{k+1}, y_{k+1})\})\Big],$$

where y_{k+1} is the noisy observed value of f at $\mathbf{x}_{k+1}$. As with any acquisition function, the next BO evaluation is:

$$\mathbf{x}_{k+1} = \arg\max_{\Omega} \Lambda_h(\mathbf{x}).$$

Recall in the MDP setting from Theorem 4.2.1 that given a sequentially consistent base policy, rollout will perform better in expectation than the base policy itself. The same is true holds true in the CMDP setting.

Theorem 5.5.1 *[Bertsekas, 2005] In the CMDP setting, a rollout policy π_{roll} does no worse than its base policy $\tilde{\pi}$ in expectation if $\tilde{\pi}$ is sequentially consistent i.e.,*

$$V_h^{\pi_{roll}}(s_0) \geq V_h^{\tilde{\pi}}(s_0).$$

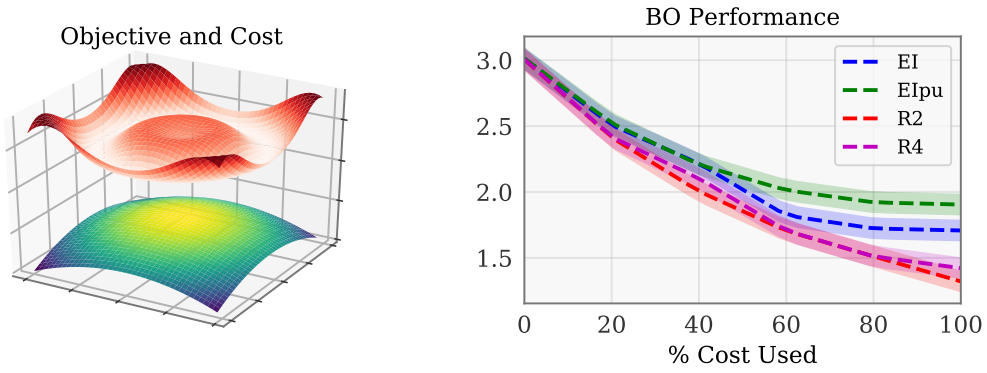
That is to say, the value function of a rollout policy is always greater than the value function of the base policy.

To guarantee sequential consistency of our acquisition function, we need only consistently break ties if the acquisition function has multiple maxima.

In Figure 5.3, we examine a carefully chosen synthetic example showcasing the strength of the rollout approach. We consider the cost-constrained optimization problem:

$$f(\mathbf{x}) = \|\mathbf{x}\|_2 \sin(2\pi\|\mathbf{x}\|_2),$$
$$c(\mathbf{x}) = 10 - 5\|\mathbf{x}\|_2,$$

in the domain $[-1, 1]^2$, and a budget of 150. The cost function has been designed so that its maximum aligns with the minimum of the objective. As we showed earlier, EIpu struggles with these types of problems. We run BO with EI, EIpu,



(a) The objective function (red) and cost (green). (b) Comparing BO performance

Figure 5.3: In this example, we examine a carefully chosen example showcasing the strength of the rollout approach. We consider the objective $f(\mathbf{x}) = \|x\|_2 \sin(2\pi\|x\|_2)$, the cost $c(\mathbf{x}) = 10 - 5\|x\|_2$, the domain $[-1, 1]^2$, and a budget of 150. The most expensive point is the global minimum. EIpu performs worse than EI, and both tend to get stuck in cheaper, local minimum. Our rollout policy for horizons 2 and for performs better than both EI and EIpu.

and rollout with our base policy 50 times and plot the results. This is seen on the right, in which EIpu (green) performs worse than EI (blue). However, rollout of our base policy, for horizons two and four in pink and red respectively, performs much better than both.

5.6 Experiments

We use rollout to perform hyperparameter optimization of different machine learning models under budget constraints. We use a GP with the Matérn 5/2 ARD kernel [Snoek et al., 2012] to model both the objective and the cost function, and learn hyperparameters via maximum likelihood estimation [Rasmussen and Williams, 2006]. When rolling out acquisition functions, we maximize them with L-BFGS-B using five restarts, selected by evaluating the

HPO problem	Budget	Cost Savings	EI	EIpu	Rollout 2	Rollout 4
KNN	800	60%	0.084	0.081	0.081	**0.075**
Decision Tree	15	45%	0.105	0.106	**0.092**	0.100
Random Forest	15	35%	0.117	0.118	**0.116**	0.118

Table 5.1: We list the HPO problem, budget, and the classification errors achieved by EI, EIpu, and rollut for horizons 2 and 4. We also bold the best-performing optimizer and provide the mean cost savings, which represents the time needed by rollout to achieve comparable results to both EI and EIpu. Rollout provides significant cost savings overall.

acquisition on a Latin hypercube of 10d points and picking the five best. Furthermore, we use the same variance reduction techniques described in Chapter 5 —QMC, common random numbers, and control variates— to significantly reduce the overhead of rollout. This is especially important in the cost-constrained setting because the optimizer overhead is important.

We compare rollout performance to EI and EIpu on HPO of three different models: K-nearest neighbors (KNN), decision trees, and random forests, with budgets of 800, 15, and 15 seconds respectively. These are relatively small problems, chosen due to the number of replications required to show statistical significance in the benchmarks. We summarize Table 5.6, in which we provide the classification errors achieved by each optimizer and bold the best. We also provide the mean cost savings, which represents the time needed by rollout to achieve comparable results to both EI and EIpu. In the following subsections, we provide more details on each of our three HPO problems.

5.6.1 K-nearest neighbors

The K-nearest neighbors algorithm (KNN) is a class of methods used for classification and regression of either spatially-orientated data or data with a known

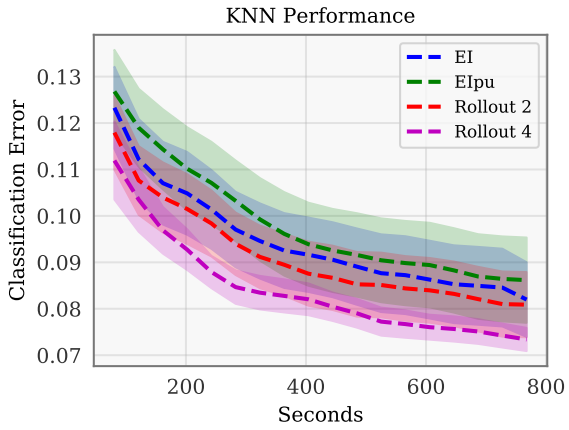


Figure 5.4: We compare KNN classification error among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4. Rollout performs significantly better than both EI and EIpu.

distance metric (i.e., data imbedded in a Hilbert space). In the case of regression, it outputs the plurality vote of its k nearest neighbors. In the case of regression, it outputs an average of the k nearest neighbors. A number of hyperparameters affect the performance of the KNN algorithm. Many KNN algorithms utilize some form of random projection to project down the data into low dimensions; in high dimensions, the KNN algorithm may perform poorly because points all appear far away from each other, thus making classification with a large number of neighbors difficult. The neighbor count itself is also a hyperparameter, as is the notion of distance.

We consider a $5d$ search space: dimensionality reduction percentage and type in [1e-6, 1.0] log-scaled and {Gaussian, Random}, respectively, neighbor count in {1, 2, ..., 256}, weight function in {Uniform, Distance}, and distance in {Minkowski, Cityblock, Cosine, Euclidean, L1, L2}. We perform hyperparameter optimization of a KNN classification model on the *a1a* dataset in the UCI data repository, and compare the reduction of classification errors over time among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4 in Figure 5.4. We

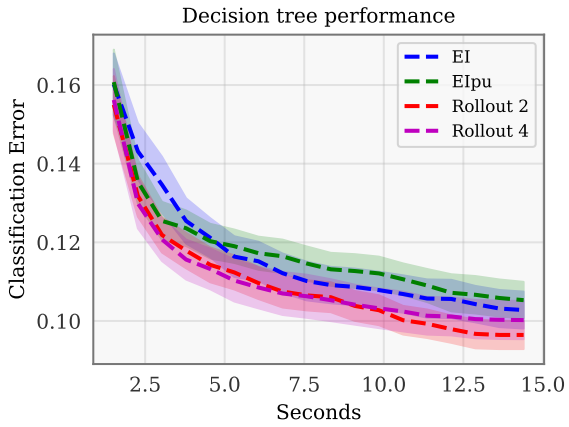


Figure 5.5: We compare decision tree classification error among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4. Rollout performs significantly better than both EI and EIpu.

see that cost-constrained rollout of horizon 4 performs better than both EI and EIpu, as well as the cost-constrained rollout of horizon 2, and achieves a similar result to EI and EIpu with only 40 percent of the budget.

5.6.2 Decision trees

Decision trees are one of the most popular predictive modeling in approaches used in statistics, data mining and machine learning. In the case of classification, leaves represent class labels and paths represent sets of features that lead to those class labels. During training, a tree is built by splitting the source set into subsets which constitute the successor children. The splitting is based on a threshold that maximizes some notion of information gain such as entropy. The depth of a decision tree is pre-specified.

We consider a $3d$ search space: tree depth in {1, 2, ..., 64}, tree split threshold in [0.1, 1.0] log-scaled, and split feature size in [1e-3, 0.5] log-scaled. We perform hyperparameter optimization of a decision tree classification model on the *a1a*

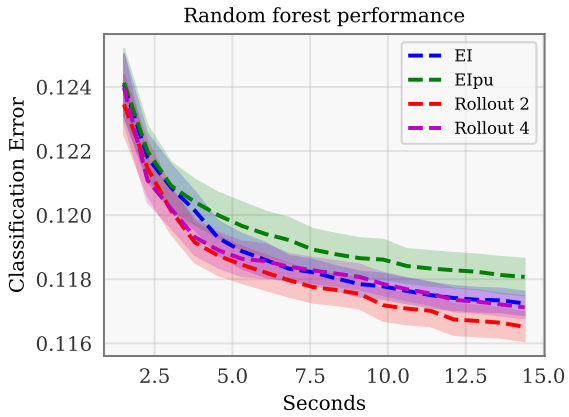


Figure 5.6: We compare random forest classification error among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4. Rollout performs significantly better than both EI and EIpu.

dataset in the UCI data repository, and compare the reduction of classification errors over time among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4 in Figure 5.5. We see that cost-constrained rollout of horizon 2 and 4 performs better than both EI and EIpu, and achieves a similar result to EI and EIpu with only 55 percent of the budget.

5.6.3 Random forest

A random forest is a set k of decision trees, and classifies based off the plurality decision generated from all its trees —this technique is generally known as *bagging*, and improves robustness in the classification algorithm. The hyperparameters in a random forest model are the same as those in a decision tree model, with the addition of an additional hyperparameter that controls the number of decision trees that must be trained.

We consider a $3d$ search space: number of trees in $\{1, 2, ..., 256\}$, tree depth in $\{1, 2, ..., 64\}$, and tree split threshold in $[0.1, 1.0]$ log-scaled. We perform

hyperparameter optimization of a decision tree classification model on the *a1a* dataset in the UCI data repository, and compare the reduction of classification errors over time among EI, EIpu, and our cost-constrained rollout for horizons 2 and 4 in Figure 5.6. We see that cost-constrained rollout of horizon 2 performs better than both EI and EIpu, and achieves a similar result to EI and EIpu with only 65 percent of the budget.

5.7 Conclusion

In this chapter, we have shown the importance of cost-constrained BO and formulated it as an instance of a constrained Markov decision process. We then developed a rollout algorithm using a cheap exploration, expensive exploitation base policy that performed better than both EI and EIpu on three hyperparameter optimization problems.

These investigations into cost-constrained BO are preliminary, and we believe there is much future research that can be done in this area. First, the overhead of the optimizer itself must be taken into account, especially in the context of hyperparameter optimization. Unfortunately, the overhead of approximately solving a CMDP was higher than the benchmarks we considered. In the next chapter, we consider simpler heuristics with a much lower overhead, and show that they also yield superior performance.

Second, we believe approximate solutions to CMDPs other than rollout are worth investigating. State aggregation and state truncation are classic methods in the MDP setting to reduces the state space according to the transition probabilities, and [Altman, 1999] recommends it as the de-facto approximation

method in the CMDP setting. Consequently, we might approximate our CMDP model through state aggregation and state truncation and then compute an exact solution via linear programming.

Finally, we have limited our discussion to the sequential BO setting. However, cost-constrained BO becomes significantly more complex in the batch setting, when evaluations are performed in parallel. We attempt to address this challenge in the following chapter.

CHAPTER 6

CARBO: PRACTICAL COST-CONSTRAINED BAYESIAN OPTIMIZATION

6.1 Introduction

There are several drawbacks to cost-constrained rollout. First, the rollouts themselves are expensive and may exceed the cost of the objective —which is the case in Chapter 5's experiments. In practical settings, the acquisition function needs to suggest the next point(s) very quickly, often within a few minutes[1]. More importantly, extending cost-constrained rollout to the batch setting, which is especially important in the cost-constrained setting, is nontrivial. Batch BO, at the time of this dissertation's writing, is quickly becoming a key research challenge. This is because cloud computing —and more generally, the rise of distributed computing— has made it simple to acquire a set of powerful machines to perform simultaneous function evaluations.

A large body of work has been devoted to batch BO, including (but not limited to) work by [González et al., 2016, Azimi et al., 2010, Wu and Frazier, 2016, Shah and Ghahramani, 2015, Wang et al., 2017, Snoek et al., 2012]. In general, like in sequential BO, batch BO has focused on the case where iterations have largely the same cost. The key challenge in this case, as we mentioned in Chapter 2, is that of *diversity*, in which batch candidates are sufficiently different from each other, but also sufficiently exploitative, so that sample efficiency is largely preserved. Note that we only concern ourselves with *synchronous* batches, in which parallel evaluations occur simultaneously and wait for each other to finish. This is in contrast to *asynchronous* batches, which do not wait for each other. Asynchronous batch BO is certainly very important, but somewhat beyond the scope of this chapter.

[1] In industry settings, five minutes is the maximum amount of time allotted to suggest the next evaluation. This is of course, a rule of thumb, but a widely followed one that we adhere to in this chapter

In this chapter, we tackle the problem of cost-constrained, batch BO with fixed batch size. We introduce a set of heuristics for cost-constrained BO with low overhead, the combination of which we call **C**ost **A**ppor**t**ioned **BO** (CArBO). CArBO takes cost-constrained rollout's early exploration, late exploitation strategy to the extreme. It does so by first building a cost-effective initial design which attempts to maximize coverage of the optimization domain with minimal cost, and then by using a technique called cost-cooling to encourage evaluation of cheap points before expensive ones. CArBO runs sequentially, and can also be run in batch with batch fantasizing. As we discussed in Chapter 2, Batch fantasizing extends any acquisition function to the batch setting by predicting multiple future evaluation trajectories [Wilson et al., 2018]. These *fantasies* are aggregated to identify a sequence of evaluation points, which is then evaluated in parallel. We will show that CArBO empirically scales well with batch size; increasing the batch size by a factor of b yields a b factor decrease in wall-clock time to achieve the same results. We show in an extensive set of experiments drawn from 20 real-world HPO problems that CArBO significantly outperforms competing methods within the same budget.

