Adaptive Bayesian Optimization for Dynamic Problems

Favour Mandanji Nyikosa
Linacre College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Hilary 2018

To the memory of my parents, Chisaya and Polile, and my grandmother Florence. I know you'd all be proud.

And to Biston Azere, who taught me the value of knowledge.

Acknowledgements

I am greatly indebted to the following for their help during my research journey:

- Members of my family, whose words of encouragement and unconditional support helped me immensely during my time in Oxford. I am particularly thankful to Chapita, Effinas and my sister Polile for continually checking on me to ensure I was doing well;

- Steve Roberts and Mike Osborne, who supervised this work, for their helpful and indispensable advice, guidance on how to approach the research problem, answering my endless questions, always having time for me despite their busy schedules, and promptly replying to my emails—qualities of great supervisors;

- Members of the machine learning research group in Oxford, who are too numerous to list, whose insightful conversations always gave me new ideas to explore. I'd like to extend special thanks to Ali Rizvi, Elmarie Van Heerden, Yves-Laurent Kom Samo, who made me feel less lost in the machine learning jungle;

- Nchimunya Tebeka, who spared time to look at my writing in spite of her busy schedule and immense workload. She proofread a bulk of the early drafts of this thesis despite its technical intricacies and kept my lengthy sentences in check. Mistakes, if any, are all mine;

- And the Rhodes Trust, Linacre college and the Oxford-Man Institute of Quantitative Finance whose invaluable support made this research possible.

Statement of Originality

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. Notwithstanding the use of the first person of plural to express personal views throughout this thesis, this dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements.

Parts of this work have appeared in a workshop paper and technical reports as described below, of which I am the main author. Some of the figures in this thesis are taken from these publications, with possible modifications.

Chapter 3

- The contents of this chapter are based on an unpublished technical report titled *Bayesian Optimization for Dynamic Problems* that is also under review at *ICML* 2018.

Chapter 5

- A preliminary version of the work in this chapter appeared in *Adaptive Bayesian Optimisation for Online Portfolio Selection* at the Workshop on Bayesian Optimization at *NIPS* 2015.
- The contents of this chapter are based on an unpublished technical report titled *Adaptive Configuration Oracle for Online Portfolio Selection* that is also under review at *UAI* 2018.

Chapters 1, 3, 4 and 5

- Selected parts of these chapters appear in a submission to the *Journal of Machine Learning Research* (JMLR) titled *Bayesian Optimization for Adaptive Parameter Configuration*, and is under review.

Favour Mandanji Nyikosa, April 2018

Abstract

This thesis studies the problem of tracking the extremum of an objective function that is latent, noisy and expensive to evaluate. This problem is notable because many large-scale learning systems with complex models operating on non-stationary data have meta-problems whose solutions require the tracking of an evolving extremum.

We start by describing dynamic optimization problems and model them using spatiotemporal Gaussian process priors. We construct an intelligent search mechanism that uses the learnt insights to skillfully guide the search by dynamically modifying the feasible search region as a device to keep track of the evolution. We also show that this mechanism induces a natural approximation scheme for cases where the number of samples for the model becomes too expensive for inference. We test the resulting method on synthetic and real-world problems.

In the next part of the thesis, we demonstrate the utility of the method on pertinent real-world meta-problems occurring in essential and widely-used large-scale learning systems.

We start by proposing intelligent meta-heuristics, which are derived from our resulting method, for the learning rate adaptation meta-problem that is part of many large-scale learning systems trained by stochastic gradient descent. We incorporate hyper-gradients into the model for the meta-problem and use the additional insights to aid the intelligent meta-heuristic search. We test the resulting adaptation scheme on training various types of neural network models to demonstrate its utility through the significant performance improvements.

We also exploit the wealth of prior art to construct a configuration oracle that adapts various types of system parameters in online trading algorithms as they interact with noisy and nonstationary financial data. We test the oracle on multiple algorithms operating using widely-accepted trading principles and demonstrate notable performance gains.

The importance of this study is that it proposes a mechanism for intelligently exploiting vast amounts of prior knowledge that exists for these meta-problems. The intelligent use of this knowledge significantly improves the performance of the primary learning systems, which are vital for solving many practical problems, operating in dynamic environments.

Contents

List of Figures

List of Tables

1 Introduction

Nothing at all takes place in the universe in which some rule of the maximum or minimum does not appear.
— Leonhard Euler, 1744

1.1 Motivation

With the advent of an increasing number of human and non-human activities producing large amounts of noisy and nonstationary data, there has been an associated increase in the number and types of problems critical to improving people's lives. As a result of this data explosion, the need for large-scale learning systems to help solve these pressing problems has become more crucial than ever. Evidence shows that we are in the *Zettabyte* era where more than 1 Zettabytes (10^9 Terabytes) of data is produced every year, and this number is projected to increase (Cisco Visual Networking Index 2017). Many governments have responded to this trend by identifying *big data* problems as a priority for future economic growth and having a potentially significant positive impact on people's lives (Willetts 2013).

The large-scale learning systems for solving these arising problems use so-

phisticated models to handle the search for meaningful insights in the data. However, due to the nonstationary nature of the data, the effectiveness of these models varies depending on the regime of processed data. Standard practice dictates that the configuration of the learning systems should be done on a held-out set of existing data in an expensive, time-consuming process that may involve painstaking and laborious hand-tuning. To mitigate this, some practitioners have developed experience-based heuristics to adjust the learning systems as a mechanism for adaptation. However, this stopgap is a difficult undertaking because this meta-problem is not naturally amenable to human intuition due to its latent, noisy and expensive nature.

This meta-problem is critical for the internal configuration of large-scale learning systems. The provision of automatic solutions to this problem is an efficient way for improving the main learning system. Some renowned thought leaders in the area of large-scale machine learning have highlighted the importance of solving these meta-problems. Hassabis (2017) observed that the solutions to meta-problems are often easier to model than the problems associated with the primary system, but these solutions can have a tremendous impact on the overall system performance. Bengio (2012) also highlighted the importance of the hyperparameters in large-scale neural network systems, stating that their correct configuration is as necessary as the learning task itself. These arguments highlight the essential role meta-systems provide in the ecosystem of intelligent inference.

Despite these meta-problems being unknown, noisy and expensive, there is a large amount of prior experience that can be exploited to create models that can intelligently solve them. This experience tells us that the best parameters for these systems can be described as being the minimum or maximum of some latent objective function that characterizes the optimality of the configurations. However, due to the nonstationarity of the data, these best settings change with time. Therefore, a solution to this problem must naturally produce means by which the system adapts to changes by tracking these minimums or

maximums as they evolve. We call this a *dynamic optimization problem*. Primarily, the solution method must determine *what* configurations to use and *when* to change them. A technique that solves this problem is not only a good solution for configuration meta-problems but is also ubiquitous and useful in countless practical situations. Proposing such a mechanism and demonstrating its efficacy in a diverse range of scenarios is the subject of this thesis.

1.2 Proposed Approach

Since these meta-problems are latent, noisy and expensive, and there is a significant amount of existing prior art, we resort to the auspices of Bayesian optimization with Gaussian process priors. We characterize the meta-solution to these adaptive configuration problems as one that can effectively track the minimum or the maximum of a temporally evolving objective function. We assume that the evolution is an inherent property of the function and not due to its evaluation by the optimization procedure.

Bayesian optimization (BO) (Mockus 1975) is a sequential design strategy for optimizing a function of this nature. The strategy uses a surrogate model to learn the latent objective function from available data samples and searches for a suitable point to evaluate the objective function. This search is performed using some predefined heuristics to get closer to the optimum. The general strategy is to have heuristics that intelligently and automatically *explore* and *exploit* the objective function.

The goal of this dissertation is to extend Bayesian optimization to the problem of tracking the minimum or maximum of a temporally evolving noisy, latent and expensive objective function within a fixed horizon of iterations.

For our approach, we are interested in latent dynamic optimization problems that are expensive to evaluate. These are problems whose evaluations may involve an expensive or time-consuming physical experiment or simulation. We use Bayesian optimization with Gaussian process priors to have a data-efficient

method that quickly converges to an optimum, which is important given the cost of function evaluations. Additionally, using Gaussian processes has the benefit of taking advantage of the rich prior art that exists for the types of real-world problems we are interested in. The supplemental merit of using Gaussian processes as a surrogate model is that they provide a mechanism for handling uncertainty about the objective function over the search space, and can efficiently capture changes through appropriate Bayesian updates.

Our approach is to model the temporally evolving function as a spatiotemporal Gaussian process prior and treat its instances as being defined on subspaces ordered along the temporal dimension. The solution method only optimizes part of the objective function model defined on the subspace corresponding to the times of interest, with bounds determined by what is learnt from the model. This is done by adaptively learning and modifying the box constraints of this optimization problem as time progresses. This is possible because using spatiotemporal Gaussian process priors allows us to exploit the learnt relationships in space and time of the dynamic optimization problem. The additional benefits come from the Bayesian optimization methodology that provides an automatic trade-off of exploring and exploiting the objective function that subsequently leads to fewer wasted costly evaluations. By exploiting temporal correlations, the proposed method also determines *when* to make evaluations, *how fast* to make those evaluations, and it induces an appropriate budget of steps based on the available information learnt by the surrogate model.

1.3 Thesis Structure

Chapter 2 is a brief survey of the background material used in this work. It may be skipped on first reading and referred to when relevant sections are addressed in the thesis.

Bayesian optimization has two main aspects that make it work. As illustrated in Figure 1.1, the first component is the surrogate model that models the latent objective function, and the second is the heuristic strategy used for deciding where to evaluate the latent function. To address dynamic optimization problems using Bayesian optimization, we will propose practical extensions corresponding to these two aspects in Chapter 3. We will first define dynamic optimization problems as is relevant to the goals of this thesis and describe an appropriate family of surrogate models for them. We will also explain the prescribed practical extensions to the heuristic search methodology applicable to dynamic optimization problems, and a mechanism for approximating the surrogate model when inference becomes too expensive. Lastly, we provide empirical analyses of the resulting method on synthetic and real-world problems.

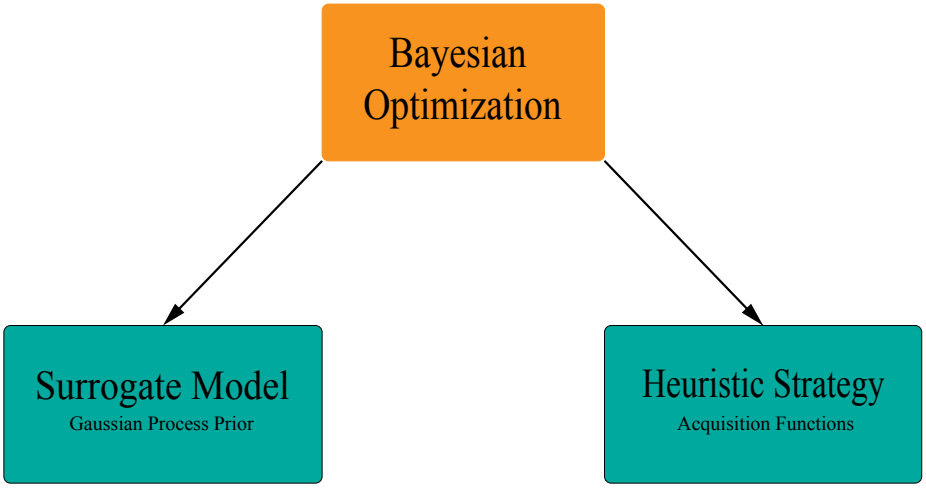

Figure 1.1: Key components of Bayesian optimization.

In Chapter 4, we use the proposed method on a real-world meta-problem of global learning rate adaptation for stochastic gradient descent. We use it to adaptively tune the global learning rate to improve the convergence of the stochastic gradient descent method regardless of the selected initial learning rate. We make further extensions to the model described in Chapter 3 to

have it incorporate and exploit derivative information of the meta-objective function. We also perform experimental analyses on learning models of varying complexity operating on widely-used standard datasets.

In Chapter 5, we delve into the online portfolio selection problem in quantitative finance where the existing techniques must adapt to new market conditions every time new data arrives. We use our method to propose a mechanism for adaptively configuring these methods in a manner that makes them acclimatize to the changing market conditions. We test our approach on algorithmic trading methods using widely-used underlying strategies operating on real-world equity and index data.

In Chapter 6, we interpret the significance of the proposed methods and their demonstrated usefulness while also critically examining the issues arising in the thesis. We also highlight potential avenues for future work. Chapter 7 provides the conclusions of the thesis.

2
Background Survey

If I have seen farther, it is by standing on the shoulders of giants.
— Isaac Newton, 1679

This thesis proposes useful extensions to Bayesian optimization to make it capable of dealing with objective functions that are time-varying. In this chapter, we go over some background material relating to Bayesian optimization and a discussion of the open and on-going problems in the field. This chapter also contextualizes the thesis within the scope of the extensions and open issues in the area of Bayesian optimization. The chapter is structured as follows. Section 2.1 discusses Bayesian optimization and Section 2.2 provides a brief overview of Gaussian processes. Section 2.3 discusses the acquisition functions used in Bayesian optimization. Section 2.4 discusses the open problems and extensions in Bayesian optimization. The chapter concludes with Section 2.5 which contextualizes the extensions to the topic of this thesis.

2.1 Bayesian Optimization

Let us assume that we want to solve the following optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}}\ f(\boldsymbol{x}), \tag{2.1}$$

where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^D$ is the decision variable, and $\mathcal{X}$ is a compact set. Let us also assume that $f(\boldsymbol{x})$ is latent, expensive and noisy.

Bayesian optimization (BO) is a sequential design strategy for optimizing objective functions that are unknown, noisy[1] and expensive to evaluate. We assume that there is a way of obtaining expensive samples from the objective function under consideration. This operation would usually involve taking a costly or time-consuming physical experiment or simulation.

Bayesian optimization works by using probabilistic tools to aid the search for the extremum. It operates in two significant phases. The first part uses a surrogate model to learn the latent objective function from available data samples. A popular surrogate model is a Gaussian process prior (Rasmussen & Williams 2006) that has the advantage of incorporating uncertainty about the objective function. The second phase involves finding a suitable point to evaluate the objective function. This search is performed using some predefined heuristics that intelligently and automatically *explore* and *exploit* the objective function with the goal of finding the optimum. In practice, this evaluation choice is achieved by performing a secondary optimization of a surrogate-dependent *acquisition function*, $a(\boldsymbol{x})$. After this evaluation input point is obtained, the objective function is evaluated at that point, and the process is repeated until the budgeted number of steps is exhausted. Algorithm 1 delineates this method.

[1] We mean measurement noise.

Algorithm 1 Bayesian Optimization with Gaussian Processes

1: **for** $\boldsymbol{i} = 1, 2, \ldots$ {Max iterations} **do**
2: Train Gaussian process model by hyperparameter optimization[2]
3: Calculate

$$\boldsymbol{x}_i = \arg\max_{\boldsymbol{x}} a(\boldsymbol{x})$$

4: Evaluate true objective function

$$y_i \leftarrow f(\boldsymbol{x}_i)$$

5: Augment new datapoint $\{\boldsymbol{x}, y_i\}$ to the dataset
6: Update current location of optimum
7: **end for**

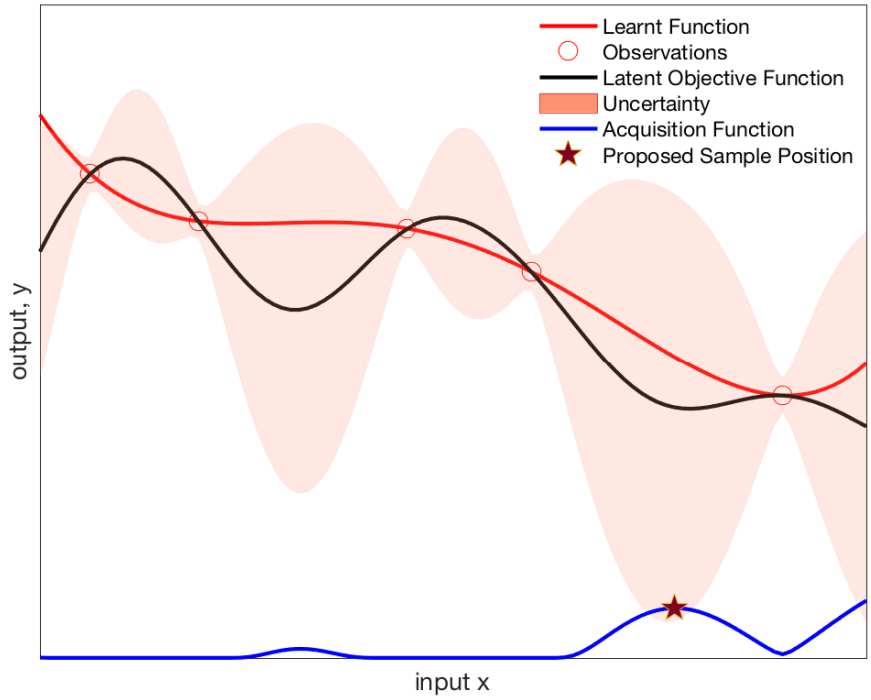

Figure 2.1: An illustration of Bayesian optimization on a one-dimensional objective function.

2.2 Gaussian Process Priors

Preliminaries

Regression is a type of prediction problem that is concerned with the prediction of continuous quantities. The available data is often in the form of input and target observations of the latent function, and the learning procedure uses this data to develop a model for prediction. This is called the *training data* because it is used to train the prediction model. We will assume we have an available dataset of the form

$$\mathcal{D} = \big\{ (\boldsymbol{x}_i, y_i) : i = 1, ..., n \big\},$$

where n is the number of datapoints, $\boldsymbol{x} \in \mathbb{R}^D$ is the input data variable, and $y \in \mathbb{R}$ is the output (or the response data variable) for the latent function f. We typically collate all the inputs $\boldsymbol{x}$ into a single matrix $\boldsymbol{X} \in \mathbb{R}^{n \times D}$, and the responses into a single vector $\boldsymbol{y} \in \mathbb{R}^n$. This allows us to denote the data as $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ for brevity, a notation we will use for the rest of this thesis.

A standard approach for regression is the *linear model*. In this model, the output is a linear combination of the input $\boldsymbol{x}$ and a weight parameter vector $\boldsymbol{w} \in \mathbb{R}^D$. This model has been well studied and extensively used in practice. This *parametric* model is of the form

$$y = f(\boldsymbol{x}) + \epsilon,$$

where

$$f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w} \quad \text{and} \quad \epsilon \sim \mathcal{N}(0, \sigma_{\mathrm{n}}^2).$$

The parameter $\boldsymbol{w}$ is a vector of weights for the model, and ϵ is an additive measurement noise that is independent and identically distributed as a zero-mean Gaussian with variance σ_{n}^2. The measurement noise captures the differences between the observed values and those predicted by $f(\boldsymbol{x})$.

One popular way of finding optimal weights $\boldsymbol{w}$ is through minimizing the mean squared error given by

$$\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^\top \boldsymbol{w})^2.$$

Since $\boldsymbol{w}$ is an unobserved quantity, we could treat it as a random variable with a prior probability distribution $p(\boldsymbol{w})$. For a dataset $\mathcal{D}$ and likelihood model $p(\mathcal{D} \mid \boldsymbol{w})$, the posterior distribution is given by

$$p(\boldsymbol{w} \mid \mathcal{D}) = \frac{p(\boldsymbol{w})\, p(\mathcal{D} \mid \boldsymbol{w})}{p(\mathcal{D})}. \tag{2.2}$$

This is called the *Bayesian linear model*. For the case described above, the likelihood model $p(\mathcal{D} \mid \boldsymbol{w})$ will be described in terms of the noise $\epsilon \sim \mathcal{N}(0, \sigma_{\text{n}}^2)$.

The linear model is heralded for its simplicity, interpretability and ease of implementation. However, it yields poor predictions for non-linear data and is thus not sufficiently flexible to handle a wide variety of latent functions that are found in many real-world applications.

A popular approach used to overcome the limited expressivity of linear models is to project the input data into a higher dimensional feature space using a set of basis functions Φ, and then apply the linear model in this new space rather than directly on the inputs. This projection mechanism results in a linear model that takes the form

$$f(\boldsymbol{x}) = \langle\, \boldsymbol{w}, \Phi(\boldsymbol{x}) \,\rangle, \tag{2.3}$$

where Φ is a fixed and finite-dimensional mapping, and $\langle \cdot, \cdot \rangle$ denotes the inner product. The solution to the model is calculated the same way as without the basis functions.

When using basis functions to perform this regression, it is observed that all the pairwise inner products between each datapoint in its basis function representation can be computed by evaluating a corresponding kernel function $k(\cdot, \cdot)$ to form a matrix K. This allows us to use a kernel $k(\cdot, \cdot)$ rather than the feature map Φ that can be hard to define in practice. This is called the *kernel*

trick. Kernels are more interpretable and allow the function f to be expressed in the form

$$f(\boldsymbol{x}) = \sum_{i}^{n} a_i \, k(\cdot, \boldsymbol{x}_i), \tag{2.4}$$

where $\{\boldsymbol{x}_i\}_{i=1}^{n}$ are the training points, $k(\cdot, \cdot)$ is a positive semi-definite kernel function and a_i is a quantity depending on both the data $\{\boldsymbol{x}_i\}_{i=1}^{n}$ and the kernel k. The Representer Theorem (Kimeldorf & Wahba 1971) guarantees that this is a sufficient representation for the function $f(\boldsymbol{x})$ for many practical cases. Since kernels are more interpretable than the feature vectors computed from basis functions, kernels become an object of crucial interest in practice, and the higher dimensional feature space becomes less important. Kernel-based regression techniques like this are called *non-parametric* methods because they involve models that consist of an infinite number parameters. Consequently, the learnt function f will be dictated by the specified kernel and the data used for training, rather than the values of the parameters $\boldsymbol{w}$.

Regression with Gaussian Process Priors

A Gaussian process (GP) is an extension of a multivariate Gaussian distribution to infinite dimensionality. It is a collection of random variables where any finite subset of those variables has a consistent joint Gaussian distribution. A Gaussian process describes a prior distribution over functions, and a mean and covariance function is utilized to specify it completely (Rasmussen & Williams 2006).

Let us consider the linear regression model previously shown in Equation 2.2. We can place a Gaussian process prior over the outputs $\boldsymbol{y}$, denoted by

$$p(\boldsymbol{y}) = \mathcal{N}(\, \boldsymbol{y} \mid m(\boldsymbol{x}, \boldsymbol{x}'), k(\boldsymbol{x}, \boldsymbol{x}')\,),$$

where $m(\boldsymbol{x}, \boldsymbol{x}')$ is the mean function, $k(\boldsymbol{x}, \boldsymbol{x}')$ the covariance function (also called the kernel), and $\boldsymbol{x}, \boldsymbol{x}' \in \boldsymbol{X}$, the input training data. It is common practice to use a zero-mean function since we can use pre-processed mean-adjusted versions of the target values $\boldsymbol{y}$.

For a Gaussian process prior

$$p(y) = \mathcal{N}(\, \boldsymbol{y} \mid \, \boldsymbol{0}, \, k(\boldsymbol{x}, \boldsymbol{x}')\,),$$

a dataset $\mathcal{D}$ and a test input $\boldsymbol{x}^\star \in \mathbb{R}^D$, we can obtain a posterior distribution for the associated target $y^\star \in \mathbb{R}$ which has the form

$$p(\, y^\star \mid \boldsymbol{x}^\star, \mathcal{D}\,, \boldsymbol{\theta}) = \mathcal{N}(\, y^\star \mid \, \mu(\boldsymbol{x}^\star)\,, \, \sigma^2(\boldsymbol{x}^\star)\,), \tag{2.5}$$

where $\boldsymbol{\theta}$ denotes collection of the hyperparameters of the mean and covariance functions. The posterior mean $\mu(\boldsymbol{x}^\star)$ and posterior marginal variance $\sigma^2(\boldsymbol{x}^\star)$ is given by

$$\mu(\boldsymbol{x}^\star) = k(\boldsymbol{x}^\star)^\top \, (\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I})^{-1} \, \boldsymbol{y}, \tag{2.6}$$

$$\sigma^2(\boldsymbol{x}^\star) = k(\boldsymbol{x}^\star, \boldsymbol{x}^\star) - k(\boldsymbol{x}^\star)^\top \, (\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I})^{-1} \, k(\boldsymbol{x}^\star). \tag{2.7}$$

where σ_{n}^2 is the noise variance from the additive measurement noise ϵ from the previously described linear model, $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $k(\boldsymbol{x}^\star)$ is a vector of covariance terms between $\boldsymbol{x}^\star$ and the data $\boldsymbol{X}$. The posterior mean and variance evaluated at any point $\boldsymbol{x}^\star$ represent the model's prediction and the associated uncertainty, respectively. Figure 2.2 shows an illustrative plot of the Gaussian posterior distribution on a toy one-dimensional regression problem.

Covariance Functions

Covariance functions[3] (or kernels) are a vital component of the Gaussian process inference machinery. They are used to encode our prior assumptions about the latent functions we wish to learn from training data. Kernels do this by describing the similarity between input datapoints. The rationale is that datapoints with inputs $\boldsymbol{x}$ which are close to each other are likely to have similar response values y. This observation implies that training points that are close

[3]The discussion in this section concerns stationary covariance functions. For more details, see Chapter 4 of Särkkä & Hartikainen (2012).

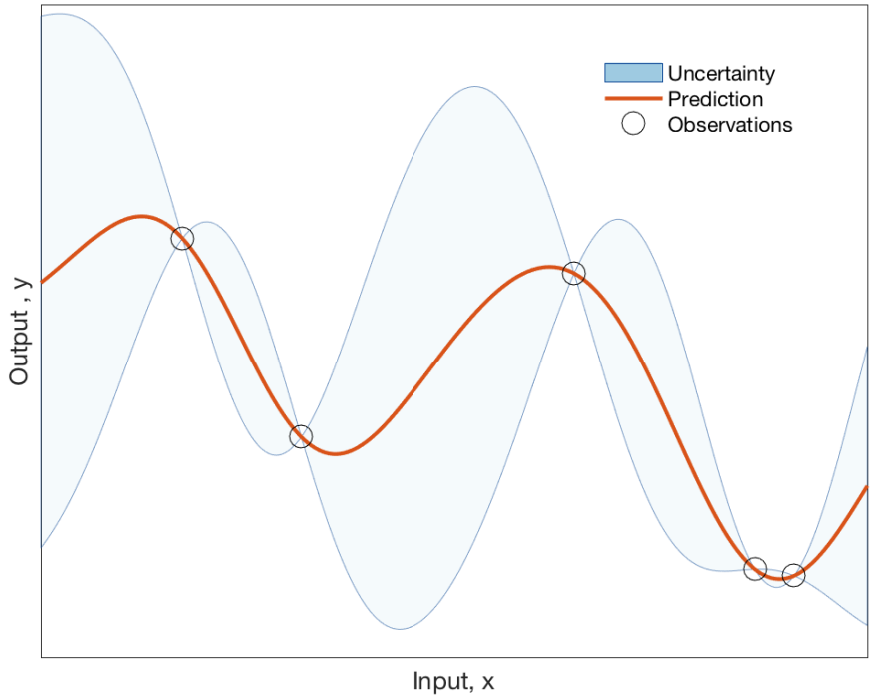

Figure 2.2: An illustration of Gaussian process regression on a one-dimensional example. The shaded uncertainty is the associated standard deviation of the prediction at that point.

to a test datapoint should be very informative about the prediction at that point.

Kernels are functions $k(\cdot,\cdot) \leq 1$ that map two inputs $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^D$ into $\mathbb{R}$ or, more generally, $\mathbb{C}$. Given a set of inputs $\{\boldsymbol{x}_i \mid i = 1, ..., n\}$, we can calculate the covariance matrix $K \in \mathbb{R}^{n \times n}$ whose entries are $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. One condition for Gaussian processes on K is that it should be positive semi-definite, i.e., it should satisfy

$$\boldsymbol{v}^\top K \boldsymbol{v} \geq 0, \qquad \forall \boldsymbol{v} \in \mathbb{R}^n.$$

A kernel which produces positive semi-definite covariance matrices is called a positive semi-definite covariance function.

Covariance functions can be used to describe the family of latent functions in terms of many properties such as their degree of differentiability or smoothness.

They often have additional hyperparameters that determine supplementary aspects of behaviour such as the characteristic input and output length-scales of the resulting family of latent functions.

Many of the widely-used covariance functions are functions of the difference $\boldsymbol{x} - \boldsymbol{x}'$. These class of covariance functions are called *stationary* because they describe the behaviour of the latent function that is invariant to translations in the input space. They are functions of the quantity describing the squared distance between datapoints that is given by

$$r^2 = \sum_{d=1}^{D} \left(\frac{x_d - x'_d}{l_d} \right)^2,$$

where $x_d \in \mathbb{R}$ is the value of $\boldsymbol{x}$ in dimension d, $l_d \in \mathbb{R}$ is the characteristic length-scale hyperparameter for dimension d.

Examples of commonly-used stationary covariance functions are described below:

- **Squared Exponential (SE).** This covariance function encodes smooth functions and is given by

$$k_{\mathrm{SE}}(r) = \sigma_f^2 \exp\left(-\frac{r^2}{2} \right) + \sigma_{\mathrm{n}}^2, \tag{2.8}$$

  where its hyperparameters are σ_f (the output scale) and σ_{n} (the noise term). Samples from a Gaussian process with this kernel are shown in Figure 2.3.

- **Matérn.** This class of covariance functions is described by

$$k_{\mathrm{Matérn}}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu r} \right)^{\nu} K_\nu \left(\sqrt{2\nu r} \right) + \sigma_{\mathrm{n}}^2, \tag{2.9}$$

  where parameter $\nu > 0$, $\Gamma(\cdot)$ is the Gamma function and $K_\nu(\cdot)$ is a modified Bassel function (Abramowitz & Stegun 1965, §9.6). The most interesting and commonly-used Matérn covariance functions are defined when the value of ν is a half-integer defined by $\nu = p + \frac{1}{2}$, where p is an

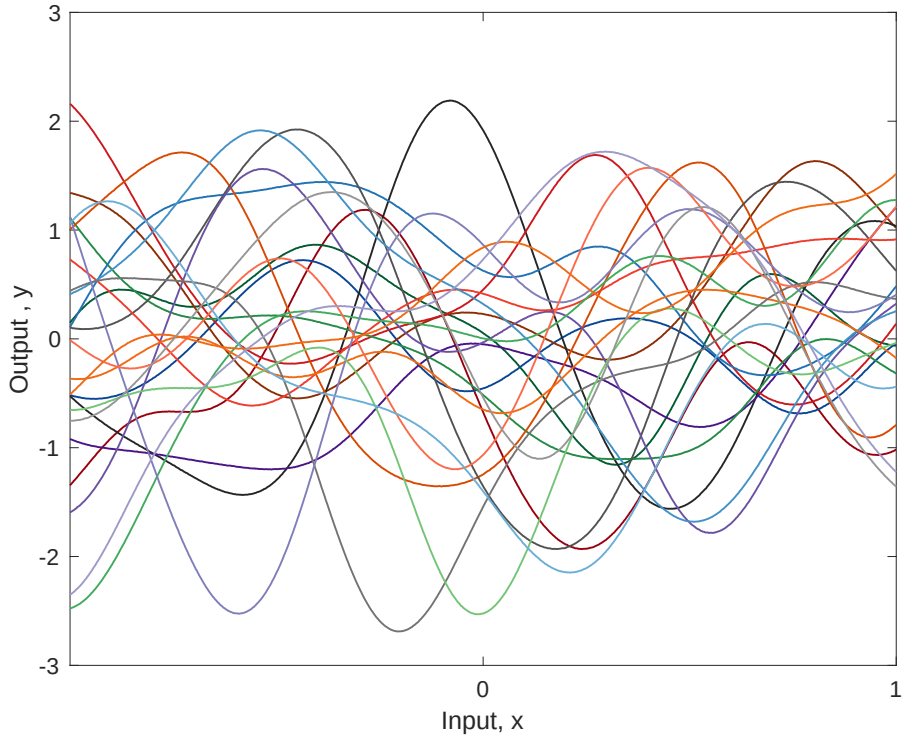

Figure 2.3: Samples from a Gaussian process with a squared exponential covariance function having a characteristic length-scale of $l = 0.25$.

integer. The specific kernels that are practically useful are defined when ν is $\nu = \frac{1}{2}$, $\nu = \frac{3}{2}$ and $\nu = \frac{5}{2}$, representing real Gaussian processes with decreasing levels of 'roughness'.

For $\nu = \frac{1}{2}$, the Matérn $\frac{1}{2}$ is given by

$$k_{\mathrm{M12}}(r) = \sigma_f^2 \exp(-r) + \sigma_{\mathrm{n}}^2. \tag{2.10}$$

Samples from a Gaussian process with this kernel are shown in Figure 2.4.

For $\nu = \frac{3}{2}$, the Matérn $\frac{3}{2}$ is given by

$$k_{\mathrm{M32}}(r) = \sigma_f^2 \left(1 + \sqrt{3}\, r\right) \exp(-\sqrt{3}\, r) + \sigma_{\mathrm{n}}^2, \tag{2.11}$$

Samples from a Gaussian process with this kernel are shown in Figure 2.5.

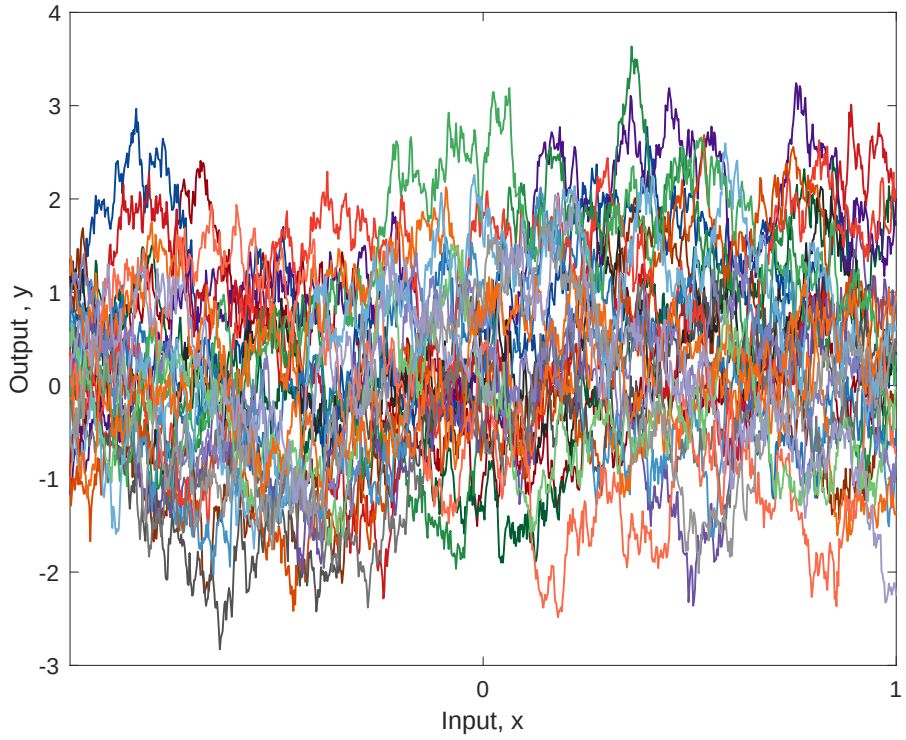

Figure 2.4: Samples from a Gaussian process with a Matérn $\frac{1}{2}$ covariance function having a characteristic length-scale of $l = 0.25$.

For $\nu = \frac{5}{2}$, the Matérn $\frac{5}{2}$ is given by

$$k_{\text{M52}}(r) = \sigma_f^2 \left(1 + \sqrt{5}r + \frac{5}{3}\, r^2\right) \exp(-\sqrt{5}\, r) + \sigma_{\text{n}}^2, \tag{2.12}$$

Samples from a Gaussian process with this kernel are shown in Figure 2.6. The hyperparameters of the Matérn kernels are σ_f (the output scale) and σ_{n} (the noise term).

- **Rational Quadratic (RQ).** This covariance function encodes a family of smooth functions with varying length-scales and is given by

$$k_{RQ}(r) = \sigma_f^2 \left(1 + \frac{r^2}{2\alpha}\right)^{-\alpha} + \sigma_{\text{n}}^2, \tag{2.13}$$

where σ_{n} is the noise term as before, σ_f is the output scale and $\alpha > 0$. The hyperparameter α determines the degree of mixing of the squared

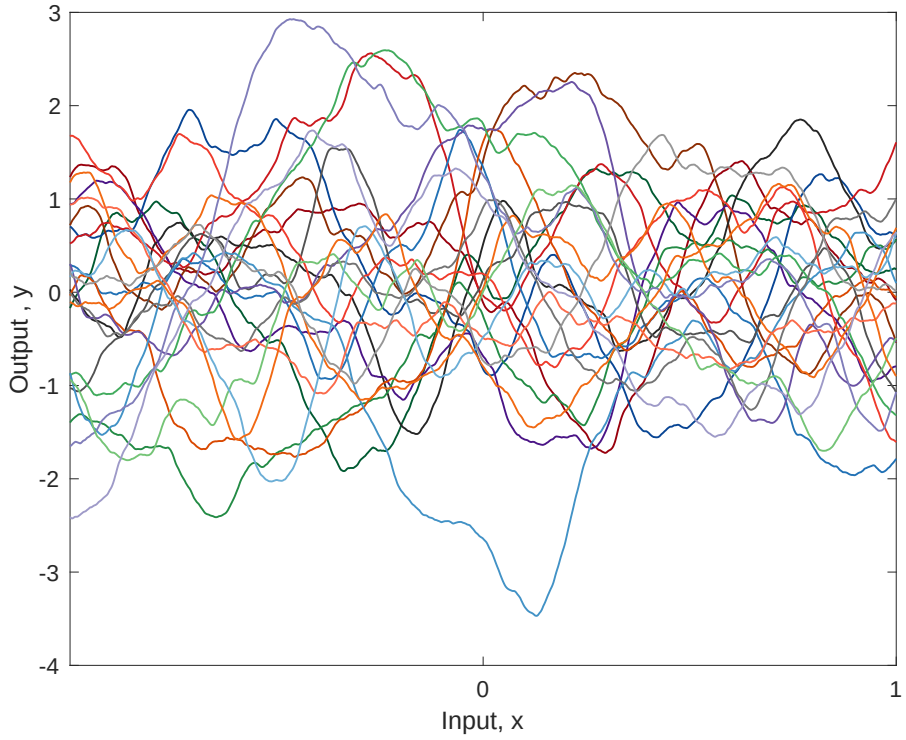

Figure 2.5: Samples from a Gaussian process with a Matérn $\frac{3}{2}$ covariance function having a characteristic length-scale of $l = 0.25$.

exponential kernels with varying length-scales. A panel showing samples from a Gaussian process with various values of α with this kernel are shown in Figure 2.7.