6.2 Batch BO

We loosely extend the cost-constrained formulation of Chapter 5 to handle the batch setting as follows:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbf{X}} \min_{\mathbf{X}\in\Omega^b} f(\mathbf{x}), \\ &\text{subject to } \sum_{i=1}^{n_{\mathbf{x}}} c(\mathbf{X}_i) \leq \tau. \end{aligned} \tag{6.1}$$

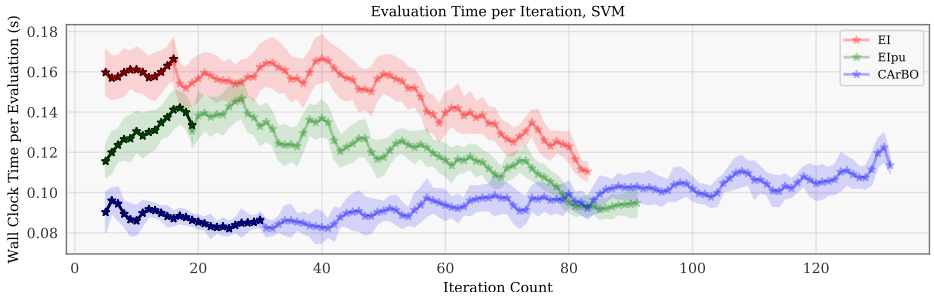


Figure 6.1: We plot the median evaluation time per iteration using each method's median number of iterations. We shade the iterations that consume the first $\tau/8$ cost, corresponding to the budget consumed by CArBO's initial design. CArBO clearly starts with many cheap evaluations and gradually evaluates more expensive points, enabling it to outperform EI and EIpu.

As with before, the above optimization problem is ill-defined and is for illustrative purposes only: we want to minimize $f(\mathbf{x})$ subject to a total cost constraint τ, where we are allowed to evaluate b points simultaneously. There is not much prior work dealing with this problem, with the exception of [Snoek et al., 2012], who uses the batch extension of EIpu (defined in Chapter 5). EIpu tends to underperform in the sequential case, and the same is true of the batch case as well, as we will show in our experiments section.

6.3 CArBO: Cost Apportioned BO

We introduce CArBO, an EI-based method employing the *early and cheap, late and expensive* strategy from the multi-fidelity and multi-task setting. This strategy is seen in Figure 6.1: in contrast with EI and EIpu, CArBO evaluates cheaper points before expensive ones. CArBO does this through a cost-effective initial design and cost-cooling. First, the cost-effective initial design aims to maximize coverage of the search space with cheap evaluations, building a good surrogate within a warm-start budget. Then, cost-cooling starts the optimization with EIpu

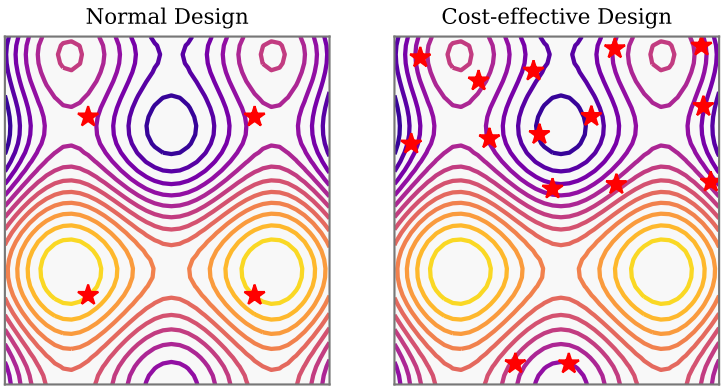


Figure 6.2: Two initial designs with the same cost, plotted over a contour of the synthetic cost function. *Left:* a grid of four points. *Right:* a cost-effective solution containing 15 points, which covers the search space better than the grid.

and ends it with EI by deprecating the cost model as iterations proceed. We discuss each of these two building blocks next.

6.3.1 Cost-effective initial design

BO is always warm-started with an *initial design*. A design is a set of points selected to learn variation in data, and BO evaluates an initial design before optimization starts to provide starting data for its GP. Initial designs consume some budget, and therefore must balance information gain with sample efficiency. An overly small design yields a poor surrogate, while an overly large design decreases sample efficiency. Initial designs must therefore be evenly spaced throughout the domain, and are often Latin hypercubes or low-discrepancy sequences [Kirk, 2012, Ryan and Morgan, 2007]. These scale better than grid points and distribute more evenly than random points [Stein, 1987, Niederreiter, 1988]. In practice, BO initial designs are small; the popular GPyOpt software uses five points [gpy, 2016], though seminal work suggests $10d$ points, where d is the

Algorithm 4 Cost-effective initial design

Input: initial budget τ_{init}, optimization domain Ω.
Cumulative time $ct = 0$, initial design $\mathbf{X}_{init} = \{\}$.
Discretize Ω into $\tilde{\Omega}$.
while $ct < \tau_{init}$ **do**
  while size $\mathbf{X}_{cand} > 1$ **do**
    exclude most expensive point from $\tilde{\Omega}$.
    exclude point closest to $\mathbf{X}_{init}$ from $\tilde{\Omega}$.
  end while
  add remaining point to $\mathbf{X}_{init}$ and evaluate.
  Update ct, cost surrogate.
end while
return $\mathbf{X}_{init}$.

problem dimension [Jones et al., 1998b].

In cost-constrained BO we aim to design a *cost-effective* initial design, which balances information gain with cost efficiency. A cost-effective design fills Ω with more evaluations than a traditional initial design within the same warm-start budget τ_{init}. We select a cost-effective initial design through the following optimization subproblem:

$$\begin{aligned} \underset{\mathbf{X} \in 2^{\Omega}}{\arg\min} \quad & \text{fill}(\mathbf{X}) := \sup_{\mathbf{x} \in \Omega} \min_{\mathbf{x}_j \in \mathbf{X}} \| \mathbf{x}_j - \mathbf{x} \|_2. \\ \text{subject to} \quad & \sum_{\mathbf{x}_i \in \mathbf{X}} c(\mathbf{x}_i) < \tau_{init}. \end{aligned} \tag{6.2}$$

Here, fill($\mathbf{X}$) is the radius of the largest empty sphere one can fit in Ω. It measures the spacing of $\mathbf{X}$ in Ω, and is known as the minimax criterion in the design-of-experiments literature [Pronzato and Müller, 2012]. The smaller a set's fill is, the better distributed it is within Ω. The argmin of Eq. (6.2) is the initial design within τ_{init} cost with the smallest fill.

Eq. (6.2) is a difficult optimization problem. In the discrete setting with constant cost, it an instance of the vertex cover problem, known to be NP-complete.

Typically, approximations to the minimax criterion are built greedily, and have a worst-case approximation factor of 2 [Damblin et al., 2013, Pronzato, 2017]. Algorithm 4 is a variation of these approaches for non-constant cost functions, and reduces to the greedy approach described in [Pronzato and Müller, 2012] given a constant cost function. Algorithm 4 first discretizes Ω into candidates $\tilde{\Omega}$ and then adds a point from $\tilde{\Omega}$ to the initial design as follows: remove the highest cost point and then the shortest distance point from $\tilde{\Omega}$, continuing until only one point remains. This remaining candidate is cheap and far from other points in the design. This inner loop is repeated, updating $c(\mathbf{x})$ every iteration until τ_{init} is exceeded. This results in a set of cheap and well-distributed points. In the batch setting, the inner loop is run b times to select b candidates that are then evaluated in parallel. Figure 6.2 shows that a cost-effective design gains far more information than a standard grid, with fifteen points compared to four.

6.3.2 Cost-cooling

The second building block is cost-cooling. Assume that at the kth BO iteration, τ_k of the total budget τ has been used (at $k = 0$, $\tau_k = \tau_{init}$). Cost-cooling, which we call EI-cool when using EI, is defined as:

$$\text{EI-cool}(\mathbf{x}) := \frac{\text{EI}(\mathbf{x})}{c(\mathbf{x})^{\alpha}}\ ,\ \alpha = (\tau - \tau_k)/(\tau - \tau_{init}). \tag{6.3}$$

Cost $c(\mathbf{x})$ is assumed to be positive and modeled with a warped GP that fits the log cost $\gamma(\mathbf{x})$. The cost is predicted by $c(\mathbf{x}) = \exp(\gamma(\mathbf{x}))$ as in the standard EIpu [Snelson et al., 2004, Snoek et al., 2012]. Learning $c(\mathbf{x})$ requires a warm-start, for which we use five points drawn from the search space uniformly at random.

As the parameter α decays from one to zero, EI-cool transitions from EIpu to

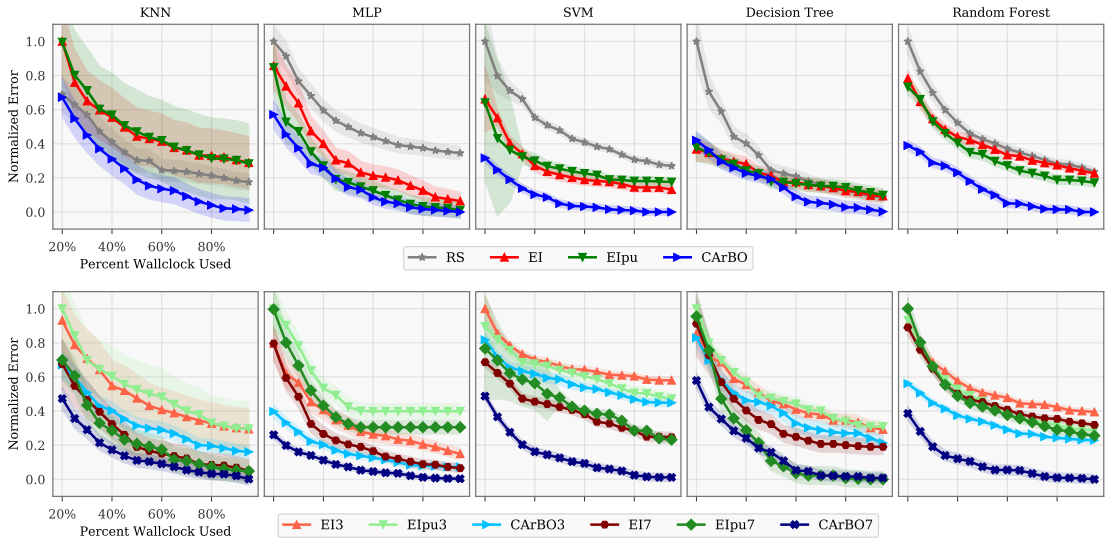


Figure 6.3: *Top:* Sequential comparison. *Bottom:* Batch comparison, of batch sizes 3 and 7. RS is plotted in grey, EI methods are plotted in red, EIpu methods are plotted in green, and CArBO methods are in plotted blue. In almost all cases, CArBO converges significantly faster than competing methods. The median is plotted, with one standard deviation shaded above and below.

EI. As a result, cost-cooling de-emphasizes the cost model as the optimization progresses and cheap evaluations are performed before expensive ones. As mentioned earlier, this behavior is shown by Figure 6.1, and also by additional benchmarks located in the appendix. The idea of cost-cooling bears connections to previous work on multi-objective, cost-preference BO [Abdolshah et al., 2019], where cost constraints are loosened to ensure that the entire Pareto frontier is explored. As we show in the appendix, EI-cool is not guaranteed to outperform both EIpu and EI but usually outperforms *at least one*.

6.3.3 CArBO

The overall method we propose is detailed in Algorithm 5, which combines the cost-effective initial design and cost-cooling. For its default cost-effective design

Algorithm 5 CArBO: Cost Apportioned BO

Input: batch b, initial budget τ_{init}, budget τ
Cumulative time $ct = 0$.
Initial budget τ_{init}.
Evaluate cost-effective initial design via Algorithm 4.
Update ct, cost and objective surrogates.
while $ct < \tau$ **do**
    $\mathbf{x}^1, \ldots, \mathbf{x}^b \leftarrow$ EI-cooling batch fantasy b as per Eq. 6.3.
    Evaluate $\mathbf{x}^1, \ldots, \mathbf{x}^b$ in parallel.
    Update ct, objective and cost surrogates.
end while
return best hyperparameter configuration observed.

budget we use $\tau_{init} = \tau/8$, where τ denotes the total BO budget. We found this value of τ_{init} to work well in experiments, and investigate different choices in Section 6.5.

We formulate CArBO in the general batch setting, namely with $b \geq 1$. Performing evaluations in parallel further reduces wall clock time, and can be achieved with standard techniques as described with pseudo-code in the appendix. Note that in the cost-constrained setting, linear scaling means convergence to the same optimum as sequential BO in $1/b$ of the wall clock time. CArBO achieves linear scaling by building its cost-efficient initial design with batches of points that are far apart from each other and then by batch-fantasizing cost-cooling. We will demonstrate this scaling on relatively large batch sizes of up to 16 in Section 6.5.