The hyperparameters of these covariance functions are often collated into a single variable $\boldsymbol{\theta}$. We briefly explore some ways of estimating $\boldsymbol{\theta}$ in the next subsection.

Model Selection

For a Gaussian process model to be valuable in practice, we need to make decisions about the specification of its hyperparameters. Since we are not usually certain about the suitable hyperparameters $\boldsymbol{\theta}$ to use, a common selection criterion is maximizing the log marginal likelihood $p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta})$ of the Gaussian process.

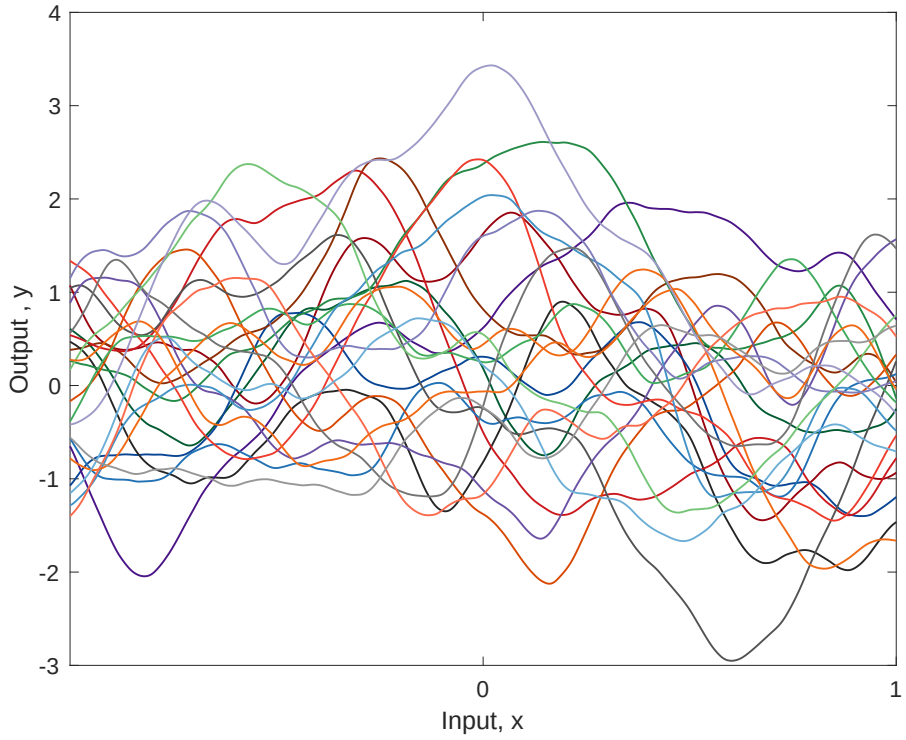

Figure 2.6: Samples from a Gaussian process with a Matérn $\frac{5}{2}$ covariance function having a characteristic length-scale of $l = 0.25$.

For a fully Bayesian treatment, we can assign hyper-priors to the hyper-parameters and subsequently marginalize over them using techniques such as Monte Carlo (Ghahramani & Rasmussen 2003, Snoek et al. 2012) or distributional inference methods (Minka 2001, Blei et al. 2017). For more details on model selection for Gaussian processes, see (Rasmussen & Williams 2006, §5.4).

Approximation Methods for Large Datasets

Gaussian process regression, as a Bayesian non-parametric method for regression, has its complexity dependent on the number of datapoints used for training the model. The standard implementation described in the previous subsections has a space complexity of $\mathcal{O}(n^2)$ resulting from the need to store the $n \times n$ covariance matrix. It also has a time complexity of $\mathcal{O}(n^3)$ resulting from

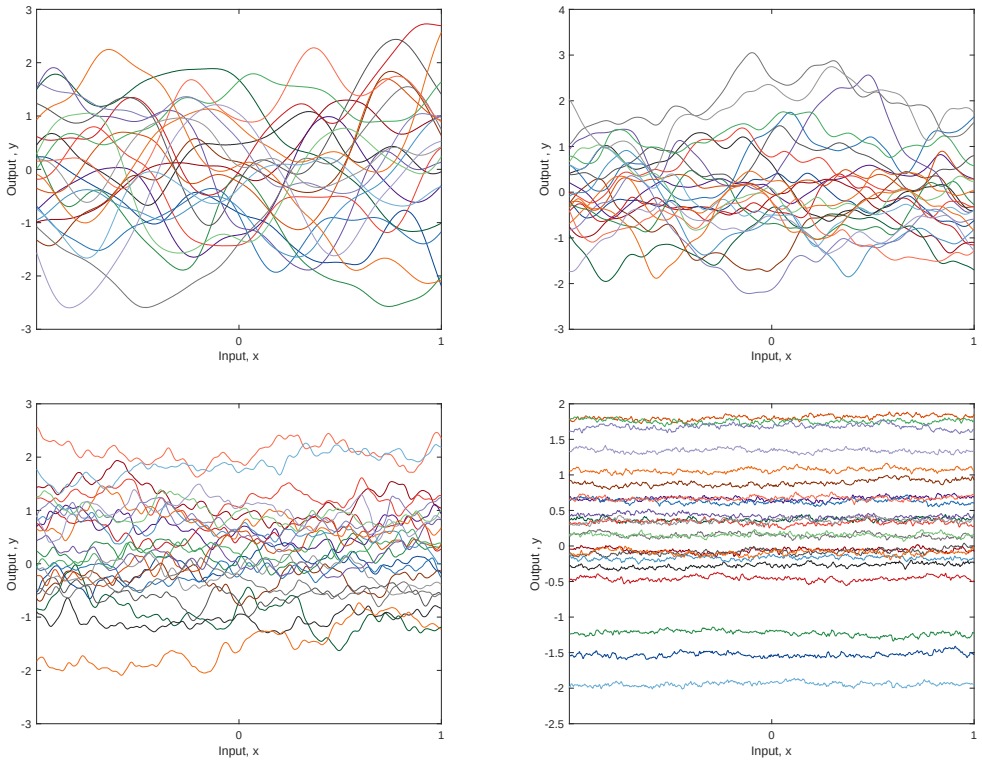

Figure 2.7: Samples from a Gaussian process with the rational quadratic covariance function. The top row shows samples drawn a from a rational quadratic kernel with $\alpha = 1$ and $\alpha = 0.1$, respectively. The bottom row shows samples drawn from a rational quadratic kernel with $\alpha = 0.01$ and $\alpha = 0.0001$, respectively. For all samples, the characteristic length-scale was $l = 0.25$.

solving the linear system found in the equations for the posterior distributions using n training datapoints—mainly due to the need to invert the covariance matrix. Additionally, the evaluation of the likelihood scales as $\mathcal{O}(n^2)$, and prediction scales as $\mathcal{O}(n^2)$ per forecast.

It is for this reason that approximations are required for cases with large amounts of training data. There has been a lot of work done on improving the scaling of Gaussian processes with n datapoints. Most of the proposed algorithms for these approximations are based on reducing the rank of the covariance matrix, rendering the complexity of the matrix inversion to $\mathcal{O}(m^3)$, where m is the new rank. This variable m is often the parameter of interest in these methods, and it usually represents a smaller amount of training data

that these approximations use instead of the full n datapoints, where $m << n$ (m is much smaller than n).

We will now briefly describe four common classes of Gaussian process approximation methods. For a more detailed treatment, refer to Chapter 8 in Rasmussen & Williams (2006), Chalupka et al. (2013) and Bui et al. (2017). The approximation methods are:

- **Subset of Data**. This method simply ignores part of the training data. The Subset of Data (SoD) method applies the full Gaussian process regression method to a subset of the training data with size m, where $m < n$. There are many ways that this subset may be selected, but there are two popular alternatives:

  1. Selecting the m points randomly, which costs $\mathcal{O}(m)$ if we do not look at the other points;
  2. Selecting the points informatively, such as using clusters as proposed by Gonzalez (1985), or other methods found in Herbrich et al. (2003) and Keerthi & Chu (2006).

- **Inducing Point Methods**. There are some methods that use an alternative kernel matrix that is built based on inducing points that live in the same D-dimensional space as the training data. Examples of algorithms using this approach include the fully independent training conditional (FITC) method (Quiñonero-Candela & Rasmussen 2005) and the Variational Free Energy (VFE) method (Titsias 2009), among many others.

- **Local Gaussian Process Regression**. This strategy divides-and-conquers the training set into clusters, each of size m, and runs Gaussian process regression on each cluster while ignoring the rest of the data. For a test point $\boldsymbol{x}^\star$ at test time, the test point is assigned to the closest cluster, and Gaussian process regression is performed.

- **Iterative Methods**. One intuitive way to speed-up Gaussian process regression is by recognizing that the linear system of interest

$$(K + \sigma_{\mathrm{n}}^2 I)^{-1} \boldsymbol{v} = y$$

can be solved by an iterative method like Conjugate Gradients (CG) (Golub & Van Loan 2012). If this algorithm runs for n iterations, it gives an exact solution, and gives approximate solutions if terminated earlier. So if it is terminated after m iterations, the resulting time complexity is $\mathcal{O}(mn^2)$. Conjugate Gradients can be further sped-up using approximate matrix-vector multiplications (Chalupka et al. 2013).

2.3 Acquisition Functions

In practice, the second phase of Bayesian optimization is performed by optimizing an acquisition function, $a(\boldsymbol{x})$. This function reflects the heuristics for the search of the optimum and is derived using information from the posterior distribution of the Gaussian process at an input point of interest. It is by this mechanism that BO sequentially selects query points. We describe examples of acquisition functions below. For all the examples, we will assume a posterior distribution over our objective function f to be

$$p(y^\star \mid \boldsymbol{x}^\star, \mathcal{D}, \boldsymbol{\theta}) = \mathcal{N}(y^\star; \mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x})), \tag{2.14}$$

where $\mu(\boldsymbol{x})$ and $\sigma^2(\boldsymbol{x})$ are the posterior mean and variance.

Expected Improvement

One popular and intuitive approach for searching for the minimum is to select points to evaluate on an objective function that will give us the most improvement over our current best point. Within Bayesian optimization, heuristics that do this are called improvement-based acquisition functions. One popular improvement-based acquisition function is called the expected improvement (EI). It is predicated on the notion of maximizing the amount of improvement

over the predefined threshold η, which characterizes the best-observed value of the objective function f. If we are searching for the minimum, this threshold η is the lowest value of the objective function that we have seen so far, i.e,

$$\eta \triangleq \min \boldsymbol{y},$$

where $\boldsymbol{y}$ is the collation of all the target training data we have seen so far.

Let us define an 'improvement' from previously gathered observations of f using a utility function given by

$$I(\boldsymbol{x}) = (f(\boldsymbol{x}) - \eta)\ \mathbb{I}(f(\boldsymbol{x}) < \eta), \tag{2.15}$$

where $\mathbb{I}$ is an indicator function. Consequently, $I(\boldsymbol{x}) > 0$ if there is an improvement. Because $f(\boldsymbol{x})$ is Gaussian, we can analytically characterize the expected improvement as

$$\begin{aligned} a_{\mathrm{EI}}(\boldsymbol{x} \mid \mathcal{D}, \boldsymbol{\theta}) &= \mathbb{E}\Big[I(\boldsymbol{x})\Big] \\ &= (\mu(\boldsymbol{x}) - \eta)\ \Phi\left(\frac{\mu(\boldsymbol{x}) - \eta}{\sigma(\boldsymbol{x})}\right) + \sigma(\boldsymbol{x})\ \mathcal{N}\left(\frac{\mu(\boldsymbol{x}) - \eta}{\sigma(\boldsymbol{x})}\right), \end{aligned} \tag{2.16}$$

where Φ is the standard Gaussian CDF[4], $\mathcal{N}$ is the standard Gaussian PDF[5], and η is the improvement threshold defined above.

The EI acquisition function measures the amount of improvement on the threshold η by considering both the posterior mean and the posterior standard deviation of the evaluation at that point. Consequently, it automatically balances how much improvement is gained from exploiting existing information about the objective function versus how much improvement is obtained from exploring the function. Some theoretical and convergence analyses of using this acquisition function with commonly-used kernels are found in Bull (2011).

[4]Cumulative distribution function.
[5]Probability density function.

Lower Confidence Bounds

An alternate strategy for searching for the minimum when uncertain of its location is to be optimistic in the face of this uncertainty. This approach is often implemented by using a fixed probability best case scenario as described by the model to negotiate exploration and exploitation. An example of an acquisition function that takes an optimistic approach to negotiate exploration and exploitation is the lower confidence bound (LCB) (Srinivas et al. 2012). It is a bandit-based search criterion that uses information from the Gaussian process posterior distribution to define confidence bounds that encode the best case scenarios. It is given by

$$a_{\text{LCB}}(\boldsymbol{x} \mid \mathcal{D}, \boldsymbol{\theta}) = \mu(\boldsymbol{x}) - \kappa \cdot \sigma(\boldsymbol{x}), \tag{2.17}$$

where $\kappa > 0$ is a *trade-off* parameter. This parameter is set to achieve optimal regret (Srinivas et al. 2012) by using the following choice

$$\kappa = \sqrt{2 \log \Big(\frac{\pi^2 \, n^{\frac{D}{2}+2}}{3\delta} \Big)}, \tag{2.18}$$

where n is the number of past evaluations of the objective function f, $\delta \in (0, 1)$ is a heuristic parameter, and D the dimensionality of the latent function. It has been shown in experiments that better performance is achieved if this parameter is scaled down by a fifth (Srinivas et al. 2012, Calandra et al. 2016). The lower confidence bounds acquisition function is also called the upper confidence bounds (UCB) for cases involving a maximization optimization problem.

A significant upside of using this acquisition function is that there exist theoretical analyses that describe the conditions when it works in terms of the notion of *regret*, where regret is the difference between the best possible and ideal function value returned by the algorithm (the maximum or minimum) minus the true optimum value. See the work in Srinivas et al. (2012) for detailed theoretical analyses.

Minimum Mean

The minimum mean (MM) heuristic proposes a point with the lowest expected value as estimated by the surrogate model with respect to a minimization problem. It is given by

$$a_{\mathrm{MM}}(\boldsymbol{x} \mid \mathcal{D}, \boldsymbol{\theta}) = \mu(\boldsymbol{x}). \tag{2.19}$$

The main advantage of this acquisition function is that if the model used is stable and has been learnt very well, it gives superior performance. Additionally, it does not have any parameter that needs to be tuned. However, it is likely to operate greedily and propose solutions that fall in local minima and just exploit the learnt model.

Entropy Search

Unlike the previous acquisition functions, there is another class of heuristics which depend on the posterior distribution over the unknown minimizer $x^\star$, denoted as $p^\star(\boldsymbol{x} \mid \mathcal{D})$. This distribution is indirectly induced by the posterior distribution over an objective function f. Entropy search (ES) techniques aim to reduce the uncertainty in the location $x^\star$ by choosing the point that is expected to cause the greatest reduction in entropy of the distribution $p^\star(\boldsymbol{x} \mid \mathcal{D})$ (Hennig & Schuler 2012), and is given by

$$a_{\mathrm{ES}}(\boldsymbol{x} \mid \mathcal{D}, \boldsymbol{\theta}) = H(\boldsymbol{x}^\star \mid \mathcal{D}) - \mathbb{E}_{y|\mathcal{D},\boldsymbol{x}}\, H(\boldsymbol{x}^\star \mid \mathcal{D} \cup \{\boldsymbol{x}, y\}), \tag{2.20}$$

where $H(\boldsymbol{x}^\star \mid \mathcal{D})$ is the differential entropy of the posterior distribution. ES measures the expected gain of information about the unknown $\boldsymbol{x}^\star$ obtained by evaluating the objective function at location $\boldsymbol{x}$. It aims to select the point that provides the most information about $\boldsymbol{x}^\star$. The function in equation 2.20 is not tractable for continuous search spaces so approximations are usually made, one such example is predictive entropy search (PES) (Hernández-Lobato et al. 2014) heuristic which is given by

$$a_{\mathrm{PES}}(\boldsymbol{x} \mid \mathcal{D}, \boldsymbol{\theta}) = H(y \mid \mathcal{D}, \boldsymbol{x}) - \mathbb{E}_{\boldsymbol{x}^\star|\mathcal{D}}\, H(y \mid \mathcal{D}, \boldsymbol{x}, \boldsymbol{x}^\star), \tag{2.21}$$

where the expectation is estimated using Monte Carlo sampling techniques.

2.4 Extensions and Open Questions in Bayesian Optimization

This section explores some extensions and open questions in the area of Bayesian optimization.

- **Multi-task.** Many tasks for which Bayesian optimization is applied to tend to be related to other tasks. For example, if Bayesian optimization is used to tune the hyperparameters of an object recognition machine learning model on some data, this model is likely to be good on some other unseen object recognition datasets. This motivates the idea that the hyperparameter configuration problem is only one type of task among many related hyperparameter configuration tasks operating on different object recognition datasets. Consequently, the question becomes whether it is possible for these tasks to share information about each other. There have been attempts to exploit this property within the auspices of Bayesian optimization (Krause & Ong 2011, Hutter et al. 2011, Swersky et al. 2013, Bardenet et al. 2013, Yogatama & Mann 2014, Feurer et al. 2015). It is assumed that these tasks are defined by a set of correlated functions where we are interested in optimizing a subset of them. One way to share information between tasks in BO is to modify the underlying GP model to involve multiple outputs. These GP models include a valid covariance over both the input and task pairs (Goovaerts 1997, Seeger et al. 2005, Bornn et al. 2012). Additionally, some of the tasks may have some additional side information or some context features that can be used in a joint model (Hutter et al. 2011, Feurer et al. 2015).

- **Partial Feedback.** For some cases, the experiments proposed by Bayesian optimization contain an inner loop of iterative and time-consuming optimization. For example, in the problem of tuning a machine learning

algorithm, an experiment involves tuning the model before evaluating it. It is sometimes possible to evaluate how good the model is during the training stage without having to subsequently evaluate the model. This is inspired by practices by expert human hand-tuners in some domains. For hand-tuned models, experts often halt experiments which show little promise early to save time, and this allows them to train more models in a fixed amount of time. Attempts to incorporate this idea of partial feedback into Bayesian optimization was done in Swersky et al. (2014) where they halted (and restarted previous experiments) based on the forecasts from the learnt Gaussian process model.

- **High-Dimensionality.** For Bayesian optimization, to ensure that the global optimum is found, there is a need for good coverage of the search domain. However, as the dimensionality increases, the number of evaluations needed to cover the space also increase exponentially (Shahriari et al. 2015). Additionally, for BO algorithms that use standard Gaussian process models for inference, there are some scalability issues with large datasets. Consequently, the standard form of Bayesian optimization is restricted to problems of moderate dimension. There has been some work done to address this. One approach proposed in (Carpentier & Munos 2012, Chen et al. 2012) uses a two-stage process that first identifies the relevant dimensions using some tests and then performs the optimization. Another approach used in Hutter et al. (2011) uses random forests with their frequentist uncertainty estimates where the random forests naturally select the relevant dimensions. One key idea that is spearheading the research in this area is that of *effective dimensionality*. This is the notion that for some problems[6] most of the dimensions do not significantly change the objective function (Shahriari et al. 2015). Therefore,

[6]Problems such as hyperparameter optimization for neural networks and deep belief networks (Bergstra & Bengio 2012) and algorithm configuration for NP-hard problems (Hutter et al. 2013).

there exists a small number of dimensions that are more informative about changes in the objective function. These ideas have been used in Bergstra & Bengio (2012) and Wang et al. (2013), among others.

- **Cost Sensitivity.** There some applications where the evaluation of the objective functions also returns the cost associated with that evaluation. And in these applications, there may be different costs associated with evaluating different parts of the design space. In Snoek et al. (2012), they model both the objective and cost function separately using two independent Gaussian processes and perform the selection of points to evaluate using a cost-adjusted acquisition function that considers both models.

- **Constrained Bayesian Optimization.** There are optimization problems where particular regions of the design space are invalid and hence out of bounds. For example, for algorithm configuration problems, some configurations can result in models that run out of memory. It, therefore, becomes imperative to search the space in a manner that avoids the invalid inputs. When these constraints are known *apriori*, then they can be easily incorporated into the auxiliary optimization of the acquisition function. The challenge, however, is when these constraints are unknown, i.e., we do not know in advance which configurations will result in unwanted behaviour that defines the constraint violation. Several methods deal with this problem by modifying the acquisition function to include some probabilistic notion of constraint satisfaction (Snoek 2013, Gelbart et al. 2014, Gardner et al. 2014, Hernández-Lobato et al. 2014, Gramacy et al. 2016). For more details, see Section 8.1 of Shahriari et al. (2015).

- **Dynamic functions.** The idea of extending Bayesian optimization to functions that are changing in time is motivated by several examples. In Osborne et al. (2009), they suggest the future possibility of using BO to

select when to evaluate a function of time in order to maximize it, such as determining the most profitable future time to make a costly financial trade. Alternatively, one could wish to optimize a dynamic function where only one observation can be chosen at every time step. These scenarios motivate the extension of Bayesian optimization to time-varying scenarios where concept-drift occurs. This thesis studies a particular aspect of this problem where the goal is not only to optimize a dynamic function, where only one observation can be made per time step, but to also keep track of it over a fixed horizon of steps. We briefly discuss some aspects of this problem in the next section.

2.5 Dynamic Optimization Problems

This thesis studies the problem of using Bayesian optimization to find and keep track of a minimum of a dynamic function over a fixed horizon of time steps. We assume that the objective function can only be evaluated once per time step and that the dynamic function is unknown, expensive to evaluate and has some measurement noise. We also assume that the process of evaluating this objective function does not modify or change it, or that the evaluations are not the cause of the concept drift. We assume that the temporal evolution of the objective function is an inherent property of the function, i.e., the objective function has no state.

The motivation for solving this problem is the need to adaptively tune parameters of an algorithm whose optimal parameters are evolving over time. This temporal evolution often happens when the algorithm being configured is sequentially processing non-stationary data and the best parameters evolve as a consequence. We assume that the optimal parameters in question are dependent on the data the underlying algorithm is trained on and, as a result, it is reasonable to claim that non-stationary data will necessitate the need to adaptively update the parameters.

This problem is amenable to Bayesian optimization for several reasons. First, the dynamic objective function that characterizes this temporal evolution in the optimal parameters is latent as we normally would not have access to analytical descriptions of it or its derivatives. Secondly, the evaluation of the objective function will often involve an expensive and time-consuming process of running an experiment or algorithm to obtain the output of the evaluation, where the output may have some measurement noise. Third, algorithm configuration is the most common uses of Bayesian optimization algorithms, and extending BO to the adaptive case only increases the repertoire of scenarios for which it can be used.

In the standard case, BO uses sequential evaluations of the objective function to search for the minimum of the objective function. These sequential evaluations use information of the learnt model to explore and exploit the objective function to aid this search. At the end of the sequence of evaluations, the best-evaluated point is selected as the minimum. In the problem we are solving in this thesis, this approach presents several problems. First, we are not only looking to find the minimum of the dynamic function but also want to keep track of it as it evolves as we perform our evaluations. Secondly, and as a result of the first problem, the timely evaluations of the objective function should be both informative about the function and its temporal evolution while also ensuring that the evaluations are as close as possible to the location of the *current* minimum at that point in time.

These requirements present several challenges. First, given that we assume that the dynamics of the objective function are inherent to it, what is the best way to model the objective function using a Gaussian process prior? How can we best make use of the flexibility of Gaussian processes to easily model the time-varying objective functions? Second, if we have a model for dynamic objective function, how do we encode the need to find and keep track of the minimum through the acquisition function? How do we ensure that what we do still maintains the flexibility afforded by Bayesian optimization? Third,

since Bayesian optimization operates by sequentially trading off exploring and exploiting the objective function, some of the evaluations in the sequence are likely to lead to less optimal values and poor evaluations from the tracking perspective (i.e., may be far from the current minimum at that point in time). We propose extensions to Bayesian optimization that address these issues and demonstrate their efficacy on a wide variety of problems in the rest of this thesis.

2.6 Summary

This chapter provided a brief overview of the topics discussed and applied in this thesis. It covered Bayesian optimization and Gaussian process regression, and contextualized them to the problems addressed in this thesis.

3

Bayesian Optimization for Dynamic Problems

To improve is to change, so to be perfect is to change often.
— Winston Churchill, 1922

This chapter studies the problem of tracking the minimum of a latent, noisy[1] and expensive dynamic function given a finite horizon of steps in which to make evaluations. We propose practical extensions to Bayesian optimization for solving this problem. We model the dynamic objective function using a spatiotemporal Gaussian process prior which captures all the instances of the function over time. Our practical extensions to Bayesian optimization use the information learnt from this model to guide the tracking of the temporally evolving minimum by automatically adjusting the feasible search region at every time-step using insights gained from the learnt model. By exploiting the temporal correlations of the model, our proposed method determines when to make evaluations, how fast to make those evaluations, and it induces an appropriate budget of steps based on the available information. We also show

[1] Measurement noise.

that the resulting method naturally produces a mechanism by which model approximation can be performed when the number of data samples becomes too expensive for inference. Lastly, we evaluate our technique on synthetic and real-world problems to comparatively demonstrate the tracking behaviour.

3.1 Introduction

In many practical situations, it is difficult to escape global optimization problems. They arise from our need to make optimal decisions. Global optimization problems are characterized by the search for the global extremum of an objective function $f(\boldsymbol{x})$. We define a global optimization problem as:

$$\underset{\boldsymbol{x}}{\text{minimize}}\ f(\boldsymbol{x}), \tag{3.1}$$

where $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^D$ is the decision variable, and $\mathcal{X}$ is a compact set.

Global optimization problems arise in a wide range of applications in computer sciences, economics, engineering, medical sciences, operations research, and statistics (Luenberger et al. 1984). They have been studied extensively, and a wide variety of methods exist for solving them (Archetti & Schoen 1984, Horst et al. 2000). Most global optimization methods require the analytic expressions and derivative information of the objective function for them to search for the optimum (Torn & Zilinskas 1989).

However, there is a class of global optimization problems where the objective function is unknown. In these cases, we have no access to the function's analytic form or its derivatives, and it is usually costly to evaluate, where evaluating the function corresponds to making an expensive simulation or physical experiment. This expense may concern time, effort, money, or any other relevant criterion. Bayesian optimization (BO) (Mockus 1975) is a sequential design approach for solving this type of problem. It models the latent objective function using a surrogate model and utilizes an auxiliary surrogate-dependent optimization to determine where to sequentially evaluate the objective function in a manner that explores and exploits it to find the optimum. A popular

surrogate model is a Gaussian process (GP) prior because of its ability to handle uncertainty about the latent objective function. Bayesian optimization has been extensively used in applications such as robotics (Lizotte et al. 2007), algorithm configuration (Brochu et al. 2010), sensor selection (Osborne et al. 2009, Garnett et al. 2010), adaptive Markov chain Monte Carlo (MCMC) (Mahendran et al. 2012), localization (Carpin et al. 2015), probabilistic programming (Rainforth et al. 2016), powered prosthetics (Kim et al. 2017), crystal structure prediction (Yamashita et al. 2018) and neuroimaging (Lancaster et al. 2018).

There is also a related class of more complex problems to consider. There are scenarios where the latent objective function is evolving with time. Such complex problems arise naturally in real-world situations that involve nonstationary environments. In these problems, elements of the underlying model change during the course of optimization. Since we operate by evaluating the model sequentially, where we are allowed one evaluation at a time, these changes in the underlying model mean that we encounter a slightly different problem with each evaluation, where all the different problems we encounter are related. Our goal is, therefore, to make sure that each evaluation is as close as possible to the *current* minimum at that point in time.

An example of such a problem is the finding and tracking of the location with the lowest temperature in a building where this location changes as we perform an optimization to locate it. The objective function representing the temperature throughout this building is unknown and for us to sequentially evaluate it, we have to take measurements with a thermometer at particular locations in the building, which is a time consuming process. The temperature measurements that we take can also be associated with some measurement noise. We assume that the location of the lowest temperature changes as we sequentially evaluate the objective function but the cause of these changes is extrinsic to the evaluations. In this example, our goal would be make evaluations of the function such that each evaluation gets us as close as possible to the location of the minimum temperature at that point in time.

This example motivates the need to study time-varying optimization problems of this nature. Doing so allows us to develop methods that can find and keep track of a temporally evolving optimum. We call this a dynamic optimization problem (DOP) (Cruz et al. 2011), and we borrow from the evolutionary optimization literature and define it as follows:

$$\text{DOP} = \left\{ \begin{array}{ll} \text{minimize} & f(\boldsymbol{x}, t) \\ \text{s.t.} & \{\boldsymbol{x}, t\} \in F(t) \subseteq \mathcal{S},\ t \in \mathcal{T} \end{array} \right\} \tag{3.2}$$

where:

- $\mathcal{S} \in \mathbb{R}^D$, $\mathcal{S}$ is the search space.
- $t \in \mathbb{R}$ is the time.
- $f : \mathcal{S} \times \mathcal{T} \mapsto \mathbb{R}$ is the objective function that assigns a numerical value ($f(\boldsymbol{x}, t) \in \mathbb{R}$) to each possible solution ($\boldsymbol{x} \in \mathcal{S}$) at time t.
- $F(t)$ is the set of all feasible solutions $\{\boldsymbol{x}, t\} \in F(t) \subseteq \mathcal{S}$ at time t.

The goal of solving a DOP is to find and efficiently keep track of a minimum's movement through the solution space. We assume that the temporal evolution is an intrinsic property of the DOP and not as a result of the function being evaluated, so an optimization algorithm does not modify it through evaluations. In Equation 3.2, $f(\boldsymbol{x}, t)$ is a spatiotemporal model where t denotes the times at which the sequential evaluations with a spatial input $\boldsymbol{x}$ are performed. The goal of any solution method for this problem is to find the global minimum and keep track of it without having to restart the optimization process after any temporal changes. In Equation 3.2, the notion of the temporal evolution is encoded using the constraints describing the feasible region for the evaluation at time t, denoted as $F(t)$. This region corresponds to the part of the spatiotemporal model where the time t applies. This means that different times will correspond to evaluations at distinct regions of $f(\boldsymbol{x}, t)$. An ideal solution is one that tracks the minimum closely, on *average*, within a fixed and finite

horizon of iterations for which we wish to perform function evaluations. Since we assume that the dynamic optimization problem is also latent, noisy and expensive, where we only observe a few samples of the function, this tracking must also be performed in a way that is data-efficient and makes maximal use of the available prior information. All this tracking must be accomplished using the fewest possible function evaluations.

We start by modelling $f(\boldsymbol{x}, t)$ using a spatiotemporal Gaussian process prior and using the learnt insights to include additional heuristics in the acquisition function search. In addition to looking for what spatial input $\boldsymbol{x}$ at specified times t, our heuristics can use the learnt insights from the Gaussian process to flexibly look a short distance into the future to also determine when to make evaluations. Consequently, our proposed algorithm can find and keep track of the minimum in two modes. The first makes evaluations of the objective function at predefined times, as described in the standard problem. The second way can also determine when to make these valuations using the information learnt by the Gaussian process model.

In this chapter, we propose several practical extensions of Bayesian optimization for solving this type of dynamic optimization problem. The challenge of using Bayesian optimization for the task we have described is compounded by the fact that BO is a sequential evaluation algorithm that explores and exploits the objective function. Since our goal is to make evaluations of the time-varying model which are as close as possible to the minimum at that point, the potentially suboptimal exploration steps from BO can impede the tracking performance of our algorithm.

This chapter is structured as follows. In Section 3.2, we explore related work. Section 3.3 describes the Gaussian process prior model for dynamic optimization problems and then Section 3.4 explains the various practical extensions to the standard Bayesian optimization algorithm that make it suitable for dynamic problems. Section 3.5 delineates why the proposed algorithm works. In Section 3.6, we delineate a subset-of-data approximation method for

time-varying regression based on the insights learnt from our proposed method. Section 3.7 contains experimental analyses and discussions. For more details on Bayesian optimization, see Section 2.1 and Shahriari et al. (2015), and for more details on Gaussian processes, see Section 2.2 and Rasmussen & Williams (2006).

3.2 Related Work

In this section, we explore related work. The contents of the section are categorized by the types of the related methods used to tackle dynamic optimization problems.

Population-based Optimization Algorithms

Nature-inspired population-based optimization algorithms were the first kind of solution methods used for dynamic optimization problems. This is because of the natural coverage over the search domain offered by their function evaluation schemes. These methods can easily be extended to dynamic scenarios by managing the diversity of their function evaluations in the search space. This diversity allows the algorithms to be reactive to changes in the objective function. Examples of such algorithms include evolutionary algorithms (EAs) (Cruz et al. 2011), particle swarm optimization (PSOs) (Pelta et al. 2009), ant colony optimization (ACOs) (Trojanowski & Wierzchoń 2009), and immune-based methods (Guntsch et al. 2001), among others. These methods have mostly been tested on synthetic dynamic problems. Many test benchmarks have been created, most notably the moving peaks benchmark (Branke 1999), which is the most widely used. This benchmark has controllable complexity and degree of dynamism. More recently, there have been some applications to real-world dynamic optimization problems in aerospace design (Mack et al. 2007), risk in financial optimization problems (Tezuka et al. 2007), path planning for ships and pollution control (Michalewicz et al. 2007).

Population-based methods assume that the objective function f is cheap to evaluate and sometimes known. As a result, many function evaluations are required for good performance which may not be feasible in many real-world applications. Furthermore, it is often not clear how much evaluation location *diversity* and associated *reactivity* is required for the algorithms to work well for dynamic optimization problems. This is because too much of it might resemble restarting the optimization process and too little of it may not track the optimum at all.

Using Bayesian optimization is a more efficient solution method because it addresses these issues. First, a Gaussian process surrogate model captures a family of objective functions as described by prior beliefs and verified by observed data, therefore offering a much more powerful description of the objective function as it describes how uncertain it is about some regions of the search space. Second, the Bayesian updates from function evaluations capture all changes in the objective function. Third, Bayesian optimization is built for expensive and unknown objective functions and can provide excellent solutions with few evaluations.

Bayesian Optimization Methods for Dynamic Problems

While there have been many advances in the area of Bayesian optimization, from increasing the number and type of acquisition functions to some convergence proofs (Bull 2011, Srinivas et al. 2012), it is only recently that some work has tackled time-varying problems.

In Bogunovic et al. (2016), they propose a sequential Bayesian optimization algorithm with bandit feedback that allows for the reward function to vary with time. They do this by modelling the reward function with a Gaussian process prior whose evolution follows a simple Markov model. They identify that the main problem with using standard Bayesian optimization with bandits on this problem is that BO treats old and new data equally, which is not optimal given that the reward function is temporally evolving. The rationale

is that older information may be stale and therefore misleading. Their solution method recognizes and smoothly discards stale data samples of the time-varying reward function via posterior updates, and this is the mechanism by which their method adapts to changes in the reward function. Their algorithm outperforms classical Bayesian optimization using upper confidence bounds (Srinivas et al. 2012).

The problem with their formulation is that there are many real-world dynamic problems whose evolution follows more complex behaviour than that described by a simple Markov model. For example, their model cannot capture smoother temporal evolution behaviour occurring over a longer horizon of time-steps, a scenario that arises in practice. So to deal with many practical problems, there must be flexibility in the ability to define and encode appropriate prior beliefs about the temporal evolution.

Additionally, in their model, due to the simple Markov nature, time is defined as being from an index set delimiting the instances when their algorithm evaluates the true reward function. This definition is very simplistic and unrealistic for many tasks. In many real-world applications, there is an independent notion of time that is invariant to the number of times the algorithm interacts with the objective function. Our proposed algorithm uses this independent time characterization and has the benefit of allowing data from different sources to be fused as long as the same measure of time is used, and also as long as the data is sampled from the same latent function.