6.4 Experiments

We evaluate the performance of CArBO on a varied set of five popular HPO problems, each trained on four different datasets, yielding twenty total benchmarks. Each benchmark is given its own wall clock budget. Each HPO problem

Objective	Budget	EI3	EIpu3	CArBO3
KNN a1a	150 (s)	0.133 (83)	0.135 (121)	0.133 (111)
KNN a3a	300 (s)	0.121 (90)	0.121 (116)	0.119 (147)
KNN splice	10 (s)	0.123 (143)	0.120 (183)	0.113 (161)
KNN w2a	400 (s)	0.055 (83)	0.056 (142)	0.048 (77)
MLP a1a	100 (s)	0.123 (50)	0.128 (34)	0.121 (119)
MLP a3a	160 (s)	0.108 (40)	0.110 (30)	0.107 (97)
MLP splice	50 (s)	0.051 (41)	0.054 (32)	0.038 (71)
MLP w2a	200 (s)	0.024 (33)	0.024 (27)	0.023 (73)
SVM a1a	20 (s)	0.120 (189)	0.120 (218)	0.120 (295)
SVM a3a	30 (s)	0.109 (197)	0.108 (256)	0.107 (343)
SVM splice	4 (s)	0.114 (100)	0.114 (127)	0.113 (225)
SVM w2a	90 (s)	0.023 (256)	0.022 (304)	0.021 (356)
DT a1a	2.5 (s)	0.135 (150)	0.135 (149)	0.135 (150)
DT a3a	2.5 (s)	0.132 (133)	0.132 (135)	0.131 (134)
DT splice	2 (s)	0.029 (300)	0.029 (300)	0.029 (332)
DT w2a	8 (s)	0.055 (77)	0.052 (80)	0.054 (78)
RF a1a	30 (s)	0.117 (68)	0.116 (133)	0.116 (160)
RF a3a	35 (s)	0.110 (80)	0.109 (118)	0.109 (143)
RF splice	10 (s)	0.015 (31)	0.015 (55)	0.014 (46)
RF w2a	80 (s)	0.049 (60)	0.045 (135)	0.044 (142)

Table 6.1: Results for all different batch methods on five HPO tasks, each tested on four datasets using 51 replications. The tasks are K-nearest-neighbors (KNN), multi-layer perceptron (MLP), support-vector machine (SVM), decision tree (DT), and random forest (RF). The datasets are a1a, a3a, splice, w2a. The median classification error is shown for different optimizers and batch sizes. CArBO3 displays strong results, showing the best on 18 out of the 20 benchmarks and lagging by a small amount in the other four cases.

is a model in scikit-learn [Pedregosa et al., 2011]. We train on four classification datasets: *splice*, *a1a*, *a3a*, and *w2a*. The splice dataset (training size: 1000, testing size: 2175) classifies splice junctions in a DNA sequence. The a1a and a3a datasets (training sizes: 1605, 2265, testing sizes: 30,956, 30,296) predict whether the annual income of a family exceeds 50,000 dollars based on 1994 US census data. The w2a dataset (training size: 3,470, testing size: 46,279) predicts the category of a webpage. All datasets are available in the UCI machine learning repository [Dua and Graff, 2017]. Each benchmark is replicated 51 times on independent AWS m4.xlarge machines to ensure consistent evaluation times. The

Objective	Budget	EI7	EIpu7	CArBO7
KNN a1a	150 (s)	0.128 (149)	0.128 (195)	0.128 (250)
KNN a3a	300 (s)	0.117 (184)	0.117 (217)	0.116 (354)
KNN splice	10 (s)	0.107 (275)	0.107 (361)	0.103 (353)
KNN w2a	400 (s)	0.052 (150)	0.049 (277)	0.046 (189)
MLP a1a	100 (s)	0.122 (96)	0.127 (72)	0.119 (227)
MLP a3a	160 (s)	0.108 (79)	0.108 (62)	0.106 (194)
MLP splice	50 (s)	0.043 (84)	0.052 (64)	0.037 (145)
MLP w2a	200 (s)	0.023 (69)	0.023 (57)	0.023 (152)
SVM a1a	20 (s)	0.120 (395)	0.120 (483)	0.119 (663)
SVM a3a	30 (s)	0.108 (418)	0.107 (572)	0.107 (722)
SVM splice	4 (s)	0.114 (191)	0.113 (307)	0.111 (540)
SVM w2a	90 (s)	0.022 (570)	0.021 (676)	0.021 (763)
DT a1a	2.5 (s)	0.132 (347)	0.132 (347)	0.132 (344)
DT a3a	2.5 (s)	0.130 (300)	0.129 (300)	0.130 (304)
DT splice	2 (s)	0.028 (645)	0.025 (655)	0.027 (664)
DT w2a	8 (s)	0.077 (177)	0.078 (181)	0.054 (173)
RF a1a	30 (s)	0.116 (137)	0.115 (270)	0.114 (272)
RF a3a	35 (s)	0.108 (170)	0.109 (243)	0.108 (252)
RF splice	10 (s)	0.013 (73)	0.013 (114)	0.013 (88)
RF w2a	80 (s)	0.053 (258)	0.053 (312)	0.042 (298)

Table 6.2: Results for batch size 7 on KNN, MLP, SVM, decision tree, and random forest (RF). CArBO7 displays strong results, showing the best on 18 out of the 20 benchmarks and lagging by a small amount in the other four cases.

problems and search spaces follow, with unlisted hyperparameters being set to the scikit-learn default.

K-nearest-neighbors (KNN). We consider a $5d$ search space: dimensionality reduction percentage and type in [1e-6, 1.0] log-scaled and {Gaussian, Random}, respectively, neighbor count in {1, 2, ..., 256}, weight function in {Uniform, Distance}, and distance in {Minkowski, Cityblock, Cosine, Euclidean, L1, L2}.

Multi-layer perceptron (MLP). We consider a $11d$ search space: number of layers in {1, 2, 3, 4}, layer sizes in {10, 11, ..., 150 } log-scaled, activation in {Logistic, Tanh, ReLU}, tolerance in [1e-5, 1e-2] log-scaled, and Adam parameters [Kingma and Ba, 2014]: step size in [1e-6, 1.0] log-scaled, initial step size in [1e-6,

Objective	Budget	EI11	EIpu11	CArBO11
KNN a1a	150 (s)	0.126 (238)	0.126 (318)	0.128 (411)
KNN a3a	300 (s)	0.115 (283)	0.115 (331)	0.115 (622)
KNN splice	10 (s)	0.099 (411)	0.102 (536)	0.095 (537)
KNN w2a	400 (s)	0.047 (206)	0.048 (373)	0.044 (314)
MLP a1a	100 (s)	0.122 (133)	0.126 (103)	0.119 (344)
MLP a3a	160 (s)	0.107 (114)	0.108 (90)	0.106 (296)
MLP splice	50 (s)	0.041 (126)	0.050 (92)	0.036 (215)
MLP w2a	200 (s)	0.022 (101)	0.022 (84)	0.023 (226)
SVM a1a	20 (s)	0.120 (587)	0.120 (753)	0.119 (956)
SVM a3a	30 (s)	0.108 (611)	0.107 (913)	0.106 (1019)
SVM splice	4 (s)	0.113 (282)	0.113 (425)	0.111 (836)
SVM w2a	90 (s)	0.022 (855)	0.021 (1040)	0.020 (1034)
DT a1a	2.5 (s)	0.132 (541)	0.132 (537)	0.132 (540)
DT a3a	2.5 (s)	0.129 (473)	0.130 (464)	0.128 (476)
DT splice	2 (s)	0.026 (1032)	0.027 (979)	0.025 (985)
DT w2a	8 (s)	0.078 (277)	0.078 (279)	0.052 (272)
RF a1a	30 (s)	0.116 (214)	0.114 (373)	0.114 (359)
RF a3a	35 (s)	0.108 (248)	0.108 (337)	0.108 (355)
RF splice	10 (s)	0.013 (110)	0.013 (162)	0.012 (118)
RF w2a	80 (s)	0.051 (389)	0.051 (484)	0.041 (383)

Table 6.3: Results for batch size 11 on KNN, MLP, SVM, decision tree, and random forest (RF). CArBO7 displays strong results, showing the best on 18 out of the 20 benchmarks and lagging by a small amount in the other four cases.

1e-2] log-scaled, beta1 and beta2 in [1e-3, 0.99] log-scaled.

Support Vector Machine (SVM). We consider a $6d$ search space: iteration count in {1, 2, ..., 128}, penalty term in {L1, L2, ElasticNet}, penalty ratio in [0, 1], step size in [1e-3, 1e3] log-scaled, initial step size in [1e-4, 1e-1] log-scaled, optimizer in {Constant, Optimal, Invscaling, Adaptive }.

Decision tree (DT). We consider a $3d$ search space: tree depth in {1, 2, ..., 64}, tree split threshold in [0.1, 1.0] log-scaled, and split feature size in [1e-3, 0.5] log-scaled.

Random forest (RF). We consider a $3d$ search space: number of trees in {1, 2, ..., 256}, tree depth in {1, 2, ..., 64}, and tree split threshold in [0.1, 1.0] log-scaled.

Objective	CArBO	CArBO3	CArBO7	CArBO11
KNN a1a	60%	49%	-10%	-11%
KNN a3a	52%	58%	22%	28%
KNN splice	73%	75%	52%	49%
KNN w2a	59%	55%	60%	59%
MLP a1a	21%	69%	67%	69%
MLP a3a	-9%	50%	61%	56%
MLP splice	34%	62%	66%	59%
MLP w2a	4%	27%	20%	-7%
SVM a1a	22%	42%	53%	39%
SVM a3a	67%	66%	65%	52%
SVM splice	-1%	50%	67%	67%
SVM w2a	74%	78%	22%	72%
DT a1a	-2%	-7%	17%	-8%
DT a3a	15%	22%	-22%	35%
DT splice	10%	2%	-25%	2%
DT w2a	- 18%	-41%	95%	96%
RF a1a	44%	28%	63%	61%
RF a3a	40%	54%	49%	-24%
RF splice	16%	33%	27%	33%
RF w2a	52%	48%	82%	84%
Net Saving	**32.5**%	**45.1**%	**41.6**%	**40.6**%

Table 6.4: For each batch size and objective, we calculate the median cost savings as a percentage of budget. Negative numbers indicate that CArBO performed worse than the best optimizer. CArBO performs strongly on the large majority of problems. Furthermore, when it does worse, it only does worse by a small amount.

Our code is built on GPyOpt [gpy, 2016]. Kernel hyperparameters for both the objective and cost Gaussian process models are calculated via maximum marginal likelihood estimation [Rasmussen and Williams, 2006]. We compare CArBO to EI, EIpu, and random search in the sequential case, as well as batch sizes three, seven, and eleven. Note that multi-fidelity methods such as Hyperband are inapplicable on these benchmarks as they do not have any fidelity parameters (with the exception of SVM).

We compare the performance of the competing algorithms in three ways. First, we plot performance for each HPO problem by averaging the classification

error for each model over the four datasets used (Figure 6.3). This is done to condense the large number of benchmarks we ran. We average as follows: first we normalize performance so that the worst optimizer starts optimization at 1.0 and the best optimizer ends at 0.0, then we take the mean over all datasets. We plot sequential results in the first row and batch results in the second. CArBO outperforms both EI and EIpu by a large margin across all batch sizes.

Second, we compile a table of classification errors and iterations taken, and bold the optimizer with the lowest classification error. For space's sake, we truncate the classification error precision to three digits. The table also lists each benchmark's time budget in seconds. CArBO for batch sizes one, three, seven, and eleven is best on 16, 18, 17, and 16 HPO problems, respectively. As expected, CArBO is able to exploit more BO iterations than either EI or EIpu for the same wall-clock time.

Third, we calculate CArBO's total cost savings, defined as the time needed by CArBO to achieve comparable results to the next best optimizer (Table 6.4). We consider Table 6.4 the most instructive comparison because it provides quantitative savings instead of a qualitative ranking. We list the median cost savings for each benchmark, as well as net savings over all benchmarks, for each batch size. CArBO achieves large cost savings of roughly 40 percent, averaged over all benchmarks and batch sizes.

6.5 Additional Experiments

This section illustrates the empirical behavior of CArBO relative to its internal design choices, such as batch size or initial design budget. First, we investigate

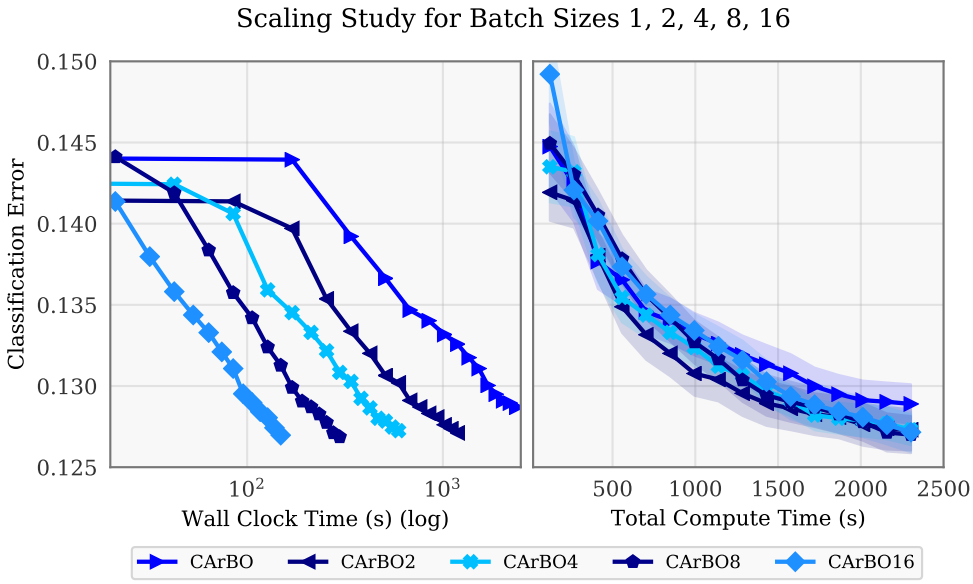


Figure 6.4: We compare CArBO's wall clock time performance (*left*) to its total compute time performance (*right*) for batch sizes 1, 2, 4, 8, and 16. CArBO scales linearly with batch, evidenced by comparable total compute time performance among all batch sizes.

the sensitivity of CArBO to its initial design budget. We also run a scaling test for batch sizes up to 16. Finally, we run an ablation study. All following experiments use the MLP a1a benchmark.

Batch Scaling. Information is used less optimally in batch BO than in sequential BO. Large batches size may result in decreased cost efficiency. We examine this potential risk by running CArBO for batch size 1, 2, 4, 8, and 16 with wall clock time budgets of 2400, 1200, 600, 300, and 150 seconds, respectively. Each batch thus is allocated the same total compute time. As seen in Figure 6.4, moving up to batch size 16 results in little to no performance loss, indicating that CArBO scales linearly with batch size.

Initial Design. In BO methods, the size of the initial design is somewhat arbitrary. This is also true for CArBO, which uses 1/8 of the budget for the

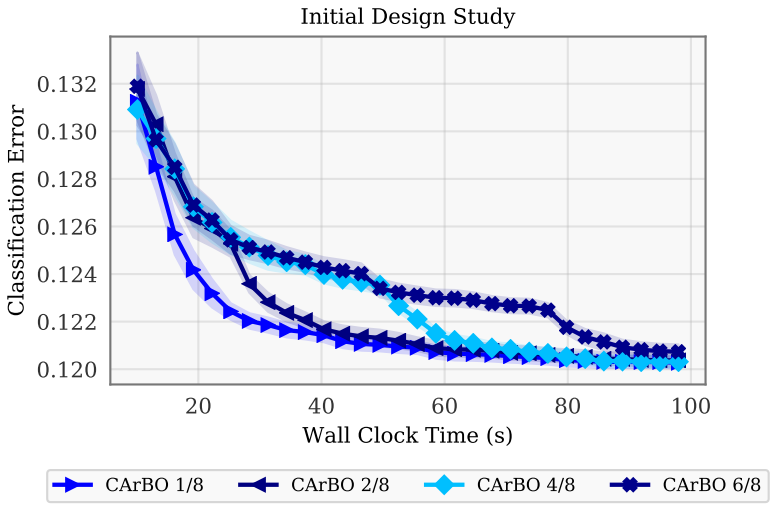


Figure 6.5: We study CArBO's initial design budget from 1/8 to 6/8 of the total budget. While CArBO 6/8 does perform worse, there is relatively little performance change, indicating at least some robustness to the initial design budget.

initial design. We investigate the impact of varying the initial design budget in Figure 6.5 from 1/8 up to an extreme value of 6/8 of the total budget. CArBO's performance was relatively unchanged; using 6/8 of the total budget for the initial design degraded performance slightly, but represents an extreme case. We leave a systematic approach to select the initial design budget as future work.