However, a significant advantage of the work by Bogunovic et al. (2016) is that they provide theoretical guarantees for time-varying bandits, where they explicitly characterize the regret bounds in terms of their dependence on the function's rate of variation. They show that using spatiotemporal formulations of the time-varying objective function leads to a cumulative regret whose lower bound is $\Omega(T)$ (Bogunovic et al. 2016, Theorem 4.1). They further characterize this regret's expectation via a joint dependence between the number of iterations T and a *forgetting factor* hyperparameter $\epsilon \in [0, 1]$, which describes the

degree to which the objective function varies. When $\epsilon = 0$, the function does not change, and when $\epsilon = 1$, the function changes so quickly that the instances over time are independent. They also show that this regret's upper bound is of the form $\mathcal{O}(T\psi(\epsilon))$, where ψ is a function that vanishes as $\epsilon \to 0$.

For the spatial squared exponential and Matérn kernels, they obtain regret bounds of the form $\mathcal{O}(T\epsilon^{\alpha})$, where $\alpha > 0$, which gives the sought-after sub-linear regret whenever $\epsilon = \mathcal{O}(T^{c})$, for some $c > 0$. Specifically, the regret bound for the squared exponential kernel is $\mathcal{O}(\max(\sqrt{T}, T\epsilon^{\frac{1}{6}}))$, and the regret bound for the Matérn class of kernels is $\mathcal{O}(\max(\sqrt{T^{1+c}}, T\epsilon^{\frac{1}{2}\frac{1-c}{3-c}}))$, for $\nu > 2$ and $c = \frac{D(D+1)}{2\nu+D(D+1)}$, where D is the dimensionality of the problem (Bogunovic et al. 2016, Corollary 4.1). They also state that regret bounds of the form $\mathcal{O}(T\psi(\epsilon))$ will result from other temporal kernels that depend on the temporal difference $|t-t'|$. They add that their choice of a simplistic temporal model that induced the simple Markov temporal evolution was done for the purpose of concreteness in presenting their proofs.

The approach used in Bogunovic et al. (2016) to obtain their regret bounds is analogous to earlier work on *Brownian restless bandits* by Slivkins & Upfal (2008). The earlier analyses demonstrated that a regret lower bound of $\Omega(T)$ is unavoidable for a time-varying objective function. The theoretical analysis in Bogunovic et al. (2016), which gives an upper regret bound of the form $\mathcal{O}(T\epsilon^{\alpha})$, makes intuitive sense because we would expect that we can only track the evolving function if we make evaluations of the function fast enough to learn any temporal correlations. For more details on time-varying Gaussian process bandit optimization, see Bogunovic et al. (2016) and the next section.

Background on Time-varying Gaussian Process Bandit Optimization

In this subsection, we describe a bandit-based Bayesian optimization algorithm that extends the work on Gaussian process bandit optimization with upper confidence bounds (Srinivas et al. 2012), often called GP-UCB, to deal with time-

varying objective functions. Time-varying functions are a topic of interest in this thesis. The time-varying Gaussian process bandit optimization algorithm (TVB) was proposed in Bogunovic et al. (2016) where they also provide regret bounds for the time-varying case [2].

The GP-UCB algorithm is designed to achieve sublinear asymptotic regret bounds, a seminal result in bandit-based Bayesian optimization literature. The work in Bogunovic et al. (2016) extends GP-UCB to allow the objective function to vary with time by modelling it using a Gaussian process prior whose evolution obeys a simple Markov model. If we let $g_1, g_2, ..., g_n$ be independent random functions defined on $\mathbb{R}^D$ and are drawn from a zero-mean Gaussian process $g_i \sim \mathcal{GP}(0, K)$, then the objective functions are modelled as follows:

$$f_1(x) = g_1(x) \tag{3.3}$$

$$f_{t+1}(x) = \sqrt{1-\epsilon}\, f_t(x) + \sqrt{\epsilon}\, g_{t+1}(x), \quad \forall t \geq 2, \tag{3.4}$$

where $\epsilon \in [0, 1]$ quantifies how much the function changes after every time-step. If $\epsilon = 0$, we recover the standard time-invariant model of the objective function as in GP-UCB. For $\epsilon = 1$, the other extreme, the function changes very quickly and its instances become independent between time-steps. For any choice of ϵ, there exists some $g_t \sim \mathcal{GP}(0, K)$ for all t.

The practical advantage of this formulation is that it only involves one additional hyperparameter ϵ that can be learnt from data. This model is equivalent to using a spatiotemporal Gaussian process model with a temporal kernel given by

$$(1-\epsilon)^{\frac{(t-t')}{2}},$$

[2]Regret is a frequentist measure of performance that is used to measure the performance of sequential evaluation bandits. It describes the difference between a chosen action versus the best course action in cases where decisions are being made in uncertain scenarios. It is commonly defined in terms of *cumulative* regret $R_T = \sum_T^t r_t$, where $r_t = f_t(\boldsymbol{x}^\star) - f_t(\boldsymbol{x})$, which is the difference between the evaluation at time t and evaluation at the true optimum $\boldsymbol{x}^\star$. Cumulative regrets is often used in the theoretical analyses for bandit methods. For more information, see Section 2 of Shahriari et al. (2015).

where the aforementioned ϵ is called the *forgetting factor*. This kernel is the Matérn $\frac{1}{2}$ (also called the exponential covariance function) and it describes an Ornstein-Uhlenbeck process (Rasmussen & Williams 2006, §4.2.1). It induces an autoregressive process of order one (AR1), hence the previously mentioned simple Markov temporal evolution behaviour.

Bogunovic et al. (2016) also state that their technique and related formalizations apply to other spatiotemporal stationary kernels, but they chose to focus on the model described by Equations 3.3–3.4 for concreteness. These types of spatiotemporal models have also been considered in work on the product kernels of contextual bandits in Krause & Ong (2011), and this time-varying bandit formalization is a special case of that.

In the time-varying bandit setting, their theoretical analyses show that for a fixed ϵ, the asymptotic lower bound of the expected regret is of the form $\Omega(T\epsilon)$, where T is the number of time-steps (See Theorem 4.1 in Bogunovic et al. (2016)). This lower bound describes the regret in terms of T and the rate that the function varies ϵ, a useful characterization. It thus motivates the study of the joint dependence of the regret on T and the variation parameter ϵ. In particular, they prove that the regret behaves as $\mathcal{O}(T\epsilon^{\alpha})$ for popular stationary kernels (each having different power laws), where $\alpha > 0$. This induces the relationship where $\epsilon^{\alpha} \to 0$ as $\epsilon \to 0$ (See Section 4.1 & Corollary 4.1 in Bogunovic et al. (2016)). This characterization is similar to approaches found in earlier work on *Brownian restless bandits* by Slivkins & Upfal (2008), which involve an unavoidable regret bound of $\Omega(T)$ for any fixed variation parameter, and they focussed on the behaviour of the implied constant in the limit of a vanishing variation parameter.

For the commonly-used squared exponential and Matérn kernels, Bogunovic et al. (2016) obtain regret bounds of the form $\mathcal{O}(T\epsilon^{\alpha})$, where $\alpha > 0$ as previously stated. This can be viewed as being sublinear if $\epsilon = \mathcal{O}(T^{-c})$ for some $c > 0$. They observed that for $c < 1$, the correlation between the initial function instance f_1 and some future function instance f_T is negligible, which implies

that the minimum of such a function changes drastically over the duration of the time horizon.

The interpretation and practical justification provided by Bogunovic et al. (2016) is that their method is useful because it aids the GP-UCB algorithm to treat old and new data differently given that the function is varying with time. They also periodically reset the data used by the algorithm as another practical way of achieving this, i.e., throwing away stale data as the algorithm proceeds.

3.3 Model for Dynamic Optimization Problems

In this section, we describe the surrogate model to use for the dynamic optimization problem. We model a DOP $f(\boldsymbol{x}, t)$ as a spatiotemporal Gaussian process prior where we assume that the instance of the objective function at time t is a slice of f constrained at t. We think of these instances as versions of the objective function when evaluated at time t. As a consequence, this GP model captures correlations of the temporally evolving function in space and time. This relationship facilitates adaptation and acclimatization to any occurring temporal evolution by encoding any deviations from the existing model using Bayesian updates. Additionally, the sequence that the instances of the dynamic function are observed is encoded by their order along the time axis of f. This order encodes a notion of real-world time into our model and allows us to consider a concept of time that is independent of that associated with a specific practical problem. This consideration of time facilitates the fusion of data from different sources as long as it is sampled from the same latent function f.

Thus we take

$$f(\boldsymbol{x}, t) \sim \mathcal{GP}(\boldsymbol{0},\ k(\{\boldsymbol{x}, t\}, \{\boldsymbol{x}', t'\})\ ), \tag{3.5}$$

where $(\boldsymbol{x}, t) \in \mathbb{R}^{D+1}$, and $k(\cdot, \cdot)$ is the covariance function of the zero-mean spatiotemporal Gaussian process. As in many practical applications, we will

make the following simplifying assumptions about this spatiotemporal Gaussian process (Gneiting et al. 2006):

- **Assumption 1.** *Separability: The process has a separable covariance*

$$\mathbb{C}\text{ov}(f(\boldsymbol{x}, t), f(\boldsymbol{x}', t')) = k_{\text{S}}(\boldsymbol{x}, \boldsymbol{x}') \cdot k_{\text{T}}(t, t'),$$

  where $k_{\text{S}}(\cdot, \cdot)$ and $k_{\text{T}}(\cdot, \cdot)$ are the spatial and temporal covariance functions, respectively. ;

- **Assumption 2.** *Stationarity in space and time: We assume there exists a positive semi-definite function K defined on $\mathbb{R}^{D+1}$ such that*

$$\mathbb{C}\text{ov}(f(\boldsymbol{x}, t), f(\boldsymbol{x}', t')) = k(\boldsymbol{x} - \boldsymbol{x}', t - t'), \qquad \forall (\boldsymbol{x}, t), (\boldsymbol{x}', t') \in \mathbb{R}^{D+1}.$$

So we will take our covariance function k to be of the form:

$$k(f(\boldsymbol{x}, t), f(\boldsymbol{x}', t')) = k_{\text{S}}(\boldsymbol{x}, \boldsymbol{x}') \cdot k_{\text{T}}(t, t').$$

Unlike in Bogunovic et al. (2016) where they took the temporal kernel k_{T} to be that of an Ornstein-Uhlenbeck process with the form

$$k_{\text{T}}(t_i, t_j) = \epsilon^{\frac{(t_i - t_j)}{2}},$$

where $\epsilon \in [0, 1]$ is called the *forgetting factor* that describes the degree of temporal function variation, we consider any suitable stationary covariance function as the temporal kernel. This characterization is also similar to the product kernels used in contextual bandits (Krause & Ong 2011), except that our context is time. Spatiotemporal GPs for time-varying models also arise naturally in kernel regressive least squares (KRLS) methods (Van Vaerenbergh et al. 2012), although their formulations only consider a smaller class of temporal kernels.

For stationary covariance functions, the common hyperparameters include the noise variance, and input and output length-scales. After training the Gaussian process model, the learnt input length-scales tell us about the variability of

$f(\boldsymbol{x}, t)$ in space and time. Given that we are using stationary covariance functions, the time-related hyperparameters such as the characteristic length-scale in the time dimension become critical. If we have enough data for adequate training, this hyperparameter can be informative about the rate that the objective function instances captured by $f(\boldsymbol{x}, t)$ change over time. Specifically, it tells us the temporal range of instances that have similar behaviour as described by the underlying learnt model. For example, if this length-scale is extremely large with respect to the duration of the finite time horizon, it means the objective function is not changing within that range. If it is very short relative to the finite temporal horizon, it means the function is changing quickly.

3.4 Adaptive Bayesian Optimization

In this section, we describe the appropriate extensions to standard Bayesian optimization that are required for tracking a temporally evolving minimum of a dynamic optimization problem. As previously stated, Bayesian optimization works by using existing data samples of the latent function and the Gaussian process prior to build a posterior distribution of the function over its domain. This posterior distribution is used to construct an acquisition function that leads the search for the minimum by exploiting and exploring the objective function. This sampling is done sequentially until either the exit conditions are met, or the evaluation budget is exhausted. The description in Section 2.1 is based on an objective of the form $f(\boldsymbol{x})$, where the goal is to search the whole domain for the minimizer $\boldsymbol{x}^\star$. However, we are interested in sequentially finding the minimum $\boldsymbol{x}^\star$ at associated time $t^\star$ so that we can track it effectively.

Modelling a dynamic optimization problem as a spatiotemporal Gaussian process as we have done introduces new complexities relating to the treatment of time in the model. By considering the real-world nature of time, we restrict the mode of observability for our model, and this brings with it several limitations:

- We maintain a notion of a current or present time when evaluating the dynamic optimization problem;
- We cannot make evaluations of the dynamic optimization problem in the past;
- The future is unseen, so we only have access to the past observations.

These limitations lead to the following consequences in how Bayesian optimization can sequentially search the dynamic optimization problem using the spatiotemporal surrogate model:

- The observed data is located on one side of the search space delimited by a hyperplane boundary defined by the current time (we assume that time t is one dimensional);
- As the algorithm determines the next evaluation to execute, the sequential search of the model can only be performed in regions where the times are in the future (of the present time).

These limitations are illustrated in Figure 3.1.

To search for and keep track of the minimum of a dynamic optimization problem, we assume that we have an appropriate spatiotemporal Gaussian process prior and an initial set of time-tagged data. Due to the limitations we have described, we can only evaluate $f(\boldsymbol{x}, t)$ at times meeting those requirements. To reflect these restrictions on the domain of $f(\boldsymbol{x}, t)$, we add constraints to the acquisition function optimization process.

Because we cannot collect samples in the past, the *lower bound* on the time variable t will be dictated by a notion of the *current* or *present time*. For the *upper bound*, which dictates how far into the future we are willing to consider solutions for, we would ideally want to consider all future times within the time-step horizon under consideration. This would allow us to solve the optimization problem in one-shot by figuring out where, at some point in the future,

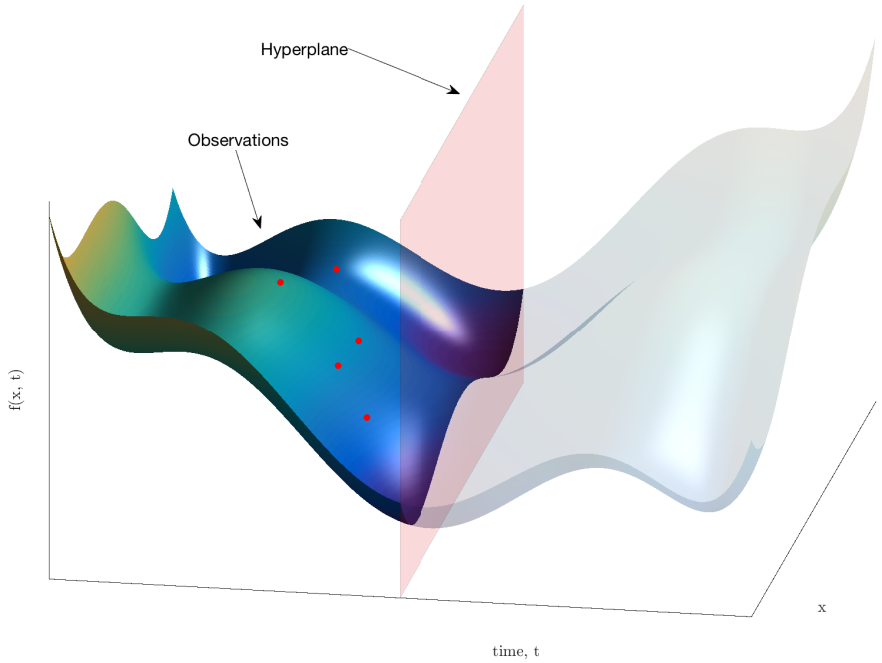

Figure 3.1: A 2D schematic illustration of the limitations introduced by considering real-word time when using a spatiotemporal model for a dynamic optimization problem. The previously observed data is shown by the red dots. The hyperplane corresponds to the current time.

we would get the best $\boldsymbol{x}^\star$. However, since we are using a Gaussian process prior for learning the objective function f, our ability to predict very far from our data samples will be dependent on the kernel used and the structure of the available data. In the worst case, the capacity to informatively extrapolate, with reasonable uncertainty, using the posterior distribution is going to be limited to a short distance from the last sample (for simple covariance functions)[3]. Specifically, and pessimistically, a length-scale distance away (Rasmussen & Williams 2006, §2.2). In our case, we can only predict well a temporal length-scale way in the time dimension. Therefore, a reasonable upper bound on the time variable would be anywhere within a length-scale distance away from the

[3]This is a worst-case assumption because some Gaussian process models, such as those having periodic kernels or those proposed in Duvenaud et al. (2011), among others, allow 'long-term' extrapolative predictions for some application domains.

lower bound. This critical time-related length-scale hyperparameter is learnt from training the Gaussian process model. The resulting feasible set is illustrated in Figure 3.2.

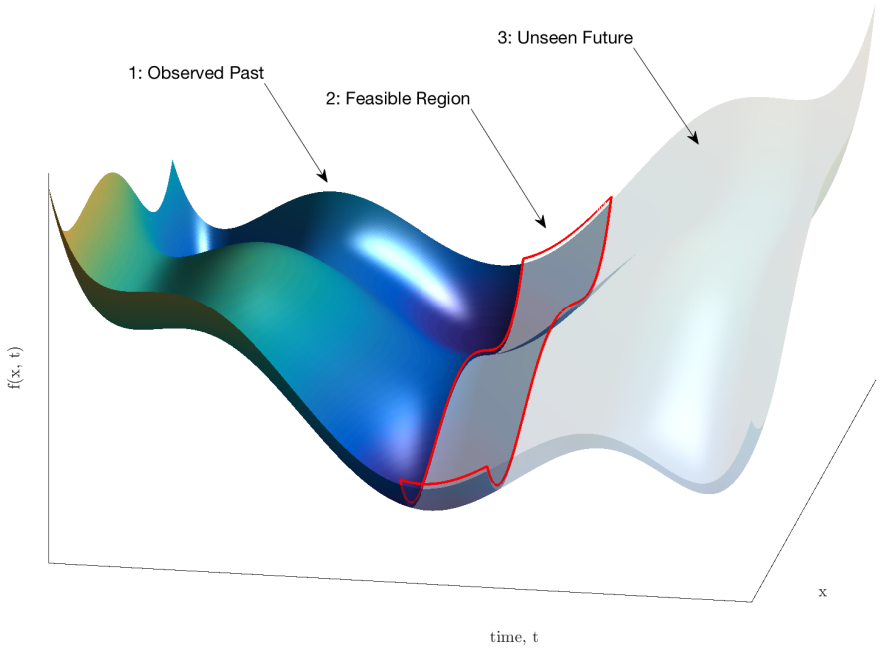

Figure 3.2: A 2D schematic illustration of $f(\boldsymbol{x}, t)$ for dynamic problems: Region 1 covers the previously observed objective function instances, region 2 shows the bounded feasible region $F(t)$ that we search at time t, and region 3 shows the unseen future.