Ablation Study. CArBO combines two components: a cost-effective design and cost-cooling. A natural question is the performance contribution of each. To answer this question we perform an ablation study, in which we remove each component and re-run optimization. In Figure 6.6, we compare CArBO to CArBO using just EI or EIpu. We also compare these to EI and EIpu. The initial design contributes the larger performance increase, which is not surprising. At the same time, CArBO using EI-cooling performs the best.

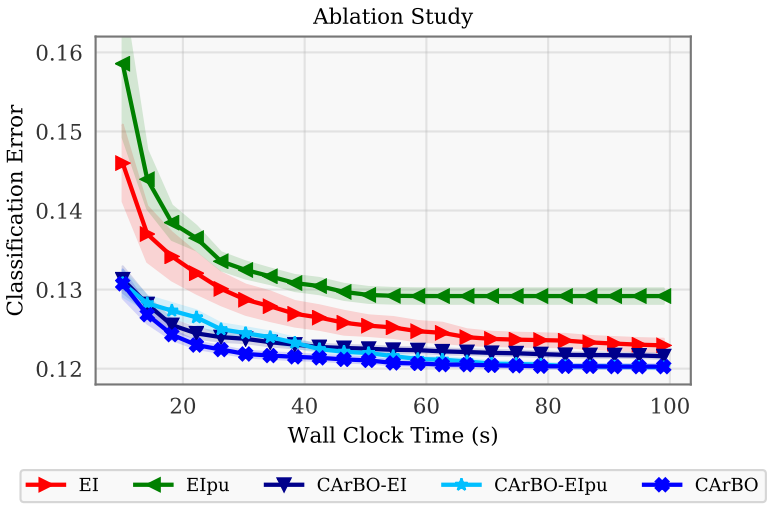


Figure 6.6: The cost-effective design contributes the larger performance increase compared to EI-cooling in this ablation study.

6.6 Building better cost models

One of the most popular applications of Bayesian optimization is neural network hyperparameter optimization [Snoek et al., 2015, Frazier, 2018a]. In this setting, BO is used to determine the optimal neural network hyperparameters that minimize test error. Typical hyperparameters for a neural network include layer count, layer sizes, types of layers, and loss function, as well as additional, architecture-dependent hyperparameters such as convolution filter sizes. Cost is certainly an important consideration in neural network training, as various hyperparameters such as layer size may drastically change the cost of neural network training. To keep cost into account, EIpu with a GP cost model is typically used as the acquisition function [Snoek et al., 2012].

Predicting the cost of a computer program is well-studied by prior work [Huang et al., 2010, Hutter et al., 2014, Di et al., 2013, Yang et al., 2018, Priya et al., 2011], in which a cost model forecasts system loads, dispatches com-

putational resources, or determines computational feasibility. Gaussian processes extrapolate poorly, leading to high-variance cost predictions far away from data. This high-variance may introduce extra error that decreases BO performance.

We show that cost-aware BO can benefit from specialized, low-variance cost models that extrapolate well. Floating point operations (flops) are a standard measure of a computer program's cost [Peise and Bientinesi, 2012]. We consider a linear model that uses a small feature set, where each feature counts the flops of a subroutine in the program. The total runtime is modeled as a linear combination of these features. We train the model through robust regression to deal with outliers in timing data [Boyd and Vandenberghe, 2004]. We model multi-layer perceptrons (MLPs) and convolutional neural network (CNNs) with low-variance cost models and show that they tend to improve BO performance.

6.6.1 Neural networks

The most basic neural network is the multi-layer perceptron (MLP). The MLP consists of multiple fully-connected layers, each with its own activation function. Each layer is a dense matrix $n \times m$ of floating points numbers. Common activation functions include the sigmoid, arctan, softmax, rectified-linear (ReLU) [LeCun et al., 2015, Goodfellow et al., 2016]. The cost of applying each layer to the input vector is the cost of a matrix-vector multiplication followed by an application of the activation function, which has floating point cost $O(nm)$. Typically, MLPs are used for generic non-linear classification and regression problems.

For more complex vision tasks such as image recognition or image segmentation, a convolutional neural network (CNN) is used. A CNN consists

of a large number of so-called convolutional layers appended over an MLP. [Alom et al., 2018]. Each layer has a kernel size r and channel width m, and will perform 3-dimensional convolutions of the input data according to the kernel size and channel width. Each layer also has an optional pooling layer with pooling ratio p_i, which compresses the output by a scalar factor p_i. Given input size I, The cost of applying the convolutional layer i is $O(I^2 r^2 m_i m_{i+1})$ and the cost of pooling is $O(I^2 p_i m_i)$.

Neural network training involves minimizing an error function with respect to the neural network layer weights using a fixed number of gradient iterations. Typically, training sets are very large, and so a full gradient is not calculated. Instead, a smaller subset of the training data, known as the batch, of size b is used to calculate a *stochastic* gradient. To further decrease training time, only a randomly chosen fraction of weights is trained at a time. This fraction is determined through a dropout rate d.

6.6.2 Robust regression

A linear regression problem is succinctly expressed as:

$$\min_x ||Ax - b||_2.$$

Alternatively, linear regression seeks to minimize the sum of squares of errors ϵ between the model and training data. The literature on robust regression has long observed that the $\mathcal{L}_2$ loss (the two-norm), tends to weigh outliers far away from data more than necessary; if an error is large, it gets squared and thus shifts the fit more than one might deem necessary. Robust regression weighs outliers in the data less than the $\mathcal{L}_2$ loss through a robust loss function by solving the

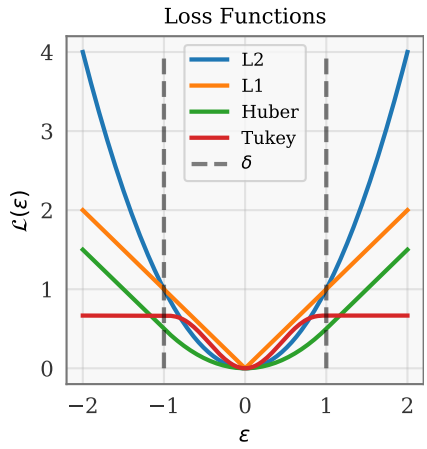


Figure 6.7: A comparison of different loss functions. The standard $\mathcal{L}_2$ loss weighs outliers, defined as points outside the interval $[-\delta, \delta]$, quadratically. Robust loss functions such as the $\mathcal{L}_1$ or the Huber weigh outliers linearly. The Tukey biweight loss weights outliers by a constant amount.

following minimization problem;

$$\min_x \|Ax - b\|_{\mathcal{L}}.$$

There are many different loss functions, and we list the common ones associated with robust regression below.

- The $\mathcal{L}_1$ loss $\mathcal{L}_1(\epsilon)$ weighs errors ϵ according to their absolute value, and is favored in compressed sensing regimes because of its sparsity-inducing properties [Boyd and Vandenberghe, 2004]:

$$\mathcal{L}_1(\epsilon) = \|\epsilon\|_1.$$

- The Huber loss $\mathcal{L}_{huber}(\epsilon)$ may be seen as a compromise between the $\mathcal{L}_1$ and $\mathcal{L}_2$. It weighs outliers, defined as errors outside some region $[-\delta, \delta]$, linearly, and points within the region quadratically.

$$\mathcal{L}_{huber}(\epsilon) = \begin{cases} \frac{1}{2}\epsilon^2 \ , & |\epsilon| < \delta \\ \delta|\epsilon| - \frac{1}{2}\delta^2 \ , & |\epsilon| \geq \delta, \end{cases}$$

- The biweight loss $\mathcal{L}_{tukey}(\epsilon)$, also known as Tukey's biweight, weighs each outlier by a constant amount. Unfortunately, unlike the $\mathcal{L}_1$ and Huber, Tukey's biweight is no longer convex.

$$\mathcal{L}_{tukey}(\epsilon) = \begin{cases} \frac{\delta^2}{6}\left[1 - \left(1 - \frac{\epsilon^2}{\delta^2}\right)^3\right], & |\epsilon| < \delta \\ \frac{\delta^2}{6}, & |\epsilon| \geq \delta, \end{cases}$$

For both the Huber and Tukey's biweight loss functions, the parameter δ must typically be estimated, often through some measure of standard deviation [Boyd and Vandenberghe, 2004]. We use the Huber loss for our experiments because we typically do not expect sparsity in our error, and it is one of the simpler robust regression loss functions to deal with due to its convexity.

6.6.3 Cost models for multi-layer perceptrons

Consider an MLP with layer sizes $n_1, n_2, \ldots, n_k$. We define the following features: the cost of all matrix multiplications $\phi_{quad} = (n_1 n_2 + n_2 n_3 + \cdots + n_{k-1} n_k)$ and the cost of batch normalization and activation functions $\phi_{linear} = (n_1 + n_2 + \cdots + n_k)$. Batch size b and epoch e are constants, and are omitted. Our cost model is:

$$c(x) = c_1 \phi_{quad} + c_2 \phi_{linear} + c_3.$$

6.6.4 Experiments

We let $k = 4$, $10 \leq n_i \leq 300$, $b = 100$, $e = 200$, and use no dropout. We assume we are training our MLP on four classification datasets: *splice*, *a1a*, *a3a*, and *w2a*. The splice dataset (training size: 1000, testing size: 2175) classifies splice junctions in

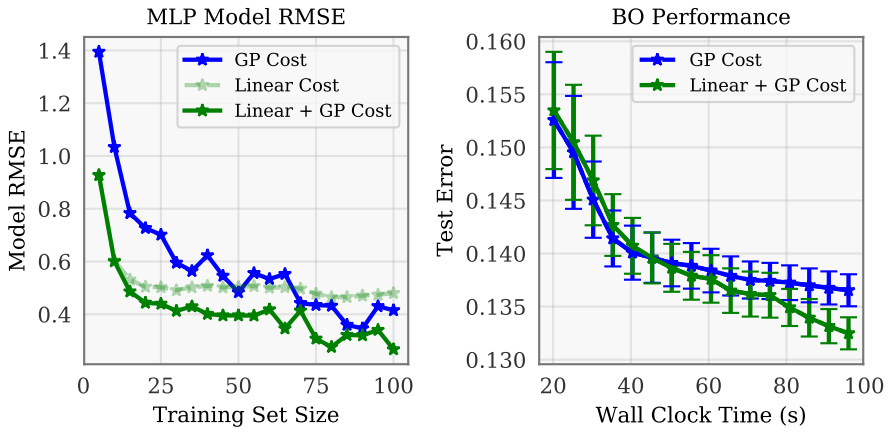


Figure 6.8: We run EIpu using both low-variance and warped GP models on MLP a1a. The warped GP (blue) has higher prediction error and slower performance than the low-variance model (green).

Objective	Warped GP	Linear Model	Net Saving
MLP a1a	0.1363	**0.1276**	42%
MLP a3a	0.1124	**0.1115**	12%
MLP splice	0.0686	**0.0629**	25%
MLP w2a	0.0257	**0.0242**	29%
Net Saving	-	-	**27%**

Table 6.5: EIpu results with warped GP and low-variance models.

a DNA sequence. The a1a and a3a datasets (training sizes: 1605, 2265, testing sizes: 30,956, 30,296) predict whether the annual income of a family exceeds 50,000 dollars based on 1994 US census data. The w2a dataset (training size: 3,470, testing size: 46,279) predicts the category of a webpage.

We train our low-variance model on timing data consistent with EIpu's evaluations, and run EIpu with this model on all four MLP benchmarks 51 times each. Using our linear cost model clearly improves BO performance. Median classification error and cost savings are shown in Table 6.5.

The left plot of Figure 6.8 compares root-mean-squared error (RMSE) among three cost models: a warped GP (blue), our low-variance model (dotted green),

and GP whose mean is our low-variance model (green), on 10,000 randomly chosen hyperparameters. A GP with a low-variance linear mean is the most accurate, while the warped GP is least accurate. Our low-variance models are strongest in the limited data regime; as the training set grows, the error gap shrinks. The right plot of Figure 6.8 shows significant improvement over the default cost model on the MLP a1a benchmark, which is the best performing benchmark out of the four we ran.

6.6.5 Cost models for convolutional neural networks

Consider a CNN with h convolutional layers of kernel size r, channel sizes $m_1, \ldots, m_k$, pooling ratios $p_1, \ldots, p_k$, a fixed activation function, and an input of size $I \times I \times c$, where c is the number of color channels. We define the additional features: the cost of convolutions $\phi_{conv} = I^2 r^2 c m_1 + \sum_{i=1}^{k-1} I^2 r^2 m_i m_{i+1}$ and the cost of pooling $\phi_{pool} = \sum_{i=1}^{k} I^2 p_i m_i$. After the convolutional layers is an MLP of input size $I^2 p_k m_k$, whose features we also include. Our final cost model is thus:

$$c(x) = c_1 \phi_{conv} + c_2 \phi_{pool} + c_3 \phi_{quad} + c_4 \phi_{linear} + c_5.$$

6.6.6 Experiments

We let $h = 4$, $1 \leq r \leq 10$, $8 \leq m_i \leq 128$, $k = 2$, $8 \leq n_i \leq 128$, $b = 100$, $4 \leq e \leq 20$, and we add a single max pooling layer after the convolutional layers and a dropout rate of 0.25. We train on MNIST [LeCun et al., 1998, Deng, 2012], a standard handwriting recognition dataset, and compare accuracy between two cost models: a warped GP and a GP whose mean is the low-variance

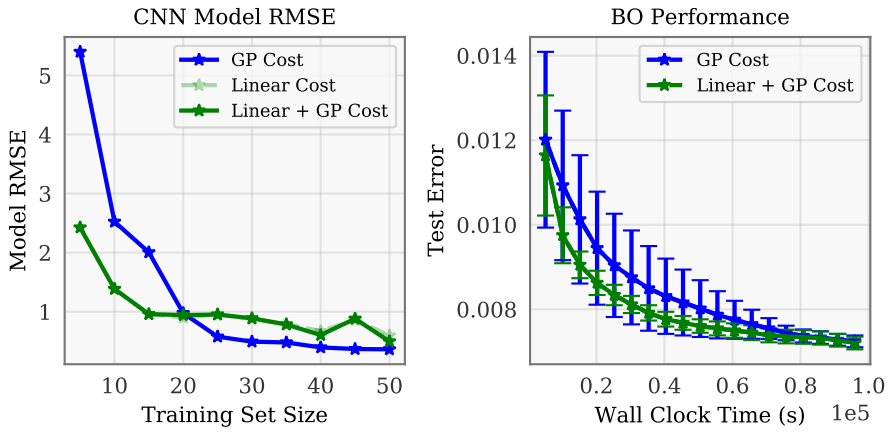


Figure 6.9: The low-variance CNN model had lower RMSE only in the limited data regime (iterations < 20). Though it converges faster than the warped GP, both converge to the same optimum.

model. We train on timings consistent with EIpu evaluation points and compare classification error using 10,000 random hyperparameter configurations.

CNNs proved more difficult to model than MLPs. Figure 6.9 shows that the low-variance model is only better than the GP for small training sets of size less than 20. This is likely because flops do not reflect the actual runtime of CNN training, which uses highly optimized libraries to perform convolutions.

6.7 Conclusion

How to best use a cost model to plan optimization when evaluation cost varies is a challenging question. EIpu, which normalizes the acquisition function by the cost model, is reasonable insofar as having the correct units, but performs poorly if the optimum is not cheap. We introduced CArBO, an algorithm adapting the *early and cheap, late and expensive* strategy from prior work on grey-box, cost-constrained BO to the black-box setting, in which no exter-

nal information about cost is given. By combining a cost-effective initial design and cost-cooling, CArBO was shown to outperform EI and EIpu on an extensive set of real-world benchmarks, both in the sequential and batch setting. 6 A number of directions for future work are open. Adapting CArBO's initial design and cost-cooling to other acquisition functions, such as predictive entropy search [Hernández-Lobato et al., 2014] or max-value entropy search [Wang and Jegelka, 2017], is straightforward. Combining CArBO with multi-fidelity to learn fidelity parameters and their relationship to cost is also of interest. As we showed, building an accurate cost model is an important problem, and our flop-counting approach can certainly be built on. Finally, CArBO assumes fixed batch size, and allowing it to vary may boost performance further. This is likely of practical importance as BO is often run on clusters or cloud services.

CHAPTER 7

BAYESIAN OPTIMIZATION WITH GRADIENTS

7.1 Introduction

In this chapter, we discuss scaling GPs with derivative information. For many simulation models, derivatives may be computed at little extra cost via finite differences, complex step approximations, adjoint methods, or algorithmic differentiation [Forrester et al., 2008].

As discussed in Chapter 1, the kernel matrix in this setting is of size $nd \times nd$. This makes regression expensive, due to the $O(n^3d^3)$ computation required. We extend prior work in scaling GP regression [Dong et al., 2017, Gardner et al., 2018, Wilson et al., 2015] by developing fast matrix-vector multiplications in the GP setting with derivatives. We demonstrate our work on applications in Bayesian Optimization (BO) [Wu et al., 2017], implicit surface reconstruction [Macedo et al., 2011], and terrain reconstruction.

7.2 Background

Many scalable approximation methods for GP regression have been proposed, including [Smola and Bartlett, 2001, MacKay and Gibbs, 1997, Harbrecht et al., 2012, Gardner et al., 2018, Wilson et al., 2015, Dong et al., 2017]. All decrease the cost of the GP log-marginal likelihood (LML), which we discussed in Chapter 1. We reiterate the LML below:

$$\mathcal{LML}(y_X \mid \theta) = -\frac{1}{2}\left[(y_X - \mu_X)^T \alpha + \log|\tilde{K}_{XX}| + n\log 2\pi\right].$$

$$\frac{\partial}{\partial \theta_j}\mathcal{LML}(y_X \mid \theta) = -\frac{1}{2}\mathrm{tr}\left((\alpha\alpha^T - \tilde{K}_{XX}^{-1})\frac{\partial \tilde{K}_{XX}}{\partial \theta_j}\right).$$

Recall that the LML's dominant costs are the solve with $\tilde{K}_{XX}$ and the computation of its log-determinant $\log|\tilde{K}_{XX}|$. Typically, we first compute the Cholesky factorization $\tilde{K}_{XX} = LL^T$, which costs $O(n^3)$ flops. Once the Cholesky is computed, computing the log-marginal likelihood and its gradient costs $O(n^2)$ flops. For large systems, the cubic complexity of the Cholesky factorization is restrictive. A general framework for more efficient LML computation uses congjugate gradient (CG) for the solve and stochastic trace estimation for the log-determinant. We detail these in a later subsection.

Scalable methods incorporating derivatives have received little attention. The unfavorable spectrum of K_{XX}^{∇} implies standard kernel matrix low-rank approximations such as subset of regressors (SoR) [Rasmussen and Williams, 2006] and fully independent training conditional (FITC) are infeasible [Snelson and Ghahramani, 2005]. In this chapter, we develop scalable methods for GPs with derivatives by extending two existing scalable GP methods, structured kernel interpolation (SKI) [Wilson and Nickisch, 2015] and structured kernel interpolation for products (SKIP) to handle K_{XX}^{∇}, the kernel matrix with derivative information. SKI uses local interpolation to map scattered data onto a large grid of inducing points, enabling fast MVMs using FFTs. SKIP approximates a high-dimensional product kernel as a Hadamard product of low rank Lanczos decompositions [Gardner et al., 2018]. Both SKI and SKIP provide fast approximate kernel MVMs, which are a building block to solve linear systems with the kernel matrix and to approximate log determinants [Dong et al., 2017].

The eigenspectrum of K_{XX}^{∇} may exhibit slow decay, despite K_{XX} itself possessing fast spectral decay. Fast forward to Figure 7.1 for two examples. We show

that a pivoted Cholesky preconditioner significantly improvs convergence.

7.2.1 Conjugate Gradient and preconditioning

Conjugate gradient (CG) is a classic method to solve the positive definite system $Ax = b$, and only requires a way to multiply a vector by A [Saad, 1992]. If the matrix-vector multiplication (MVM) is quicker than the dense $O(n^2)$, CG is usually favored over a direct solve. The convergence of CG depends heavily on clustering of the eigenvalues of A [Trefethen and Bau III, 1997]. A popular way to increase convergence is with a preconditioner P, and solving instead for $P^{-1}Ax = P^{-1}b$. If $P^{-1}A$ exhibits favorable spectral clustering or decay, CG will often converge much faster in practice. Choosing a suitable preconditioner is problem-specific.

In the context of scalable GPs, CG is used to solve $\widetilde{K}_{XX}\lambda = (y_X - \mu_X)$. We discuss our pivoted Cholesky preconditioner in later sections.

7.2.2 Stochastic trace estimation

Stochastic trace estimation [Dong et al., 2017] is used to calculate the log-determinant $\log|\widetilde{K}_{XX}| = tr(\log \widetilde{K}_{XX})$, and like many other iterative methods in numerical linear algebra, only requires an MVM. The motivation behind stochastic trace estimation is the following identity:

$$\log|\widetilde{K}_{XX}| = tr(\log \widetilde{K}_{XX}) = \mathbb{E}[z^T(\log \widetilde{K}_{XX})z],$$

where z a vector whose components are independent with mean zero and variance one. We can calculate the expectation via Monte Carlo using p probe vectors $z_1, z_2, \ldots, z_p$, leading to the following approximation:

$$\mathbb{E}[z^T(\log \widetilde{K}_{XX})z] \approx \frac{1}{k}\sum_{i=1}^{k} z_i^T(\log \widetilde{K}_{XX})z_i,$$

where our probe vectors are drawn to have independent standard normal components. If we first apply the Lanczos decomposition $\widetilde{K}_{XX} = QTQ^T$, which only requires an MVM w/ $\widetilde{K}_{XX}$, we obtain the following:

$$\widetilde{K}_{XX} = QTQ^T \implies \log \widetilde{K}_{XX} = Q\log(T)Q^T,$$

$$\mathbb{E}[z^T(\log \widetilde{K}_{XX})z] \approx \sum_{i=1}^{k}(Qz_i)^T(\log T)(Qz_i).$$

In practice, we truncate the Lanczos decomposition early and set $p << n$. This makes stochastic trace estimation far faster than the Cholesky approach.

7.2.3 Structured kernel interpolation

If a GP's kernel is stationary and its data locations are on a uniform grid, then K_{XX} has structure such that it can be multiplied quickly with fast Fourier transforms (FFTs) in $O(n \log n)$ time. If the kernel is a product of 1D kernels i.e., $k(\mathbf{x}, \mathbf{x}') = \Pi_{i=1}^{d} k(\mathbf{x}^{(i)}, \mathbf{x}'^{(i)})$, then K is a Kronecker product of 1D kernel matrices:

$$K_{XX} = K_1 \otimes K_2 \otimes, \ldots, \otimes K_d.$$

In this case, K_{XX} can be applied more quickly by combining the identity $(K_1 \otimes K_2)\text{vec}(X) = \text{vec}(K_1 X K_2)$ with FFTs.

These structured kernel MVMs are inapplicable when the data locations are non-uniform, and SKI interpolates their kernel values with kernel values on a

pre-specified grid. More formally, we apply the interpolation scheme:

$$k(\mathbf{x}, \mathbf{x}') \approx \sum_i w_i(\mathbf{x}) k(\mathbf{x}_i, \mathbf{x}'),$$

where each $\mathbf{x}_i$ is a point on a grid of size $q = m^d$ and $w_i(\mathbf{x})$ is an appropriately chosen weight. Written in matrix form, SKI is expressed as:

$$K_{XX} \approx W K_{UU} W^T,$$

where $K_{UU} \in \mathbb{R}^{q \times q}$ and $W \in \mathbb{R}^{n \times q}$. SKI uses local interpolation scheme such as cubic convolutional interpolation [Keys, 1981], which results in a sparse W with 4^d entries per row. Therefore, the total cost of an MVM with $W K_{UU} W^T$ is $O(n4^d + q \log q)$, which is far cheaper in low dimensions than $O(n^2)$.

7.2.4 Structured kernel interpolation for products

Structured kernel interpolation for products (SKIP) avoids the exponential scaling in dimension of SKI, and is thus suited better for higher-dimensional problems. Let $A \odot B$ denote the Hadamard product (i.e., element-wise product) of matrices A and B with the same dimensions. SKIP decomposes a product kernel into a Hadamard product of one-dimensional kernels matrices, and performs a SKI approximation along each dimension. Each SKI MVM is then used to construct a rank-p, truncated Lanczos decomposition. This gives the following sequence of approximations:

$$K_{XX} \approx (W_1 K_1 W_1^T) \odot, \ldots, \odot (W_d K_d W_d^T) \approx (Q_1 T_1 Q_1^T) \odot, \ldots, \odot (Q_d T_d Q_d^T).$$

Note that $A \odot B$ can applied to a vector with the following identity:

$$(A \odot B)\, \mathbf{x} = \operatorname{diag}(A D_{\mathbf{x}} B),$$

where diag extracts the diagonal and $D_{\mathbf{x}}$ is the diagonal matrix of $\mathbf{x}$. If SKIP performs cubic convolutional interpolation on one-dimensional grid of size m in each dimension, then $W_j \in \mathbb{R}^{n \times m}$, $K_j \in \mathbb{R}^{m \times m}$, $Q_j \in \mathbb{R}^{n \times p}$, and $T_j \in \mathbb{R}^{p \times p}$.

Constructing the SKIP kernel costs $O(dr(n + m \log m) + p^3 n \log d)$ flops and each MVM costs $O(p^2 n)$ flops, where p is the rank of the Lanczos decomposition.

7.3 Scalable GPs with derivatives

One standard approach to scaling GPs substitutes the exact kernel with an approximate kernel. When the GP fits values and gradients, one may attempt to separately approximate the kernel and the kernel derivatives. Unfortunately, this may lead to indefiniteness, as the resulting approximation is no longer a valid kernel. Instead, we differentiate the approximate kernel, which preserves positive definiteness. We do this for the SKI and SKIP kernels below, but our general approach applies to any differentiable approximate MVM.

7.3.1 D-SKI

D-SKI (SKI with derivatives) is the standard kernel matrix for Gaussian processes with derivatives, but applied to the SKI kernel. Equivalently, we differentiate the interpolation scheme:

$$k(x, x') \approx \sum_i w_i(x) k(x_i, x') \rightarrow \nabla k(x, x') \approx \sum_i \nabla w_i(x) k(x_i, x').$$

SKI uses cubic convolutional interpolation [Keys, 1981], but higher order methods lead to greater accuracy, and we therefore opt for quintic interpola-

tion [Meijering et al., 1999]. The resulting D-SKI kernel matrix has the form

$$\begin{bmatrix} K_{XX} & (\partial K_{XX})^T \\ \partial K_{XX} & \partial^2 K_{XX} \end{bmatrix} \approx \begin{bmatrix} W \\ \partial W \end{bmatrix} K_{UU} \begin{bmatrix} W \\ \partial W \end{bmatrix}^T = \begin{bmatrix} W K_{UU} W^T & W K_{UU} (\partial W)^T \\ (\partial W) K_{UU} W^T & (\partial W) K_{UU} (\partial W)^T \end{bmatrix},$$

where the elements of sparse matrices W and ∂W are determined by $w_i(x)$ and $\nabla w_i(x)$ — assuming quintic interpolation, W and ∂W will each have 6^d elements per row. As with SKI with q gridpoints, we use FFTs to obtain $O(q \log q)$ MVMs with K_{UU}. Because W and ∂W have $O(n6^d)$ and $O(nd6^d)$ nonzero elements, respectively, our MVM complexity is $O(nd6^d + q \log q)$.

7.3.2 D-SKIP

The same Hadamard of product structure of SKIP applies to the kernel matrix with derivatives. Without loss of generality, differentiating for $d = 2$ to get an approximation of K_∇ has the matrix form:

$$K_{XX}^\nabla \approx \begin{bmatrix} W_1 K_1 W_1^T & W_1 K_1 \partial W_1^T & W_1 K_1 W_1^T \\ \partial W_1 K_1 W_1^T & \partial W_1 K_1 \partial W_1^T & \partial W_1 K_1 W_1^T \\ W_1 K_1 W_1^T & W_1 K_1 \partial W_1^T & W_1 K_1 W_1^T \end{bmatrix} \odot \begin{bmatrix} W_2 K_2 W_2^T & W_2 K_2 W_2^T & W_2 K_2 \partial W_2^T \\ W_2 K_2 W_2^T & W_2 K_2 W_2^T & W_2 K_2 \partial W_2^T \\ \partial W_2 K_2 W_2^T & \partial W_2 K_2 W_2^T & \partial W_2 K_2 \partial W_2^T \end{bmatrix}. \quad (7.1)$$

Equation 7.1 expresses K_{XX}^∇ as a Hadamard product of one dimensional kernel matrices. Following this approximation, we apply the SKIP reduction [Gardner et al., 2018] and use Lanczos to further approximate equation 7.1 as $(Q_1 T_1 Q_1^T) \odot (Q_2 T_2 Q_2^T)$. This can be used for fast MVMs with the kernel matrix, see the appendix for details. Applied to kernel matrices with derivatives, we call this approach D-SKIP. D-SKIP achieves better scaling with d than D-SKI as constructing the D-SKIP kernel costs $O(d^2(n + p \log p + p^3 n \log d))$ flops, and each MVM costs $O(dp^2 n)$ flops where p is the effective rank of the kernel at each step (rank of the Lanczos decomposition). We achieve high accuracy with $p \ll n$.