Algorithm 2, which we call adaptive Bayesian optimization (ABO), shows the resulting adaptive method. The algorithm first trains the Gaussian process model and takes note of the learnt temporal length-scale hyperparameter which is subsequently used to set the temporal bounds of the feasible region $F(t)$.

The lower bound of the time variable is set to be a δ_t distance away from the current time t_c. We call this distance δ_t the *lower bound time delta*. This temporal distance is the minimum length of time into the future that we would like to consider solutions for. It encodes the minimum amount of allowable

time between evaluations of the objective function, and it is determined by the nature of the underlying physical experiment or simulation. For example, if the fastest frequency that the experiments or simulations can be executed is hourly, then $\delta_t = 1$ hour.

The upper bound is set to be within a fraction ρ of the temporal length-scale distance from the lower bound. We call this fraction $\rho \in [0, 1]$ the *upper bound threshold*. This user-defined threshold describes the degree to which our algorithm can flexibly look into the future for solutions. It is practically useful for cases where we cannot afford to make fixed-frequency evaluations of the objective functions within the finite horizon of allowed evaluations, i.e., a value of $\rho = 0$ tells the algorithm to ignore lookahead flexibility and make evaluations of the latent objective function at a fixed-frequency. On the other hand, a setting of $0 < \rho \leq 1$ tells the algorithm to be flexible in looking for solutions within a fraction ρ of the temporal length-scale distance into the future.

Algorithm 2 Adaptive Bayesian Optimization (ABO)

Input: Initial data $\mathcal{D}_0$, GP prior $f(\boldsymbol{x}, t) \sim \mathcal{N}(\boldsymbol{y} \mid \mu, K)$
Input: Budget of BO steps N, iteration label i
Input: Lower bound time delta δ_t
Input: Upper bound threshold $\rho \in [0, 1]$
Output: Minimum's trajectory $\{\boldsymbol{x}_i, t_i, y_i\}_{i=1}^{N}$

for $\boldsymbol{i} = 1, 2, ..., N$ **do**
  Train GP model via hyperparameter optimization
  Set current time t_c
  Set temporal length-scale l_t from optimized hyperparameters
  Set bounds $F(t_i)$ for feasible set

$$F(t_i) = \{\, \mathcal{S} : (t_\mathrm{c} + \delta_t) \leq t \leq (t_\mathrm{c} + \delta_t + \rho\, l_t) \,\}$$

  $\{\boldsymbol{x}_i, t_i\} = \underset{\{\boldsymbol{x}, t\} \in F(t_i)}{\arg\max}\, a(\boldsymbol{x}, t \mid \mathcal{D}_i)$
  $y_i \leftarrow f(\boldsymbol{x}_i, t_i)$, query objective function
  $\mathcal{D}_{i+1} = \mathcal{D}_i \cup \{\boldsymbol{x}_i, t_i, y_i\}$, update data
  Update current location of minimum
end for

Adaptive Bayesian optimization operates by changing the temporal bounds

of the feasible region $F(t)$ via heuristics using insights learnt from the spatiotemporal Gaussian process surrogate model. This region is constructed such that it describes a part of the dynamic optimization problem's domain which is pessimistically most informative for forecasting future changes relative to the current time. This feasible set is then used as the search domain by the mechanism that optimizes the acquisition function. The feasible region is modified in every iteration by taking advantage of new information from the Bayesian updates from previous steps. Consequently, the algorithm determines where and when to evaluate f to induce the behaviour that tracks the minimum. Thus the modification of the search region helps to keep track of a temporally evolving minimum.

Fixed-Frequency Evaluations

In practice, there are many circumstances where the times when to make evaluations of the dynamic optimization problem f are known in advance, and we can afford to make those evaluations within the allowed finite horizon of steps. These times are usually described in terms of how frequently the function is evaluated, and this frequency is determined by how the underlying physical problem is constructed and defined. In such cases, where we are least flexible about looking into the future, and the upper bound threshold is set to $\rho = 0$, the temporal bounds of the feasible region $F(t)$ will be fixed to the known times of interest. This vastly simplifies the problem and the feasible region $F(t_i)$ at Line 5 of Algorithm 2 will be defined by the temporal equality constraint given by

$$F(t_i) = \{\, \mathcal{S} : t = (t_\mathrm{c} + \delta_t) \,\}. \tag{3.6}$$

The acquisition function search in Line 6 will also be simplified to

$$\boldsymbol{x}_i = \underset{\boldsymbol{x} \in F(t_i)}{\arg\max}\; a(\boldsymbol{x}, t_i \mid \mathcal{D}_i). \tag{3.7}$$

In summary, adaptive Bayesian optimization tracks the minimum of a dynamic optimization problem in two possible ways. The first way involves flexibly determining when and where to evaluate the function, with some practical and pessimistic restrictions on how far into the future we can search. The second is making function evaluations at a fixed frequency when we can afford to do so. The modality of this behaviour is determined by the user-defined upper bound threshold parameter $\rho \in [0, 1]$, with a setting of $\rho = 0$ for the fixed-frequency case and $0 < \rho \leq 1$ for the temporally flexible case.

Sub-optimal Exploration Steps

In its standard form, Bayesian optimization operates by continuously exploiting and exploring the objective function in its search for the minimum. Using this mechanism for a dynamic case introduces one key problem for the procedure. Since the goal is to track the minimum, some explore steps could potentially heavily penalize the algorithm's tracking performance despite the importance of exploration for the learning process. Therefore, it becomes important to think about how to mitigate this critically.

One simple solution arises from the fact that initial data plays a crucial role in training the Gaussian process model before the subsequent evaluation procedure. The quality of the information learnt from training the GP determines how fast Bayesian optimization will reach the minimum. Since the evaluation choice automatically explores and exploits the surrogate model, good initial datasets facilitate a data-efficient way of finding the optimum.

In the case of adaptive Bayesian optimization, the *initial design* affects how well the temporal length-scale hyperparameter is initially learnt and also determines the quality of knowledge about the next tentative times for evaluation.

For a known budget of BO steps, we can allocate a few of those to obtaining these initial samples to learn the function well without worrying about the optimization and tracking task at hand. Ideally, this should be some initial steps we are willing to "throwaway" to aid better tracking of the minimum

at future steps. One way to select these initial samples is via the Bayesian optimization steps themselves that explore and exploit the allowable space.

Another way is to generate these initial samples randomly but place them in a space-filling manner using a Latin Hypercube Design (LHD) (Stein 1987). This placement would ensure that the initial data is spread out and covers as much of the allowable domain within those first few steps. This initial design is prepared as follows. Before the algorithm starts, the initial spatial samples are generated using an LHD. We then assign time-tags based on our initial budget of steps to these data samples and subsequently perform the associated initial evaluations of the objective function. For example, for a case of a budget of five initial latin hypercube samples, we generate those five spatial samples and then augment the time-tags t_1 to t_5 to them. Then ABO makes the five initial evaluations as described by the time-tagged initial LHD samples.

A mixture of these methods can also be used. This initial budget of steps allocated to *learning* can be further split between LHD and some explore-exploit BO steps. These evaluations, where tracking of the minimum is not required, are used to place additional points in the space to aid the optimization at future steps. Doing this takes advantage of the benefits of the initial design and the Bayesian optimization heuristics for initial learning of f.

Data Updates

The data updates from the sequential nature of this algorithm bring about the question of how to scale the inversion of the covariance matrix K with each data update. As with all modern implementations of Gaussian process regression, we do not invert the covariance matrix directly but use its Cholesky decomposition. In addition to being faster than inverting a matrix by providing better conditioning, it also allows us to use Cholesky updates and downdates for adding and removing data from the covariance matrix (Press et al. 1992, Chen et al. 2008, Osborne 2010). For more information on Cholesky updates and downdates, see Appendix A.

3.5 Why Adaptive Bayesian Optimization Works

In this section, we explain why adaptive Bayesian optimization manages to track the minimum of a dynamic optimization problem.

Standard Bayesian optimization operates by sequentially exploring and exploiting the latent objective function f with the aim of finding the extremum, and it has been shown to perform well in experiments (Osborne et al. 2009, Snoek et al. 2012, Shahriari et al. 2015, Calandra et al. 2016). There have also been theoretical analyses on the convergence of Bayesian optimization with different heuristics. For all those studies, the algorithmic convergence behaviour depends on the GP prior $p(f)$ used to model f and how it gains information about the objective function during the sequential evaluation process.

The theoretical analysis for the optimistic bandit-based GP-UCB method in Srinivas et al. (2012) shows that it achieves sublinear regret. The work in Bogunovic et al. (2016) extends GP-UCB to the case of time-varying bandits, and proves regret bounds of the form $\mathcal{O}(T\epsilon^{\alpha})$, where $\alpha > 0$ (for squared exponential and Matérn stationary kernels), with their formulation inspired by that used in Slivkins & Upfal (2008). This analysis characterizes the dependence of the regret on the function's rate of variation (captured by the parameter ϵ).

This theoretical analysis applies to the version of adaptive Bayesian optimization with fixed-frequency evaluations, where the rate of variation in our case is captured by the proportion of the horizon of time-steps covered by the temporal length-scale.

For the case where adaptive Bayesian optimization also determines *when* to make evaluations, the regret bounds only hold if the information gains resulting from the evaluations are as informative as the information gains from the fixed-frequency scenario. This version of ABO does not search parts of the input domain where the posterior distribution reverts to the prior, or where the posterior uncertainty is too high to be informative. Given that this variant

of ABO behaves in this way, it is reasonable to claim that this pessimistic evaluation scheme ensures that the function instances in the searchable temporal range encoded by $F(t)$ are similar to the function instances encountered in the fixed-frequency case. Consequently, the resulting regret bounds for this case will also behave similarly. This similarity in behaviour is because the information gains for this case, which are vital for characterizing the regret bounds, will be as informative as those for the fixed-frequency evaluation scenario.

The advantage of using the temporal length-scale hyperparameter rather than the forgetting factor as a parameter capturing function variation is that it is available in many stationary kernels. Having the ability to use these kernels as temporal covariance functions gives us the flexibility in defining and encoding appropriate prior beliefs about the evolution, thus giving us the ability to deal with many practical problems.

These theoretical analyses show that for adaptive Bayesian optimization to solve a DOP, the function's rate of change over time must be slow enough for us to gather enough samples to learn the relationships in space and time. If the rate of change of the function is faster than we can sample the function, the algorithm learns nothing. Therefore, the most suitable scenario is where the rate of change is slow enough for the algorithm to capture an evolution pattern.

3.6 Sparsification and Addressing Data Staleness

In this section, we describe the data sparsification (or model approximation) scheme to use if the number of datapoints for adaptive Bayesian optimization becomes too expensive for inference. We show that the insights learnt during ABO can be exploited to sparsify the Gaussian process model intelligently.

Given the sequential nature of Bayesian optimization algorithms, one problem that arises from using Gaussian process inference machinery is the cost

associated with a growing dataset. Specifically, the cost associated with inverting an $N \times N$ covariance matrix in every iteration, which costs $\mathcal{O}(N^3)$, and a prediction step, which costs $\mathcal{O}(N^2)$ per prediction. For large N, the prediction efficacy gained from adding a new datapoint may be negligible in comparison to the associated added computational cost (Samo & Roberts 2016).

Our goal is to construct a mechanism that identifies what parts of our time-tagged dataset are most informative in modelling the DOP $f(\boldsymbol{x}, t)$ for a new prediction. For a spatiotemporal model, we can use the covariance information to determine how informative each datapoint in the training set is for a new prediction at a future time t_f. We can then exclude the less informative data and place a limit on the number of datapoints we are using for GP inference, i.e., choose M most informative datapoints for the prediction out of the N available datapoints, where $N > M$. This subset selection would help with managing the computational costs and ensuring scalability.

To begin with, we first observe that if the latent function $f(\boldsymbol{x}, t)$ is dynamic, then the response y_p that we observe corresponding to $\boldsymbol{x}_\mathrm{p}$ at *past* time t_p would be different from the response y_f we would observe at a *future* time t_f (also corresponding to $\boldsymbol{x}_\mathrm{p}$). We assume that $t_\mathrm{f} > t_\mathrm{p}$ and t_f is far enough into the future for some variation in f. However, if the latent function is *not* dynamic, there would be *no* difference between the response observed at the past time t_p and its corresponding response at future time t_f.

Using this rationale, we can infer what our past time-tagged data $\{\boldsymbol{x}_i, t_i, y_i\}_{i=1}^N$ would be if it was collected at a future time t_f using our trained spatiotemporal GP model. The inference of the posterior distribution $p(\, y_\mathrm{sf} \mid \boldsymbol{x}_\mathrm{p}, t_\mathrm{p}, y_\mathrm{p}, t_\mathrm{f}\,)$ from our GP model yields mean y_sf and standard deviation σ_sf estimates at future time t_f corresponding to all past data, which we will call the *synthetic future* (sf) data. We can calculate the signal-to-noise ratio of the future mean and standard deviation estimates. This criterion will tell us of how informative each synthetic future datum provides under the GP model. The SNR criterion is

given by

$$\text{SNR} = \frac{y_{\text{sf}}}{\sigma_{\text{sf}} + \sigma_{\text{n}}}, \tag{3.8}$$

where σ_{n} is the measurement noise variance.

Therefore, if we have an inference budget of M points, we can then select M points out of existing N points with the highest signal-to-noise ratio (SNR). This procedure is illustrated in Figure 3.3 and is also delineated in Algorithm 3.

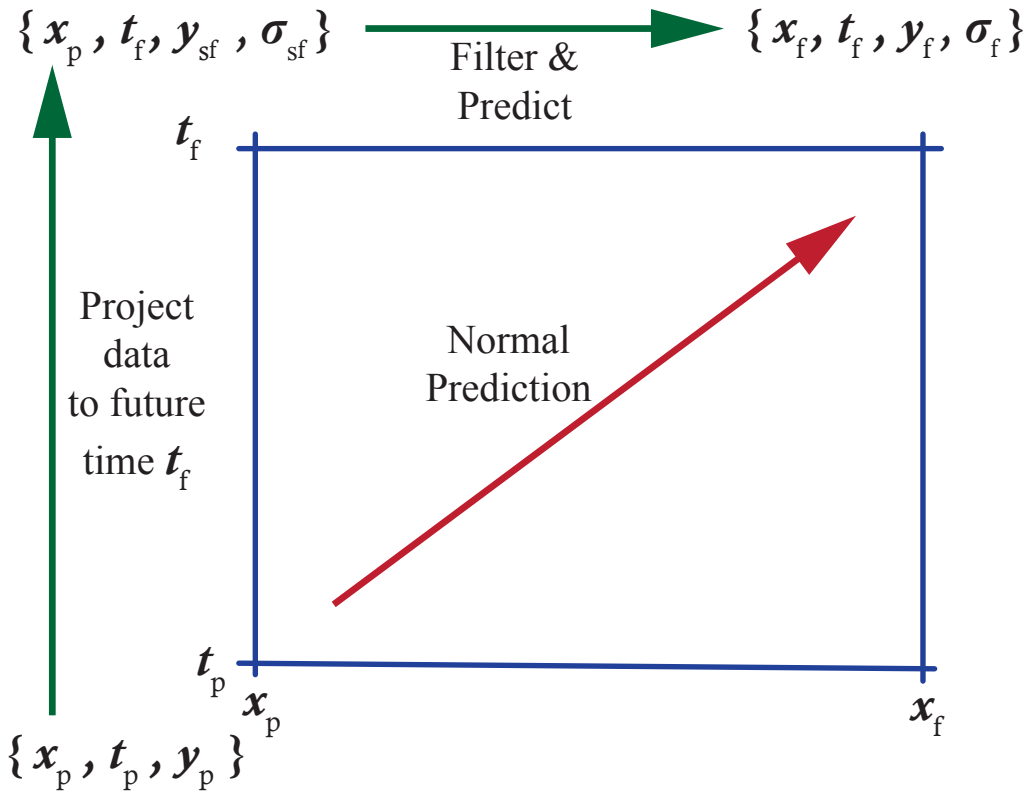

Figure 3.3: Illustration of how our subset-of-data sparsification works.

Using the SNR criterion makes sense because the posterior means and standard deviations at t_{f} tell us how much uncertainty is added due to an epistemic shift in time under the learnt GP model. The SNRs of the future synthetic measurements are lower if $\sigma_{\text{sf}} > 0$, and the SNR, therefore, characterizes the epistemic attenuation under the GP model of f.

Algorithm 3 Subset of Data Sparsification

Input: Number of datapoints: N
Input: Number of desired points M
Input: Set of past data $\mathcal{D} = \{\boldsymbol{x}_i, t_i, y_i\}_{i=1}^{N}$
Input: GP prior $\mathcal{GP}(\mu, K)$
Output: Subset of data $\mathcal{D}_{\text{SOD}}$

1: **if** $N > M$ **then**
2: Project each past datum in $\mathcal{D}$ from past time t_p to future time t_f,

$$\{\boldsymbol{x}_\text{p}, t_\text{p}, y_\text{p}\} \mapsto \{\boldsymbol{x}_\text{p}, t_\text{f}, y_\text{sf}, \sigma_\text{sf}\}$$

using GP posterior distribution equations:

$$\begin{aligned} y_\text{sf} &\triangleq \mu(\{\boldsymbol{x}_\text{p}, t_\text{f}\}) \\ &= k(\{\boldsymbol{x}_\text{p}, t_\text{f}\})^\top \left(\boldsymbol{K} + \sigma_\text{n}^2 \boldsymbol{I}\right)^{-1} \boldsymbol{y}, \end{aligned}$$

$$\begin{aligned} \sigma_\text{sf}^2 &\triangleq \sigma^2(\{\boldsymbol{x}_\text{p}, t_\text{f}\}) \\ &= k(\{\boldsymbol{x}_\text{p}, t_\text{f}\}, \{\boldsymbol{x}_\text{p}, t_\text{f}\}) - k(\{\boldsymbol{x}_\text{p}, t_\text{f}\})^\top \left(\boldsymbol{K} + \sigma_\text{n}^2 \boldsymbol{I}\right)^{-1} k(\{\boldsymbol{x}_\text{p}, t_\text{f}\}). \end{aligned}$$

where $K_{i,j} = k(\{\boldsymbol{x}_\text{p}, t_\text{p}\}_i, \{\boldsymbol{x}_\text{p}, t_\text{p}\}_j)$ and $k(\{\boldsymbol{x}_\text{p}, t_\text{f}\})$ is a vector of covariance terms between $\{\boldsymbol{x}_\text{p}, t_\text{f}\}$ and $\boldsymbol{X}$, the input training data.

3: Rank projected points by the signal-to-noise ratio
4: Select past data corresponding to the first M points in the ranked set
5: Return subset of projected points that satisfy signal-to-noise ratio criteria
6: **end if**

One other issue relating to the informativeness of the data samples in relation to future evaluations of the DOP is that of *data staleness* as identified in Bogunovic et al. (2016). They demonstrate that periodically removing older data improves the learnt model for the dynamic problem. Given our mechanism for identifying less informative data and the sequential nature of Bayesian optimization, we can perform the sparsification and the purging of stale data in tandem.