7.3.3 Preconditioning

Recent work has explored several preconditioners for exact kernel matrices without derivatives [Cutajar et al., 2016]. We have had success with preconditioners of the form $M = \sigma^2 I + FF^T$ where $K_{XX}^{\nabla} \approx FF^T$ with $F \in \mathbb{R}^{n \times p}$. Solving with the Sherman-Morrison-Woodbury formula (*a.k.a* the matrix inversion lemma) is inaccurate for small σ; we use the more stable formula $M^{-1}b = \sigma^{-2}(f - Q_1(Q_1^T b))$ where Q_1 is computed in $O(p^2 n)$ time by the economy QR factorization

$$\begin{bmatrix} F \\ \sigma I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R.$$

In our experiments with solvers for D-SKI and D-SKIP, we have found that a truncated pivoted Cholesky factorization, $K_{XX}^{\nabla} \approx (\Pi L)(\Pi L)^T$ works well for the low-rank factorization. Computing the pivoted Cholesky factorization is cheaper than MVM-based preconditioners such as Lanczos or truncated eigendecompositions as it only requires the diagonal and the ability to form the rows where pivots are selected. Pivoted Cholesky is a natural choice when inducing point methods are applied as the pivoting can itself be viewed as an inducing point method where the most important information is selected to construct a low-rank preconditioner [Harbrecht et al., 2012]. The D-SKI diagonal can be formed in $O(nd6^d)$ flops while rows cost $O(nd6^d + q)$ flops; for D-SKIP both the diagonal and the rows can be formed in $O(nd)$ flops.

7.3.4 Dimensionality reduction

In many high-dimensional function approximation problems, only a few directions are relevant. That is, if $f : \mathbb{R}^d \to \mathbb{R}$ is a function to be approximated,

there is often a matrix P with $\tilde{d} < d$ orthonormal columns spanning an *active subspace* of $\mathbb{R}^d$ such that $f(x) \approx f(PP^T x)$ for all x in some domain Ω of interest [Constantine, 2015]. The optimal subspace is given by the dominant eigenvectors of the covariance matrix $C = \int_\Omega \nabla f(x) \nabla f(x)^T \, dx$, generally estimated by Monte Carlo integration. Once the subspace is determined, the function can be approximated through a Gaussian process on the reduced space, i.e. we replace the original kernel $k(x, x')$ with a new kernel $\check{k}(x, x') = k(P^T x, P^T x')$. Because we assume gradient information, dimensionality reduction based on active subspaces is a natural pre-processing phase before applying D-SKI and D-SKIP.

7.4 Experiments

Our experiments use the squared exponential (SE) kernel, which has product structure and can be used with D-SKIP; and the spline kernel, to which D-SKIP does not directly apply. We use these kernels in tandem with D-SKI and D-SKIP to achieve the fast MVMs derived in §7.3. We write D-SE to denote the exact SE kernel with derivatives.

7.4.1 Eigenspectrum approximation

D-SKI and D-SKIP with the SE kernel approximate the original kernel well, both in terms of MVM accuracy and spectral profile. Comparing D-SKI and D-SKIP to their exact counterparts in Figure 7.1, we see their matrix entries are very close (leading to MVM accuracy near 10^{-5}), and their spectral profiles are indistinguishable. The same is true with the spline kernel. Additionally, scaling

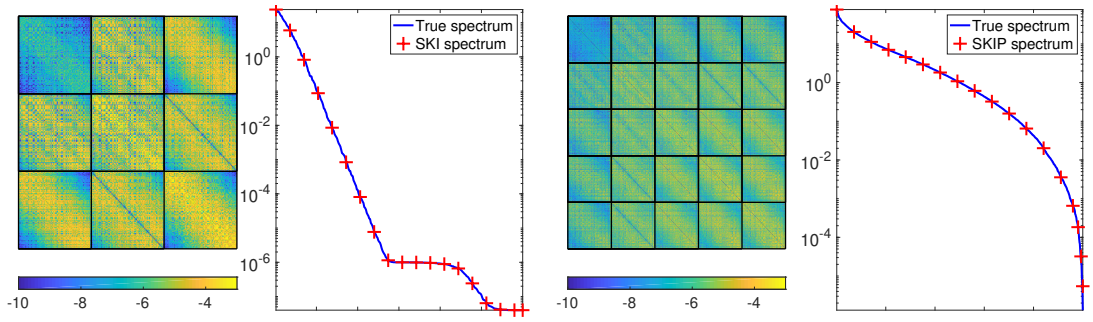


Figure 7.1: (Left two images) $\log_{10}$ error in D-SKI approximation and comparison to the exact spectrum. (Right two images) $\log_{10}$ error in D-SKIP approximation and comparison to the exact spectrum.

tests in Figure 7.2 verify the predicted complexity of D-SKI and D-SKIP. We show the relative fitting accuracy of SE, SKI, D-SE, and D-SKI on some standard test functions in Table 7.1.

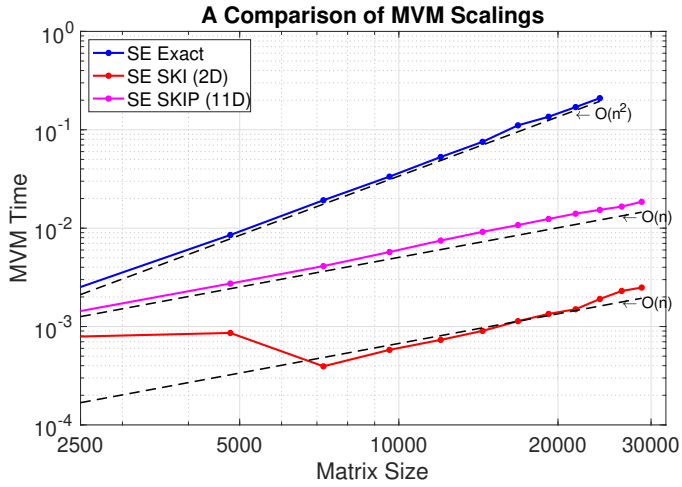


Figure 7.2: Scaling tests for D-SKI in two dimensions and D-SKIP in 11 dimensions. D-SKIP uses fewer data points for identical matrix sizes.

7.4.2 Kernel learning on test functions

We consider several popular test functions in two and three dimensions to illustrate that derivative information leads to higher accuracy. We compare D-

SKI to SKI, D-SE, and SE in Table 7.1. The fast regression of D-SKI allows us to handle a larger training set in comparable time. As a result, D-SKI achieves the lowest RMSE among all test functions.

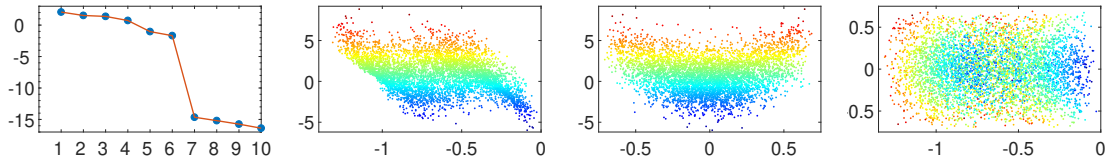
	Branin	Franke	Sine Norm	Sixhump	StyTang	Hartman3
SE	6.02e-3	8.73e-3	8.64e-3	6.44e-3	4.49e-3	1.30e-2
SKI	3.97e-3	5.51e-3	5.37e-3	5.11e-3	2.25e-3	8.59e-3
D-SE	1.83e-3	1.59e-3	3.33e-3	1.05e-3	1.00e-3	3.17e-3
D-SKI	1.03e-3	4.06e-4	1.32e-3	5.66e-4	5.22e-4	1.67e-3

Table 7.1: Relative RMSE error on 10000 testing points for test functions from [Surjanovic and Bingham, 2018], including five 2D functions (Branin, Franke, Sine Norm, Sixhump, and Styblinski-Tang) and the 3D Hartman function. We train the SE kernel on 4000 points, the D-SE kernel on $4000/(d+1)$ points, and SKI and D-SKI with SE kernel on 10000 points to achieve comparable runtimes between methods.

7.4.3 Dimensionality reduction

We apply active subspace pre-processing to the 20 dimensional Welsh test function in [Ben-Ari and Steinberg, 2007]. The top six eigenvalues of its gradient covariance matrix are well separated from the rest as seen in Figure 7.3(a). However, the function is far from smooth when projected onto the leading 1D or 2D active subspace, as Figure 7.3(b) - 7.3(d) indicates, where the color shows the function value.

We therefore apply D-SKI and D-SKIP on the 3D and 6D active subspace, respectively, using 5000 training points, and compare the prediction error against D-SE with 190 training points because of our scaling advantage. Table 7.2 reveals that while the 3D active subspace fails to capture all the variation of the function, the 6D active subspace is able to do so. These properties are demonstrated by the poor prediction of D-SKI in 3D and the excellent prediction of D-SKIP in 6D.



(a) Log Directional Variation (b) First Active Direction (c) Second Active Direction (d) Leading 2D Active Subspace

Figure 7.3: 7.3(a) shows the top 10 eigenvalues of the gradient covariance. Welsh is projected onto the first and second active direction in 7.3(b) and 7.3(c). After joining them together, we see in 7.3(d) that points of different color are highly mixed, indicating a very spiky surface.

	D-SE	D-SKI (3D)	D-SKIP (6D)
RMSE	4.900e-02	2.267e-01	3.366e-03
SMAE	4.624e-02	2.073e-01	2.590e-03

Table 7.2: Relative RMSE and SMAE prediction error for Welsh. The D-SE kernel is trained on $4000/(d+1)$ points, with D-SKI and D-SKIP trained on 5000 points. The 6D active subspace is sufficient to capture the variation of the test function.

7.4.4 Preconditioning

We discover that preconditioning is crucial for the convergence of iterative solvers using approximation schemes such as D-SKI and D-SKIP. To illustrate the performance of conjugate gradient (CG) method with and without the above-mentioned truncated pivoted Cholesky preconditioner, we test D-SKI on the 2D Franke function with 2000 data points, and D-SKIP on the 5D Friedman function with 1000 data points. In both cases, we compute a pivoted Cholesky decomposition truncated at rank 100 for preconditioning, and the number of steps it takes for CG/PCG to converge are demonstrated in Figure 7.4 below. It is clear that preconditioning universally and significantly reduces the number of steps required for convergence.

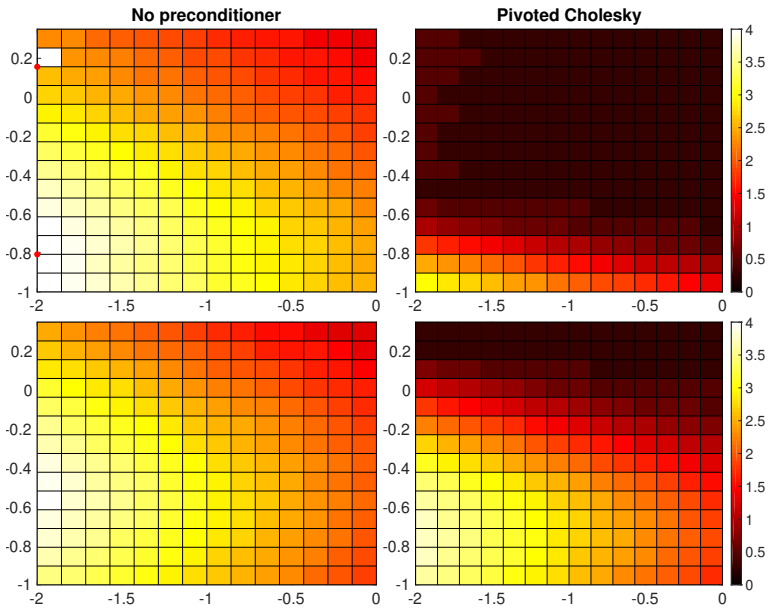


Figure 7.4: The color shows $\log_{10}$ of the number of iterations to reach a tolerance of 1e-4. The first row compares D-SKI with and without a preconditioner. The second row compares D-SKIP with and without a preconditioner. The red dots represent no convergence. The y-axis shows $\log_{10}(\ell)$ and the x-axis $\log_{10}(\sigma)$ and we used a fixed value of $s = 1$.

7.4.5 Rough terrain reconstruction

Rough terrain reconstruction is a key application in robotics, autonomous navigation, and geostatistics [Gingras et al., 2010, Konolige et al., 2010]. Through a set of terrrain measurements, the problem is to predict the underlying topography of some region. In the following experiment, we consider roughly 23 million nonuniformly sampled elevation measurements of Mount St. Helens obtained via LiDAR [Consortium, 2002]. We bin the measurements into a 970×950 grid, and downsample to a 120×117 grid. Derivatives are approximated using a finite difference scheme.

We randomly select 90% of the grid for training and the remainder for testing. We do not include results for D-SE, as its kernel matrix has dimension roughly

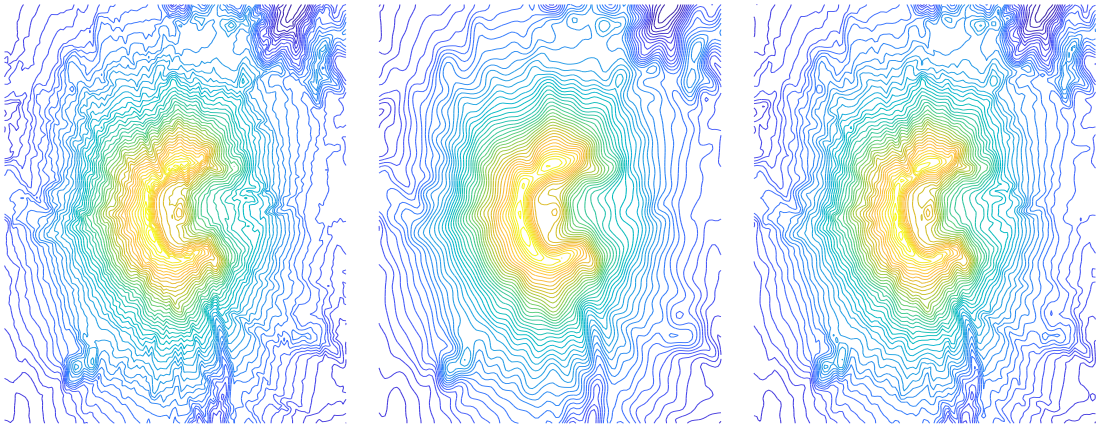
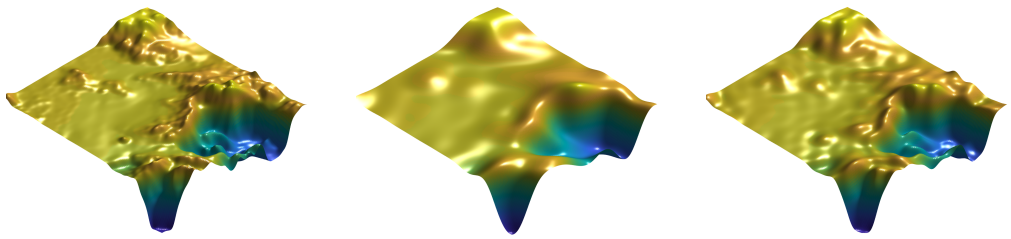


Figure 7.5: On the left is the true elevation map of Mount St. Helens. In the middle is the elevation map calculated with the SKI. On the right is the elevation map calculated with D-SKI.

$4 \cdot 10^4$. We plot contour maps predicted by SKI and D-SKI in Figure 7.5 —the latter looks far closer to the ground truth than the former. This is quantified in the following table:

	ℓ	s	σ	σ_2	Testing SMAE	Overall SMAE	Time[s]
SKI	35.196	207.689	12.865	n.a.	0.0308	0.0357	37.67
D-SKI	12.630	317.825	6.446	2.799	0.0165	0.0254	131.70

Table 7.3: The hyperparameters of SKI and D-SKI are listed. Note that there are two different noise parameters σ_1 and σ_2 in D-SKI, for the value and gradient respectively.



(a) Ground Truth (b) SKI (c) D-SKI

Figure 7.6: D-SKI is clearly able to capture more detail in the map than SKI. Note that inclusion of derivative information in this case leads to a negligible increase in calculation time.

The Korean Peninsula elevation and bathymetry dataset is sampled at a resolution of 12 cells per degree and has 180×240 entries on a rectangular grid. We take a smaller subgrid of 17×23 points as training data. To reduce data noise, we apply a Gaussian filter with $\sigma_{\text{filter}} = 2$ as a pre-processing step. We observe that the recovered surfaces with SKI and D-SKI highly resemble their respective counterparts with exact computation and that incorporating gradient information enables us to recover more terrain detail.

	ℓ	s	σ	SMAE	Time[s]
SKI	16.786	855.406	184.253	0.1521	10.094
D-SKI	9.181	719.376	29.486	0.0746	11.643

7.4.6 Implicit surface reconstruction

Reconstructing surfaces from point cloud data and surface normals is a standard problem in computer vision and graphics. One popular approach is to fit an implicit function that is zero on the surface with gradients equal to the surface normal. Local Hermite RBF interpolation has been considered in prior work [Macedo et al., 2011], but this approach is sensitive to noise. In our experiments, using a GP instead of splining reproduces implicit surfaces with very high accuracy. In this case, a GP with derivative information is required, as the function values are all zero.

In Figure 7.7, we fit the Stanford bunny using 25000 points and associated normals, leading to a K_{XX}^{∇} matrix of dimension 10^5, clearly far too large for exact training. We therefore use SKI with the thin-plate spline kernel, with a total of 30 grid points in each dimension. The left image is a ground truth mesh of the underlying point cloud and normals. The middle image shows the same mesh,

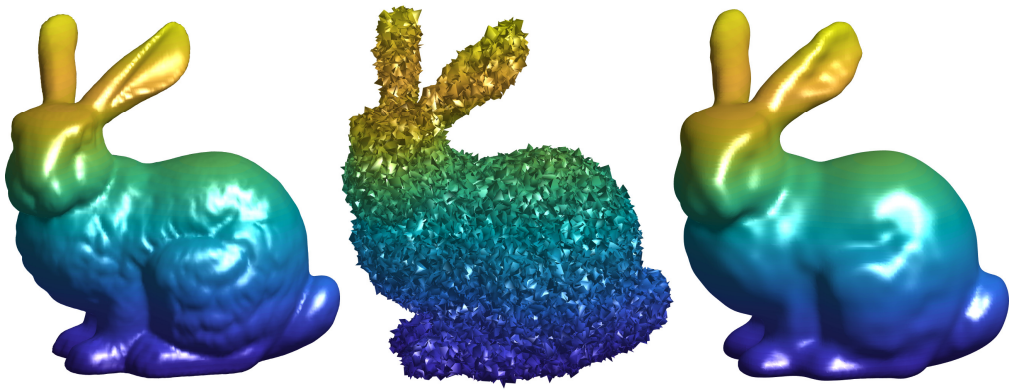


Figure 7.7: (Left) Original surface (Middle) Noisy surface (Right) SKI reconstruction from noisy surface ($s = 0.4, \sigma = 0.12$)

but with heavily noised points and normals. Using this noisy data, we fit a GP and reconstruct a surface shown in the right image, which looks very close to the original.

7.4.7 Bayesian optimization with derivatives

Prior work examines Bayesian optimization (BO) with derivative information in low-dimensional spaces to optimize model hyperparameters [Wu et al., 2017]. Wang et al. consider high-dimensional BO (without gradients) with random projections uncovering low-dimensional structure [Wang et al., 2013]. We propose BO with derivatives and dimensionality reduction via active subspaces, detailed in Algorithm 1.

Algorithm 1 estimates the active subspace and fits a GP with derivatives in the reduced space. Kernel learning, fitting, and optimization of the acquisition function all occur in this low-dimensional subspace. In our tests, we use the expected improvement (EI) acquisition function, which involves both the mean and predictive variance. We consider two approaches to rapidly evaluate the

Algorithm 6 BO with derivatives and active subspace learning

while Budget not exhausted **do**
  Calculate active subspace projection $P \in \mathbb{R}^{D \times d}$ using sampled gradients
  Optimize acquisition function, $u_{n+1} = \arg\max \mathcal{A}(u)$ with $x_{n+1} = Pu_{n+1}$
  Sample point x_{n+1}, value f_{n+1}, and gradient ∇f_{n+1}
  Update data $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{x_{n+1}, f_{n+1}, \nabla f_{n+1}\}$
  Update hyperparameters of GP with gradient defined by kernel $k(P^T x, P^T x')$
end while
end

predictive variance $v(x) = k(x, x) - K_{xX}\tilde{K}_{XX}^{-1}K_{Xx}$ at a test point x. In the first approach, which provides a biased estimate of the predictive variance, we replace $\tilde{K}_{XX}^{-1}$ with the preconditioner solve computed by pivoted Cholesky; using the stable QR-based evaluation algorithm, we have

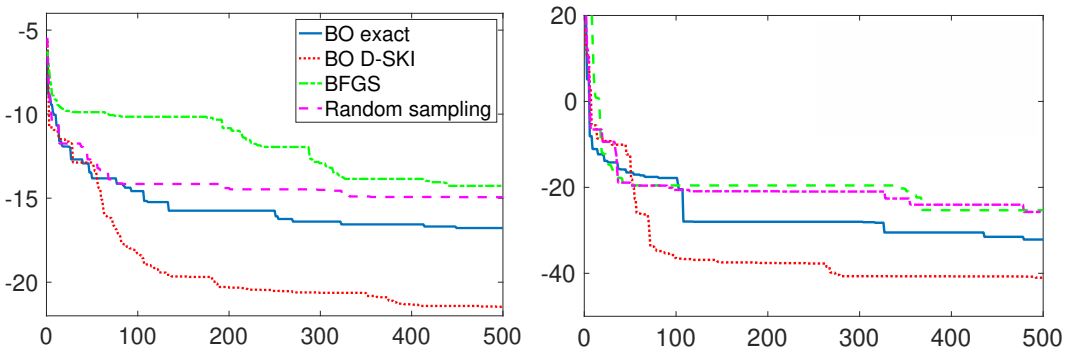
$$v(x) \approx \hat{v}(x) \equiv k(x, x) - \sigma^{-2}(\|K_{Xx}\|^2 - \|Q_1^T K_{Xx}\|^2).$$

In the second approach, we use a randomized estimator as in [Bekas et al., 2007] to compute the predictive variance at many points X' simultaneously, and use the pivoted Cholesky approximation as a control variate to reduce the estimator variance:

$$v_{X'} = \text{diag}(K_{X'X'}) - \mathbb{E}_z\left[z \odot (K_{X'X}\tilde{K}_{XX}^{-1}K_{XX'}z - K_{X'X}M^{-1}K_{XX'}z)\right] - \hat{v}_{X'}.$$

The latter approach is unbiased, but gives very noisy estimates unless many probe vectors z are used. Both the pivoted Cholesky approximation to the predictive variance and the randomized estimator resulted in similar optimizer performance in our experiments.

To test this algorithm, we consider five instances of the 5D Ackley and 5D Rastrigin functions randomly embedded in $[-10, 15]^{50}$ and $[-4, 5]^{50}$, respectively. In Figure 7.8(a) and Figure 7.8(b), we show the performance of our algorithm using the D-SKI kernel and the EI acquisition function. We fix $d = 2$, and at



(a) BO on Ackley (b) BO on Rastrigin

Figure 7.8: In the following experiments, 5D Ackley and 5D Rastrigin are embedded into 50 a dimensional space. We run Algorithm 1, comparing it with BO exact, multi-start BFGS, and random sampling. D-SKI with active subspace learning clearly outperforms the other methods.

each iteration we pick two directions in the estimated active subspace at random. We also compare to three other methods: BO with EI and no gradients in the original space; multi-start BFGS with full gradients; and random search. In both examples, the BO variants perform better than the alternatives, and our method outperforms standard BO.

7.5 Conclusion

Gradients are a valuable additional source of information for GP regression, but inclusion of d extra pieces of information per point naturally leads to new scaling issues. We introduced two methods to deal with these scaling issues: D-SKI and D-SKIP. Both are structured interpolation methods, and the latter also uses kernel product structure. We have discussed practical details — preconditioning is necessary to guarantee convergence of iterative methods and active subspace calculation reveals low-dimensional structure when gradients are available. We

presented several experiments with kernel learning, dimensionality reduction, terrain reconstruction, implicit surface fitting, and scalable Bayesian optimization with gradients. For simplicity, these examples all possessed full gradient information; however, our methods trivially extend if only partial gradient information is available.

CHAPTER 8

CONCLUSION

We have presented a series of improved Bayesian optimization (BO) methods in this dissertation that we hope will contribute to the greater body of data-driven global optimization literature. These methods largely concern BO in the setting where there is an a-priori, known budget, also known as *non-myopic BO*. In Chapter 3, we formulated this problem as a finite-horizon, infinite state and action space Markov decision process (MDP), where the horizon of the MDP is equivalent to the given BO iteration budget. We also showed in Chapter 3 that the standard expected improvement (EI) acquisition is equivalent to a greedy MDP policy, which explains its overly-exploitative empirical behavior.

High-dimensional MDPs, especially ones with an infinite state and action space, are notoriously difficult to solve. In Chapter 4, we discuss rollout, a tractable approximation to the optimal MDP policy. We apply a series of variance reduction techniques, including quasi-Monte carlo (QMC), common random numbers, and control variates, in order to further decrease the overhead of rollout. We show that rollout leads to improvement BO performance of multi-modal function, and identify the impact of model error as a key challenge to improving the performance of non-myopic BO acquisition functions. We also investigate policy search, an alternative approximation to the optimal MDP policy, in which a best policy out of a parameterized policy class is chosen. In the context of BO, this corresponds to selecting the best acquisition function that yields the highest expected decrease in the objective function over the next h steps.

BO implicitly measures progress and convergence in terms of iterations, which assumes that each function evaluation takes the same amount of time . In Chapter 5, we show that this assumption is rarely true in practical problems, in which the cost of evaluating the function varies across the optimization domain.

This cost may be money, time, or material consumption, and the problem of BO with non-uniform evaluation cost and a-priori cost budget is known as *cost-constrained BO*. We formulate cost-constrained BO as a constrained MDP (CMDP), which is an MDP with an additional set of cost constraints, and similarly extend rollout to the CMDP setting. We show that rollout yields improved performance over traditional acquisition functions such as EI.

Unfortunately, the overhead of CMDP rollout is very large, rendering it unsuitable for all but the most expensive problems. In Chapter 6, we introduce cheap and straightforward, two-phase heuristic to tackle cost-constrained BO we call cost apportioned BO (CArBO). The first is a cost-effective initial design phase, during which we prioritize cheap exploration. The second is a cost-cooling phase, during which the cost constraint is lessened over time to encourage expensive exploitation. We show that CArBO yields significant improvements over traditional acquisition functions, both in the sequential BO and batch BO settings. We also build cost models by counting flops and using robust regression to determine flop constants, and show that using cost models with lower prediction error increase cost-constrained BO convergence speed.

Finally, we consider BO in the setting where derivative information is available in Chapter 7. GP regression of both values and derivatives scales cubically in both the number of data points and the dimension of the problem. We extend existing scalable GP regression methods, structured kernel interpolation (SKI) and structured kernel interpolation with product structure (SKIP) to include derivative information, which we call D-SKI and D-SKIP respectively. We also discuss dimensionality reduction through computation of the active subspace, which is the span of the dominant directional derivatives, and show that BO

combined with gradient information and active subspace dimensionality reduction performs better than both BO without derivative information and randomly restarted local optimization methods such as BFGS.

Global optimization is an important problem of ever-growing relevance, and no doubt there will be much research progress made in the upcoming years concerning global optimization methodologies. We hope this dissertation represents a not-immodest step towards principled, data-driven global optimization under a finite budget, and plan to continue this line of work beyond its publishing.

BIBLIOGRAPHY

[gpy, 2016] (2016). GPyOpt: A Bayesian optimization framework in Python. `http://github.com/SheffieldML/GPyOpt`.

[Abdolshah et al., 2019] Abdolshah, M., Shilton, A., Rana, S., Gupta, S., and Venkatesh, S. (2019). Cost-aware multi-objective bayesian optimisation. *arXiv preprint arXiv:1909.03600*.

[Alom et al., 2018] Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., Van Esesn, B. C., Awwal, A. A. S., and Asari, V. K. (2018). The history began from Alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*.

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

[Azimi et al., 2010] Azimi, J., Fern, A., and Fern, X. Z. (2010). Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pages 109–117.

[Back, 1996] Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

[Bekas et al., 2007] Bekas, C., Kokiopoulou, E., and Saad, Y. (2007). An estimator for the diagonal of a matrix. *Applied Numerical Mathematics*, 57(11-12):1214–1229.

[Bellman, 1952] Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716.

[Bellman, 1961] Bellman, R. E. (1961). *Adaptive control processes: a guided tour*, volume 2045. Princeton university press.

[Bemporad et al., 2002] Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E. N. (2002). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1):3–20.

[Ben-Ari and Steinberg, 2007] Ben-Ari, E. N. and Steinberg, D. M. (2007). Modeling data from computer experiments: an empirical comparison of kriging with MARS and projection pursuit regression. *Quality Engineering*, 19(4):327–338.

[Bertsekas, 2005] Bertsekas, D. (2005). Rollout algorithms for constrained dynamic programming. *Lab. for Information and Decision Systems Report*, 2646.