We sparsify the Gaussian process model when we have more datapoints than we can afford for inference. On the other hand, to address data staleness, we throw away blocks of stale data and sequentially rebuild them to the acceptable limit to allow the sequential learning of the model to naturally take place with

a growing dataset.

To deal with and balance these two requirements, let us assume that we have a predefined maximum limit $M_{\max}$ on the number of datapoints that we can afford. For the initial phase of adaptive BO, where the number of datapoints N is less than $M_{\max}$, we perform the GP inference normally. For each of the steps with N datapoints, an $N \times N$ covariance matrix is used for inference and one new datapoint is added due to adaptive BO. At the ABO step after we add the $(M_{\max}+1)$-th datum, where $N > M_{\max}$, we activate the sparsification. We use the covariance information from the previous step to calculate the synthetic future values of all the $M_{\max}$ past data and perform the sparsification according to Algorithm 3, only selecting M_{block} datapoints out of the $M_{\max}+1$ (where $M_{\text{block}} < M_{\max}$). For the subsequent steps, we add new datapoints normally until we reach the maximum $M_{\max}$, after which we repeat the process. This procedure is shown in Algorithm 4

Since the covariance matrix is capped at being of size $M_{\max} \times M_{\max}$, and the matrix inversion information is available from the previous adaptive BO step, the only additional cost of sparsification is $\mathcal{O}(M_{\max}^2)$ per synthetic prediction, which happens when a block of $(M_{\max} - M_{\text{block}})$ datapoints is added (after the initial $M_{\max}$ datapoints).

Algorithm 4 Sparsification and Stale Data Removal

Input: Number of datapoints: N
Input: Maximum number of desired datapoints: $M_{\max}$
Input: Number of desired datapoints per data block: M_{block}
Input: Maximum number of steps: T
1: **for** $i = 1, 2, ..., T$ **do**
2: **if** $N > M_{\max}$ **and** t mod $M_{\text{block}} = 1$ **then**
3: Select M_{block} datapoints from $M_{\max}$ using Algorithm 3
4: **end if**
5: Perform adaptive Bayesian optimization step
6: **end for**

3.7 Experiments

We have compared the results of our method with alternatives using tests on an extensive set of standard problems and one real-world example. The other competing methods that we consider are:

- Standard Bayesian optimization using lower confidence bounds (GP-UCB) (Srinivas et al. 2012),
- Time-varying Gaussian process bandit optimization (TVB) (Bogunovic et al. 2016),
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen 2006),
- Dividing Rectangles (DIRECT) (Jones et al. 1993), and
- Particle Swarm Optimization (PSO) (Poli et al. 2007).

Table 3.1 shows the mechanisms that the non-BO competitor methods use to search for the minimum. These methods are also derivative-free. Our experiments tested how well the competing algorithms tracked the minimum of a dynamic optimization problem by measuring the average tracking behaviour over the duration of the finite time horizon. The BO-based methods (ABO, GP-UCB, TVB) used Gaussian process surrogate models while the others did not. The surrogate models used by ABO and TVB took time variations into account while CMA-ES, DIRECT and PSO adapted to changes by managing the diversity of their function evaluations to be reactive to changes of the objective function.

We use two versions of our algorithm. We denote the variant that performs function evaluations at a fixed-frequency as ABO-f (with an upper bound threshold setting of $\rho = 0$), where the suffix '-f' stands for *fixed*. We denote the variant that is pessimistically flexible about making evaluations in the future as ABO-t (we used an upper bound threshold setting of $\rho = 0.5$), where the

Table 3.1: How the non-BO and derivative-free competitor optimization methods search for the minimum.

Method	Search Mechanism
CMA-ES	Uses stochastic and evolutionary evaluations
DIRECT	Uses multi-dimensional pattern search
PSO	Uses population-based evolutionary metaheuristics

suffix '-t' stands for *time*. Because we assume that the dynamic optimization problems that we consider are expensive to evaluate, we limit the number of allowed evaluations during the optimization process. This limit is a natural stopping criterion as we are also interested in cases of this problem with finite evaluation horizons. All the competing methods used the same budget of function evaluations during the experiments.

Implementation Details

Each method that we tested used its recommended default settings with an exception on the number of initial evaluations used for initial algorithmic configurations. The BO-based techniques used the lower confidence bound (LCB) search heuristic (Srinivas et al. 2012). A parallelized hybrid of the PSO algorithm performed the optimization of the acquisition functions, where the Broyden-Fletcher-Goldfarb-Shanno (BFGS) optimization algorithm (Fletcher 1987) was executed after the PSO method terminated to further fine-tune the solution returned by PSO to the closest local minimum. Additionally, only two initial data samples were generated using a Latin hypercube design within the predefined allowed region for the initial data.

The input and output length-scales of the Gaussian process models were estimated using maximum marginal likelihood to make the testing procedure cheaper to perform over many time-steps and experimental batches. We repeated the experiments ten times to obtain the average performance on each problem.

Metric for Assessing and Comparing Performance

The goal of our experiments is to compare how competing algorithms find and keep track of the minimum of a dynamic problem using optimization. As stated before, as the algorithms perform the optimization, the objective function changes slightly, and thus the algorithms must keep track of the temporally evolving optimum. We want the algorithms to make sequential evaluations of the objective function such that each evaluation is as close as possible to the location of the minimum at that point in time. Consequently, when the algorithms terminate, we expect to obtain a sequence of evaluations that are as close as possible to the locations of the actual minimum at those times.

An ideal approach to compare the performance of the competing optimization methods is to use the difference between an evaluated point and the value of the minimum then. However, we have no access to the true minimums at those points in time for the experiments in this chapter. So an alternate comparison approach is to only compare the values of the function at the evaluated points, i.e., lower values are better.

Given that there is a temporal evolution in the underlying objective function as the optimization takes place, not all the evaluated points will be as close to the minimum as desired. So to make the comparison fairer, we compare the best values returned by the optimization algorithm within a window of evaluation steps. For example, we might want to consider the lowest evaluation in the last five steps as our score for the current step. The rationale of doing this is that the optimization algorithms will have a few time steps of evaluations within this window to acclimatize to changes, and picking the best point within this window of steps is a good proxy for appraising how well the tracking of the minimum has occurred within that time span.

One additional consideration to make when comparing optimization algorithms operating on time-varying problems is the requirement to assess the

complete sequence of *windowed evaluations*[4] for a problem. Looking at an entire series of these evaluations allows us to compare the performance of different optimization methods on various dynamic problems. Since we already have a way of comparing evaluations at particular time steps (i.e., get the lowest value within a window of evaluations), we can make use of these values to get an overall score for a sequence of windowed evaluations. One intuitive metric is the average of these windowed evaluations over the complete series of steps. An average value over the whole sequence comparatively tells us how well the optimization methods tracked the temporally evolving minimum of the various problems.

What we have described thus far corresponds to a popular metric for comparing optimization methods operating on dynamic problems. This comparison metric is called the *offline-performance* (B) and defined as

$$B(T) = \frac{1}{T}\sum_{t=1}^{T} b_t, \tag{3.9}$$

where

$$b_t = \min\{b_i : i = t, (t-1), ..., (t-w-1)\},$$

is the best solution[5] within a sliding window w of iterations at iteration t, and the abbreviation 'B' stands for 'best' function evaluation seen so far (i.e., the lowest value seen within a window of steps). This metric is based on the average performance over the duration of a moving short evaluation window w (we use a duration of $w = 5$ iterations for the experiments). In practice, this measure assumes that the comparison of the performances of the sequential optimization methods occurs after their full sequence of evaluations are completed; hence the prefix 'offline' as this assessment is done after the algorithms terminate. This metric was proposed in Branke (1999), a seminal paper in evolutionary

[4]These are the evaluations that are selected based on the lowest value within a window of time steps as described in the previous paragraph. This assessment is done at every time step to produce the complete sequence which we call the windowed evaluations.

[5]The lowest value that has been seen thus far.

Table 3.2: Standard test objective functions.

Function	Function Name	Test Region
6C	6-hump Camelback	$[-3,3] \times [-2,2]$
Br	Scaled Branin	$[0,1]^2$
G-P	Goldstein-Price	$[-2,2]^2$
Gr	Griewank	$[-5,5]^D$
H3/6	Hartmann 3/6	$[0,1]^D$
Sh	Shekel	$[0,10]^4$
Sty2/7	Syblinsky-Tang 2/7	$[-5,5]^D$

optimization for dynamic problems, and is a commonly-used comparator for optimization methods operating on dynamic problems.

Standard Test Objective Functions

The first set of experiments were performed on a diverse set of multidimensional test functions (see Table 3.2). As a class of dynamic optimization problems, their evolution over time is smooth. For each function, we randomized which dimension was treated as the time variable t, with the rest being treated as spatial variables for every instance of the batch experiments. To model smooth variations over time, we used the squared exponential kernels to model both the temporal and spatial components. Additionally, since the temporal variable was randomly selected, it was practically easier to model all dimensions using a squared exponential kernel.

The results of the experiments using fixed-frequency evaluations are shown in Table 3.3. Figures 3.4 and 3.5 show a visualization of some evaluation trajectories of ABO-f on some of the two-dimensional standard problems. ABO-f performed better (or tied with) the best out of the algorithms tested on most of the test problems. For the cases where it was not the best, like for Hartmann and Shekel functions, it was due to the behaviour of the minimum over time for those functions. While the value at the minimum changed over time for those functions, its location did not vary much. As a result, methods that do not

Table 3.3: The mean of offline-performances (B) for various optimization methods operating using fixed-frequency evaluations on various dynamic optimization problems. Numbers highlighted in bold are the best results for the relevant problem.

Problem	GP-UCB	CMA-ES	DIRECT	PSO	TVB	ABO-f
6C	0.60	0.72	0.86	1.42	0.25	**-0.09**
Br	-0.69	-0.62	0.36	1.70	-0.74	**-0.89**
G-P	3952	6311	4686	29543	16908	**1130**
Gr	1.11	0.61	0.88	0.82	0.57	**0.37**
H3	**-3.63**	-1.18	-2.95	-3.10	-2.50	-3.31
H6	**-2.95**	-1.77	-1.57	-2.05	-1.33	-2.32
She	**-5.18**	**-5.18**	**-5.18**	**-5.18**	-0.35	-5.10
Sty2	-30.2	-32.5	-44.7	-43.4	9.9	**-44.9**
Sty7	-116.1	-119.4	-66.4	-96.9	-89.4	**-158.9**
MPB-1	-4.8	-23.8	-6.7	-6.0	-25.7	**-29.0**
MPB-2	-0.06	-0.03	-0.05	-0.01	-0.01	**-0.64**
Intel	-1.4e-05	-1.5e-73	2.2e-3	4.3e-10	-1.0e-06	**-9.7e-3**

explicitly consider time variations benefited from exploiting information gained from assuming a static objective function. On the other hand, ABO's exploration to assess how the objective function was varying led to some suboptimal evaluations that caused the slightly lower performance.

Table 3.4 shows the results related to ABO-t, which also searches for suitable times to make evaluations. Instead of a budget of evaluations, the termination condition for ABO-t was when it reached the end of the time-step horizon. The results show the offline-performance of ABO-t and the percentage difference between the iterations used by ABO-t and those used by the other algorithms operating at a fixed-frequency. The results show that ABO-t performed well on most of the problems with fewer iterations. In some cases, it used more iterations than the fixed interval case which also led to better performance than ABO-f on the corresponding tests. This observation empirically demonstrates the importance of faster-frequency evaluation in terms of its impact on the tracking behaviour as shown by the regret bounds by Bogunovic et al. (2016). These results also demonstrate ABO-t's ability to select the times for evalua-

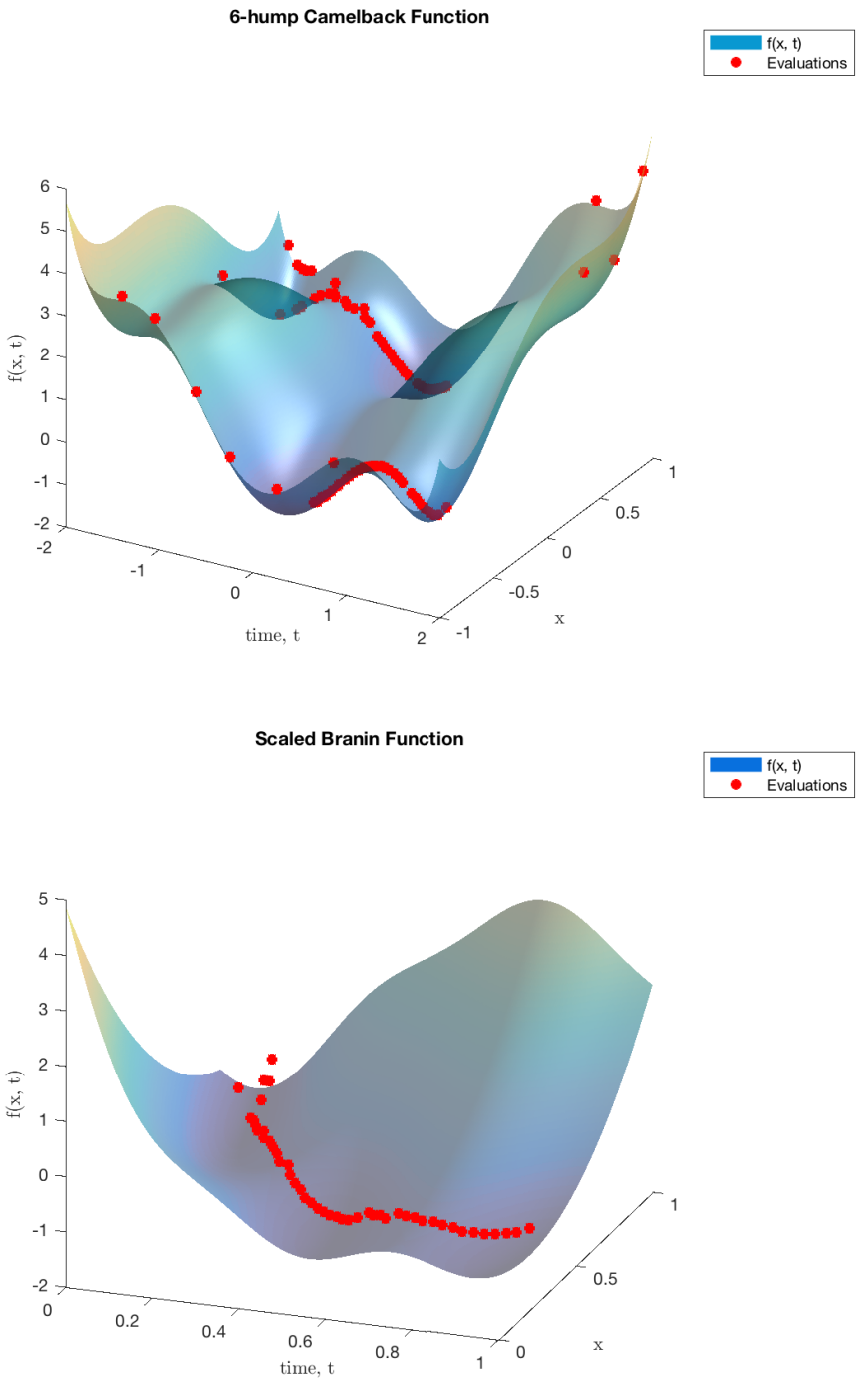

Figure 3.4: Visualization of some trajectories of the evaluations of ABO-f on some of the 2D standard problems. Top: 6-hump Camelback function. Bottom: Scaled Branin function.

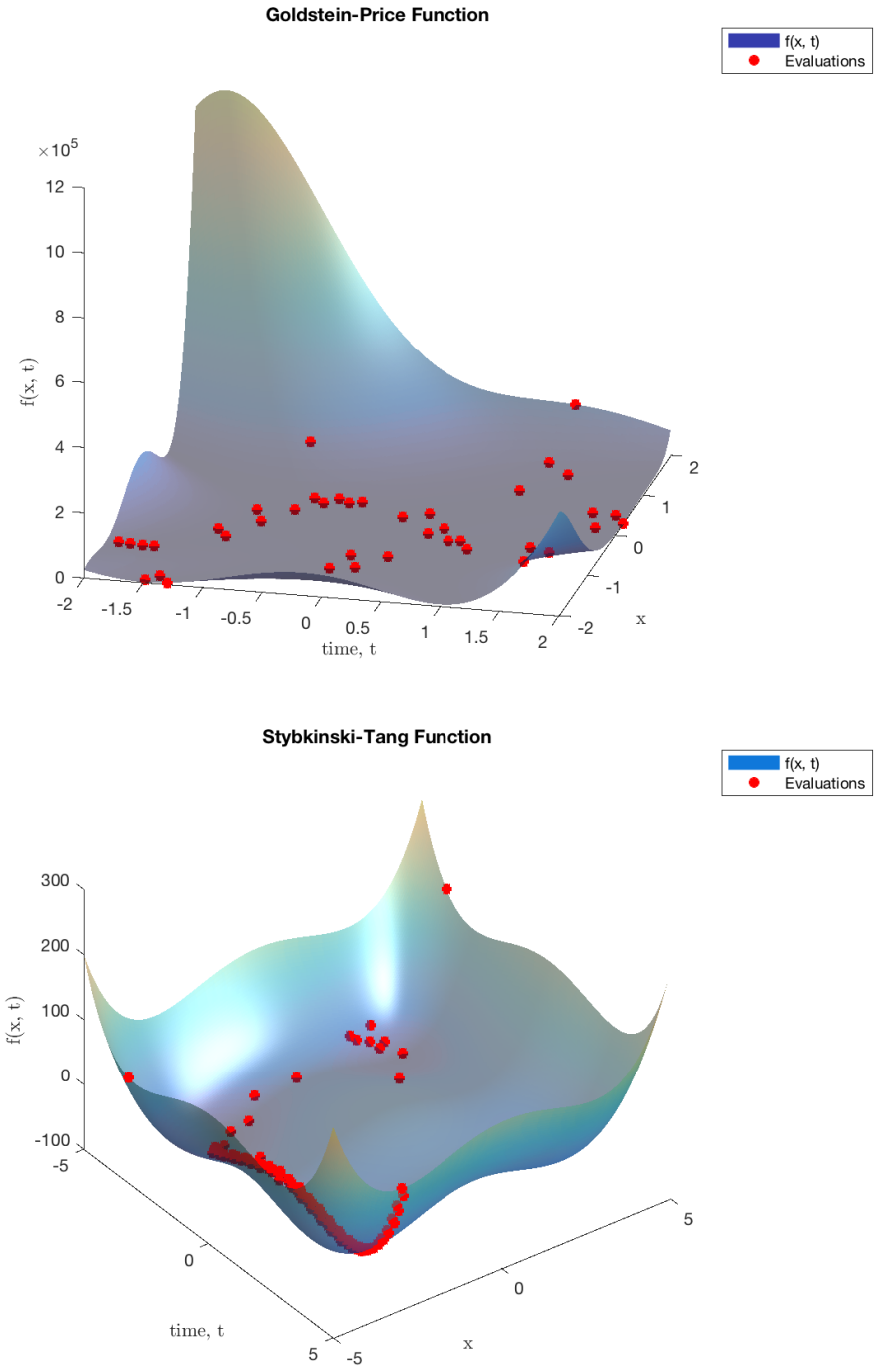

Figure 3.5: Visualization of some trajectories of the evaluations of ABO-f on some of the 2D standard problems. Top: Goldstein function. Bottom: Styblinski-Tang function.