[Bertsekas, 2010] Bertsekas, D. (2010). Rollout algorithms for discrete optimization: A survey. *Handbook of Combinatorial Optimization, D. Zu and P. Pardalos, Eds. Springer*.

[Bertsekas, 1995] Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*, volume I. Athena Scientific Belmont, MA, 4th edition.

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

[Bull, 2011] Bull, A. D. (2011). Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12(Oct):2879–2904.

[Caflisch, 1998] Caflisch, R. E. (1998). Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 7:1–49.

[Chatterjee et al., 2006] Chatterjee, K., Majumdar, R., and Henzinger, T. A. (2006). Markov decision processes with multiple objectives. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 325–336. Springer.

[Consortium, 2002] Consortium, P. S. L. (2002). Mount Saint Helens LiDAR data. University of Washington.

[Constantine, 2015] Constantine, P. G. (2015). *Active subspaces: Emerging ideas for dimension reduction in parameter studies*. SIAM.

[Cutajar et al., 2016] Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. (2016). Preconditioning kernel matrices. pages 2529–2538.

[Damblin et al., 2013] Damblin, G., Couplet, M., and Iooss, B. (2013). Numerical studies of space-filling designs: optimization of latin hypercube samples and subprojection properties. *Journal of Simulation*, 7(4):276–289.

[Deb, 2001] Deb, K. (2001). *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons.

[Deng, 2012] Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142.

[Di et al., 2013] Di, S., Wang, C.-L., and Cappello, F. (2013). Adaptive algorithm for minimizing cloud task length with prediction errors. *IEEE Transactions on Cloud Computing*, 2(2):194–207.

[Dolatnia et al., 2016] Dolatnia, N., Fern, A., and Fern, X. (2016). Bayesian optimization with resource constraints and production. In *Twenty-Sixth International Conference on Automated Planning and Scheduling*.

[Dong et al., 2017] Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. (2017). Scalable log determinants for Gaussian process kernel learning. pages 6330–6340.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[Eriksson et al., 2018] Eriksson, D., Dong, K., Lee, E., Bindel, D., and Wilson, A. G. (2018). Scaling gaussian process regression with derivatives. In *Advances in Neural Information Processing Systems*, pages 6867–6877.

[Falkner et al., 2018] Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*.

[Forrester et al., 2008] Forrester, A., Keane, A., et al. (2008). *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons.

[Forrester et al., 2007] Forrester, A. I., Sóbester, A., and Keane, A. J. (2007). Multi-fidelity optimization via surrogate modelling. *Proceedings of the royal society a: mathematical, physical and engineering sciences*, 463(2088):3251–3269.

[Frazier, 2018a] Frazier, P. I. (2018a). Bayesian optimization. In Gel, E. and Ntaimo, L., editors, *Recent Advances in Optimization and Modeling of Contemporary Problems*, pages 255–278. INFORMS.

[Frazier, 2018b] Frazier, P. I. (2018b). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

[Frazier et al., 2008] Frazier, P. I., Powell, W. B., and Dayanik, S. (2008). A knowledge-gradient policy for sequential information collection. *SIAM Journal on Control and Optimization*, 47(5):2410–2439.

[Gardner et al., 2018] Gardner, J. R., Pleiss, G., Wu, R., Weinberger, K. Q., and

Wilson, A. G. (2018). Product kernel interpolation for scalable Gaussian processes. In *Artificial Intelligence and Statistics (AISTATS)*.

[Garnett et al., 2010] Garnett, R., Osborne, M. A., and Roberts, S. J. (2010). Bayesian optimization for sensor set selection. In *Proceedings of the 9th ACM/IEEE Conference on Information Processing in Sensor Networks*, pages 209–219.

[Gingras et al., 2010] Gingras, D., Lamarche, T., Bedwani, J.-L., and Dupuis, É. (2010). Rough terrain reconstruction for rover motion planning. In *Proceedings of the Canadian Conference on Computer and Robot Vision (CRV)*, pages 191–198. IEEE.

[González et al., 2016] González, J., Dai, Z., Hennig, P., and Lawrence, N. (2016). Batch Bayesian optimization via local penalization. In *Artificial Intelligence and Statistics*, pages 648–657.

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

[Goodman, 2020] Goodman (2020). Gaussian sampling tutorial.

[Harbrecht et al., 2012] Harbrecht, H., Peters, M., and Schneider, R. (2012). On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440.

[Hernández-Lobato et al., 2014] Hernández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the 28th Conference on Neural Information Processing Systems*, pages 918–926.

[Hoffman et al., 2011] Hoffman, M. D., Brochu, E., and de Freitas, N. (2011). Portfolio allocation for Bayesian optimization. In *UAI*, pages 327–336. Citeseer.

[Huang et al., 2010] Huang, L., Jia, J., Yu, B., Chun, B.-G., Maniatis, P., and Naik, M. (2010). Predicting execution time of computer programs using sparse polynomial regression. In *Advances in Neural Information Processing Systems*, pages 883–891.

[Hutter et al., 2014] Hutter, F., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2014). Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111.

[Jones et al., 1998a] Jones, D. R., Schonlau, M., and Welch, W. J. (1998a). Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492.

[Jones et al., 1998b] Jones, D. R., Schonlau, M., and Welch, W. J. (1998b). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492.

[Kandasamy et al., 2017] Kandasamy, K., Dasarathy, G., Schneider, J., and Poczos, B. (2017). Multi-fidelity Bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1799–1808. JMLR. org.

[Kawaguchi et al., 2015] Kawaguchi, K., Kaelbling, L. P., and Lozano-Pérez, T. (2015). Bayesian optimization with exponential convergence. In *Advances in Neural Information Processing Systems*, pages 2809–2817.

[Keys, 1981] Keys, R. (1981). Cubic convolution interpolation for digital image processing. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(6):1153–1160.

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

[Kirk, 2012] Kirk, R. E. (2012). Experimental design. *Handbook of Psychology, Second Edition*, 2.

[Klein et al., 2016] 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*.

[Klein et al., 2017] Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In Singh, A. and Zhu, J., editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 528–536.

[Klein and Hutter, 2019] Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

[Konolige et al., 2010] Konolige, K., Agrawal, M., and Sola, J. (2010). Large-

scale visual odometry for rough terrain. In *Robotics Research*, pages 201–212. Springer.

[Lam and Willcox, 2017] Lam, R. and Willcox, K. (2017). Lookahead Bayesian optimization with inequality constraints. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, pages 1890–1900.

[Lam et al., 2016] Lam, R., Willcox, K., and Wolpert, D. H. (2016). Bayesian optimization with a finite budget: An approximate dynamic programming approach. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, pages 883–891.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lee et al., 2020a] Lee, E. H., Eriksson, D., Cheng, B., McCourt, M., and Bindel, D. (2020a). Efficient rollout strategies for bayesian optimization. *arXiv preprint arXiv:2002.10539*.

[Lee et al., 2020b] Lee, E. H., Perrone, V., Archambeau, C., and Seeger, M. (2020b). Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*.

[Li et al., 2017] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

[Macedo et al., 2011] Macedo, I., Gois, J. P., and Velho, L. (2011). Hermite radial basis functions implicits. *Computer Graphics Forum*, 30(1):27–42.

[MacKay and Gibbs, 1997] MacKay, D. and Gibbs, M. (1997). Efficient implementation of Gaussian processes. *Neural Computation*.

[Meijering et al., 1999] Meijering, E. H. W., Zuiderveld, K. J., and Viergever, M. A. (1999). Image reconstruction by convolution with symmetrical piecewise *n*th-order polynomial kernels. *IEEE Transactions on Image Processing*, 8(2):192–201.

[Mockus et al., 1978] Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The ap-

plication of Bayesian methods for seeking the extremum. *Towards Global Optimization*, 2(117-129):2.

[Morokoff and Caflisch, 1994] Morokoff, W. J. and Caflisch, R. E. (1994). Quasi-random sequences and their discrepancies. *SIAM Journal on Scientific Computing*, 15(6):1251–1279.

[Morokoff and Caflisch, 1995] Morokoff, W. J. and Caflisch, R. E. (1995). Quasi-Monte Carlo integration. *Journal of computational physics*, 122(2):218–230.

[Niederreiter, 1988] Niederreiter, H. (1988). Low-discrepancy and low-dispersion sequences. *Journal of number theory*, 30(1):51–70.

[Osborne et al., 2009] Osborne, M. A., Garnett, R., and Roberts, S. J. (2009). Gaussian processes for global optimization. In *Proceedings of the 3rd International Conference on Learning and Intelligent Optimization (LION3)*, pages 1–15.

[Owen, 2009] Owen, A. (2009). Variance reduction.

[Papageorgiou, 2003] Papageorgiou, A. (2003). Sufficient conditions for fast quasi-Monte Carlo convergence. *Journal of Complexity*, 19(3):332–351.

[Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Peise and Bientinesi, 2012] Peise, E. and Bientinesi, P. (2012). Performance modeling for dense linear algebra. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 406–416. IEEE.

[Piunovskiy, 2006] Piunovskiy, A. B. (2006). Dynamic programming in constrained Markov decision processes. *Control and Cybernetics*, 35(3):645.

[Poloczek et al., 2017] Poloczek, M., Wang, J., and Frazier, P. (2017). Multi-information source optimization. In *Advances in Neural Information Processing Systems*, pages 4288–4298.

[Powell, 2007] Powell, W. B. (2007). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. Wiley, 2nd edition.

[Priya et al., 2011] Priya, R., de Souza, B. F., Rossi, A. L., and de Carvalho, A. C. (2011). Predicting execution time of machine learning tasks using metalearning. In *2011 World Congress on Information and Communication Technologies*, pages 1193–1198. IEEE.

[Pronzato, 2017] Pronzato, L. (2017). Minimax and maximin space-filling designs: some properties and methods for construction. *Journal de la Societe Française de Statistique*.

[Pronzato and Müller, 2012] Pronzato, L. and Müller, W. G. (2012). Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701.

[Puterman, 2014] Puterman, M. L. (2014). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

[Qin et al., 2017] Qin, C., Klabjan, D., and Russo, D. (2017). Improving the expected improvement algorithm. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, pages 5381–5391.

[Rasmussen and Williams, 2006] Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

[Rios and Sahinidis, 2013] Rios, L. M. and Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293.

[Roijers et al., 2013] Roijers, D. M., Vamplew, P., Whiteson, S., and Dazeley, R. (2013). A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113.

[Rudolph, 1996] Rudolph, G. (1996). Convergence of evolutionary algorithms in general search spaces. In *Proceedings of IEEE international conference on evolutionary computation*, pages 50–54. IEEE.

[Ryan and Morgan, 2007] Ryan, T. P. and Morgan, J. (2007). Modern experimental design. *Journal of Statistical Theory and Practice*, 1(3-4):501–506.

[Saad, 1992] Saad, Y. (1992). *Numerical methods for large eigenvalue problems*. Manchester University Press.

[Saad, 2003] Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

[Shah and Ghahramani, 2015] 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*, pages 3330–3338.

[Shahriari et al., 2016] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

[Singer and Nelder, 2009] Singer, S. and Nelder, J. (2009). Nelder-Mead algorithm. *Scholarpedia*, 4(7):2928.

[Smola and Bartlett, 2001] Smola, A. J. and Bartlett, P. (2001). Sparse greedy Gaussian process regression. In *Advances in Neural Information Processing Systems 13*.

[Snelson and Ghahramani, 2005] Snelson, E. and Ghahramani, Z. (2005). Sparse Gaussian processes using pseudo-inputs. pages 1257–1264.

[Snelson et al., 2004] Snelson, E., Ghahramani, Z., and Rasmussen, C. E. (2004). Warped Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 337–344.

[Snoek et al., 2012] Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Proceedings of the 26th Conference on Neural Information and Processing Systems*, pages 2951–2959.

[Snoek et al., 2015] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable Bayesian optimization using deep neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2171–2180.

[Solis and Wets, 1981] Solis, F. J. and Wets, R. J.-B. (1981). Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30.

[Srinivas et al., 2010] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. *Proceedings of the 27th International Conference on Machine Learning*, pages 1015–1022.

[Stein, 1987] Stein, M. (1987). Large sample properties of simulations using latin hypercube sampling. *Technometrics*, 29(2):143–151.

[Surjanovic and Bingham, 2018] Surjanovic, S. and Bingham, D. (2018). Virtual library of simulation experiments: Test functions and datasets. http://www.sfu.ca/ ssurjano.

[Surjanovic and Bingham, 2020] Surjanovic, S. and Bingham, D. (2020). Virtual library of simulation experiments: Test functions and datasets.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition.

[Swersky et al., 2013] Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 2004–2012.

[Trefethen and Bau III, 1997] Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam.

[Wang and Jegelka, 2017] Wang, Z. and Jegelka, S. (2017). Max-value entropy search for efficient Bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 3627–3635.

[Wang et al., 2017] Wang, Z., Li, C., Jegelka, S., and Kohli, P. (2017). Batched high-dimensional Bayesian optimization via structural kernel learning. *arXiv preprint arXiv:1703.01973*.

[Wang et al., 2013] Wang, Z., Zoghi, M., Hutter, F., Matheson, D., De Freitas, N., et al. (2013). Bayesian optimization in high dimensions via random embeddings. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 1778–1784.

[Wendland, 2004] Wendland, H. (2004). *Scattered data approximation*, volume 17. Cambridge University Press.

[Wilson et al., 2015] Wilson, A. G., Dann, C., Lucas, C., and Xing, E. P. (2015). The human kernel. In *Advances in Neural Information Processing Systems*, pages 2854–2862.

[Wilson and Nickisch, 2015] Wilson, A. G. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). pages 1775–1784.

[Wilson et al., 2018] Wilson, J., Hutter, F., and Deisenroth, M. (2018). Maximizing acquisition functions for bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 9884–9895.

[Wu and Frazier, 2016] Wu, J. and Frazier, P. (2016). The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 3126–3134.

[Wu and Frazier, 2019] Wu, J. and Frazier, P. (2019). Practical two-step lookahead Bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 9810–9820.

[Wu et al., 2017] Wu, J., Poloczek, M., Wilson, A. G., and Frazier, P. (2017). Bayesian optimization with gradients. pages 5273–5284.

[Yang et al., 2018] Yang, C., Akimoto, Y., Kim, D. W., and Udell, M. (2018). Oboe: Collaborative filtering for AutoML initialization. *arXiv preprint arXiv:1808.03233*.

[Yue and Kontar, 2019] Yue, X. and Kontar, R. A. (2019). Lookahead Bayesian optimization via rollout: Guarantees and sequential rolling horizons. *arXiv preprint arXiv:1911.01004*.

[Zinzen et al., 2009] Zinzen, R., Girardot, C., Gagneur, J., Braun, M., and Furlong, E. (2009). Combinatorial binding predicts spatio-temporal cis-regulatory activity. *Nature*, 462(7269):65–70.