Table 3.4: The mean of offline-performances (B) for ABO-t on various dynamic optimization problems. The last column shows the percentage difference between the iterations used by ABO-t (with $\rho = 0.5$) and those used by the other algorithms operating at fixed time intervals.

Problem	ABO-t	Δ %
6C	0.14	-56%
Br	-0.87	-16%
G-P	727	-40%
Gr	0.21	+13%
H3	-3.77	+92%
H6	-1.33	-60%
She	-3.67	-33%
Sty2	-58.0	-76%
Sty7	-110.6	-69%
MPB-1	-16.4	-27%
MPB-2	-0.19	-16%

tion in a manner that improved tracking performance if we could afford it (as determined by the user-configured upper confidence bound threshold ρ).

Moving Peaks Benchmark

The Moving Peaks Benchmark (MPB) (Cruz et al. 2011) has been used extensively to test dynamic optimization approaches. The problem is designed to cover a large range of dynamically changing fitness landscapes. It consists of a multi-dimensional landscape with a definable number of peaks m, where the height, width, and position of each peak are altered slightly every time a change occurs. The MPB is defined as

$$F(\boldsymbol{x}, t) = \max_{i=1,\ldots,m} \frac{H_i(t)}{1 + W_i(t) \sum_{i=1}^{d} (x_i - X_i j(t))^2}, \tag{3.10}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the spatial input, t is time, $X \in \mathbb{R}^{m \times d}$ is the set of m peak locations of d dimensions, and $H, W \in \mathbb{R}^m$ are vectors storing the height and width of every peak. Figure 3.6 shows an illustrative scenario with $m = 5$ peaks and $d = 2$ dimensions.

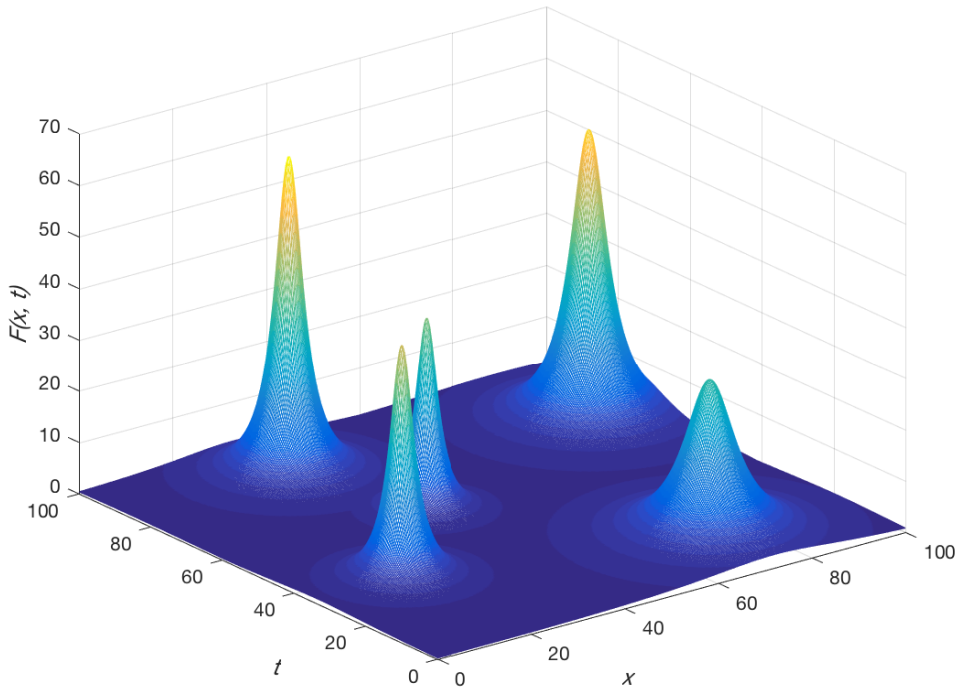

Figure 3.6: Illustration of the moving peaks benchmark with $d = 2$ and $m = 5$.

The coordinates, the height H_i and the width W_i of each peak are randomly initialized. At certain times, the height and width of every peak are changed by adding a random Gaussian variable multiplied by a specific "severity" factor (h_{sev}, w_{sev}, respectively). The location of every peak is moved by a vector $\boldsymbol{v}$ of fixed length s in a random direction for $\alpha = 0$, or a direction depending on the previous one for $\alpha > 0$. Thus α is a correlation coefficient allowing to control whether the changes exhibit a trend or not.

Given $\sigma \sim \mathcal{N}(0, 1)$ and $\boldsymbol{r}$ is a randomly drawn vector ($r_i \sim \mathcal{N}(0, 1)$), we can describe these changes as

$$H_i(t) = H_i(t-1) + h_{\text{sev}} \times \sigma,$$

$$W_i(t) = W_i(t-1) + w_{\text{sev}} \times \sigma,$$

$$X_i(t) = X_i(t-1) + \boldsymbol{v},$$

$$v(t) = \frac{x_{\text{sev}}}{\boldsymbol{v}(t-1) + \boldsymbol{r}}\Big[(1-\alpha)\boldsymbol{r} + \alpha\boldsymbol{v}(t-1)\Big].$$

For ABO, our choice for the temporal covariance function for this problem was the sum of a Matérn $\frac{1}{2}$ and a squared exponential covariance function to capture both smooth and abrupt temporal variations in the objective function. We used the squared exponential covariance function as the spatial kernel.

For our experiments, we performed tests on popular scenarios 1 (10 dimensions) and 2 (50 dimensions) with the MPB settings described in Moser & Chiong (2013). The results for the case of fixed-frequency evaluations are shown in the bottom part of Table 3.3. ABO-f performed best on the tests. Since the MPB has abrupt changes in the objective function, these results demonstrate how ABO can also capture non-smooth time dynamics. This behaviour is in addition to being able to capture smooth dynamics in the previous set of experiments on the standard test functions. These results demonstrate ABO's temporal modelling flexibility unlike the temporal modelling restriction imposed by TVB, which only uses Matérn $\frac{1}{2}$ as its temporal kernel.

The non-BO methods performed worse than ABO despite being able to solve this problem if allowed more iterations. ABO performed better with fewer iterations because it is both data-efficient and learns time variations, i.e. it makes maximal use of the information from the Bayesian updates to guide the search for the moving minimum. The other BO-based methods do even worse. This is because standard BO does not consider time dynamics, and TVB's time variation model is not general and flexible enough to capture the dynamics in the MPB.

Temperature Dataset

The third set of experiments is on temperature data from 54 sensors at the Intel Research Laboratory in Berkeley, California between February 28th and April 5th, 2004, deployed as shown in Figure 3.7 (Bodik et al. 2004). The goal

was to find which location on the map had the maximum temperature at any given time[6].

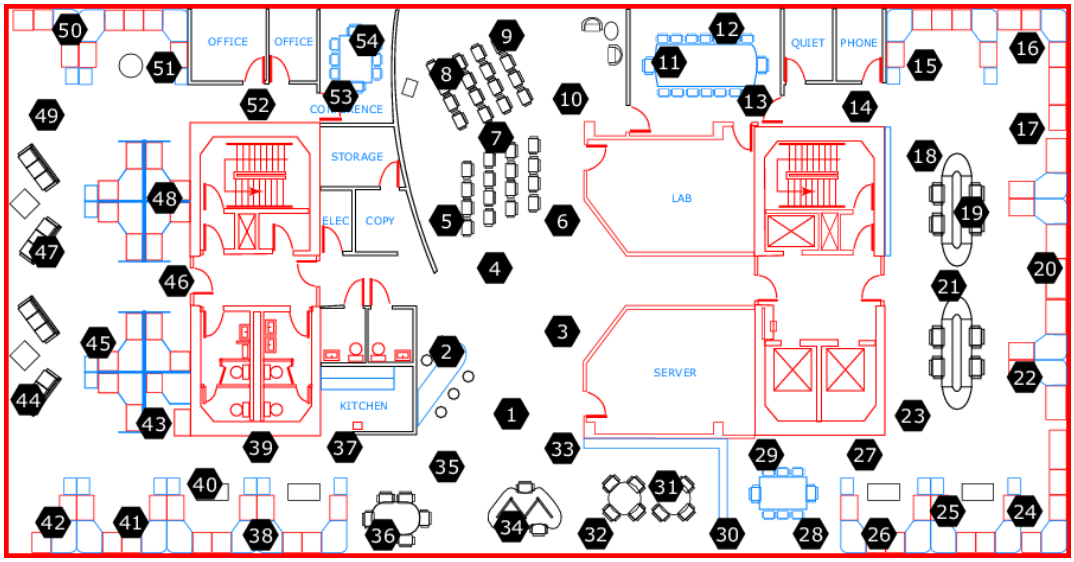

Figure 3.7: Intel Berkeley laboratory sensor deployment map. Source: Bodik et al. (2004).

The data contained the sensor location coordinates x and y (in metres relative to the upper right corner of the lab). The experiments used this spatial orientation. The dataset also contained readings collected from these sensors with the schema: `date, time, epoch, temperature`. Epochs are monotonically increasing number sequences from each sensor. Two readings with the same epoch number were produced from different sensors at the same time. There were some missing epochs in this dataset.

For ABO, our choice for both the temporal and spatial kernels for this problem was the sum of a Matérn $\frac{1}{2}$ and a squared exponential covariance function. These kernels were selected to capture both smooth and abrupt changes in the spatial and temporal components of the objective function. The data was also mean-adjusted so that we could use a zero-mean Gaussian process prior.

[6]We treat this as a minimization problem by considering negated values of the readings from the sensors.

We considered data over the first 3000 readings. Due to the large size of the dataset for repeated batch testing, the GP model was initially trained on 66% of the data and used the learnt hyperparameters for the optimization on the rest of the data sequence without retraining. For the non-BO methods, the first 66% of the budgeted steps were not scored but used to configure the algorithms.

The bottom part of Table 3.3 also shows the outcome of the experiments. ABO had the best mean offline-performance, and the BO-based methods did better than the rest. We did not run ABO-t on this dataset because the model selection was done only once using hyperparameter optimization with 66% of the data. ABO-t requires model selection to take place in every step of the optimization for it to adaptively change the feasible region. If the model selection is done once, the resulting algorithm is essentially ABO-f.

Sparsification

This subsection includes experiments comparing the proposed sparsification approach with some baseline approaches. The experiment is set up to compare the prediction accuracy of different methods after they sparsify (approximate) their time-varying regression model using less data. We use the same standard optimization functions as in previous experiments where one of the input dimensions is selected as the temporal dimension.

The experiment was setup as follows. Each GP model starts out with 500 randomly initialised datapoints and at each fixed-frequency time step, a new randomly selected datapoint is added. At each step, the regression-approximation models are tested by how accurate they are at predicting 100 new randomly generated input test points corresponding to the next time step. This is repeated for at least 100 time steps for all the methods and standard functions. The comparative metrics used are the standardized mean squared errors (SMSE) and standardized mean log loss (SMLL) as described in (Ras-

Table 3.5: The standardized mean squared errors (SMSE) for various approximation methods on various time-varying regression problems.

Problem	Full	Age	Random	Ours
6C	0.0042	0.0041	0.0045	0.0043
Br	1.29e-05	6.52e-06	7.01e-06	6.32e-06
G-P	2.23e-04	2.56e-04	3.84e-04	2.75e-04
Gr	0.0035	0.0034	0.0014	0.0030
H3	0.0036	0.0039	0.0020	0.0040
H6	1.34e+24	1.53e+24	1.53e+24	1.28e+24
She	0.5951	0.5898	0.6528	0.5872
Sty2	2.04e-05	2.87e-05	6.34e-05	2.90e-05
Sty7	1.0971	1.0903	1.1732	1.0867

Table 3.6: The standardized mean log loss (SMLL) for various approximation methods on various time-varying regression problems.

Problem	Full	Age	Random	Ours
6C	-2.6657	-2.6632	-2.6617	-2.6770
Br	-6.6342	-6.7499	-6.7489	-6.6558
G-P	-2.47e+10	-2.47e+10	-2.47e+10	-2.47e+10
Gr	-2.7602	-2.7604	-2.7585	-2.7535
H3	-2.3127	-1.9861	-2.4502	-2.0497
H6	-7.2021	-7.2052	-7.1701	-7.2060
She	-0.2532	-0.3691	-0.2799	-0.3803
Sty2	-1.01+03	-1.01e+03	-1.01e+03	-1.01e+03
Sty7	-4.07e+03	-4.07e+03	-4.05e+03	-4.07+e03

mussen & Williams 2006, §2.5). Each GP model used a squared exponential covariance function for both temporal and spatial covariance kernels.

The baseline approximation approaches that were tested are: (i) using the full set of data for inference (denoted by 'Full' in the results); (ii) selecting the most recent M datapoints out of the full N datapoints (denoted by 'Age' in the results), and (iii) randomly selecting M out of the N datapoints (denoted by 'Random' in the results). We denote our subset selection method as 'Ours' in the results. The experimental results are shown in Tables 3.5 and 3.6, showing the SMSE and SMLL, respectively. The results were averaged over 10 runs.

There are four interesting observations that can be made from the results

in the tables. First, the performance of our method is very similar to the one that picks the most recent datapoints on all the lower dimensional problems. Both our method and the 'Age' method were consistently top performers on the lower dimensional problems (except for the Griewank function where 'Random' selection beat all the methods.). For the higher dimensional problems, our method proved slightly superior than the 'Age' method.

Second, the 'Full' baseline that uses all the available data performed poorer than the other methods for some of the functions. This observation demonstrates that the time-varying nature of the underlying model requires the need to discard part of the dataset when performing regression for future time steps.

Third, the 'Random' approach that selected datapoints randomly performed well on the lower dimensional problems and relatively well on the higher dimensional problems. For the lower dimensions, this makes sense because having 500 training randomly selected points provides good coverage of the design space for the model to learn enough about the variations over time.

Fourth, the differences in performance on the SMLL metric between all four alternative methods is relatively small, which shows that the predictive distribution induced by the methods have similar losses on the test data.

In conclusion, our proposed method behaves similarly to selecting the most recent datapoints for lower dimensional problems and provides a marginally better performance for higher dimensional problems. This makes sense considering that we used a simple stationary covariance function to model both spatial and temporal patterns whose distance measurement is directly correlated to the age of the datapoint (i.e., we used the squared exponential covariance function). For more sophisticated temporal covariance functions, such as those considering anti-correlations, this result may have not been the case. The results of our method on higher dimensional problems demonstrated the efficacy of using the SNR selection criteria for subsets of data. However, when the computational cost is taken into account, picking the most recent datapoints can be a better choice, especially for lower dimensional problems.

Final Remarks

The experiments show that ABO performs well on problems with different kinds of dynamics such as those where the time variations are smooth and those where the changes are abrupt. While the competing approaches are good optimization methods under the right conditions, they suffer because they either do not take advantage of time variations, or rely on having numerous function evaluations to get good performance. Given that we are looking at problems where function evaluations are expensive, having good performance with fewer evaluations is desirable.

Efficient learning of the dynamic optimization problem explains the performance of adaptive Bayesian optimization in these experiments. The ability to exploit available prior information and incorporate it into the optimization of the acquisition function demonstrated that our proposed extensions to Bayesian optimization allow for efficient tracking of the minimum.

Additionally, the exploitation of the temporal relationships to learn how the DOP evolved gave the advantage of being able to determine when to make evaluations, how fast to make those evaluations and induced an appropriate budget of steps based on the available information and a user-defined threshold.

3.8 Summary

We have proposed practical extensions of Bayesian optimization for dynamic problems that are latent and expensive. Our resulting method has clear advantages over previous work as demonstrated by experiments on synthetic and real-world problems. There is an additional benefit of being able to schedule optimal evaluations automatically. Our formulation of the DOP as a spatiotemporal GP proved to be an appropriate prior for finding and efficiently tracking the minimum of dynamic problems.

4

Intelligent Metaheuristics for Learning Rate Adaptation

What we know is not much. What we do not know is immense.
— Pierre-Simon Laplace, 1827

The performance of the vanilla stochastic gradient descent algorithm is critically dependent on the chosen initial global learning rate parameter. A remedy for this, which has led to many modifications of the stochastic gradient descent algorithm, involves tuning the learning rate parameter over time. In this chapter, we propose another method for automatically and adaptively tuning the global learning rate of the vanilla stochastic gradient descent algorithm to solve the problem caused by pathologically selected initial learning rates. The goal of this chapter is to demonstrate the ubiquity of the adaptive Bayesian optimization algorithm for automatic algorithm configuration in the context of stochastic gradient descent. We model the set of training curve functions that map the global learning rates to epoch-based average training losses during the training using a spatiotemporal Gaussian process prior. Additionally, since hyper-gradients of the training curve functions are readily available in this

problem, we use a multi-task approach to model them using another spatiotemporal Gaussian process. This additional derivative Gaussian process improves the learning of the training curve maps by exploiting the commonalities and differences across these processes. We then use the adaptive Bayesian optimization mechanism to tune the global learning rate by intelligently searching these models for suitable values. Since stochastic gradient descent is one of the most commonly used training algorithms for large-scale learning systems, we test the resulting method on the training of learning models of varying complexity on widely-used datasets.

4.1 Introduction

Stochastic gradient descent (SGD) is one of the most popular algorithms for training many types of learning systems (Bottou 2012, §2.2). One major challenge of using this algorithm comes with adapting the global learning rate for faster convergence and for finding better local minima.

Many algorithms tackling this problem have been proposed. One approach has been to search for the best schedule for reducing the global learning rate over time. Learning rate schedules for the stochastic gradient descent algorithm operating on stationary data are usually decreased based on a schedule of the form $\eta(t) = \eta_0 (1 + \gamma t)^{-1}$ (Robbins & Monro 1951, Moulines & Bach 2011, Xu 2011).

Researchers have also proposed methods for adaptively tuning per-parameter learning rates (see George & Powell (2006) for an overview). One early scheme proposed in Almeida et al. (1999) used diagonal preconditioning, where the learning rate of each parameter was updated using the gradient of the loss function with respect to the learning rate in that dimension. Another method called stochastic meta-descent (SMD) (Schraudolph 1999, 2002) used multiplicative learning rate updates. Approaches using natural gradients were also proposed in Amari et al. (2000) and Roux & Fitzgibbon (2010).

More recently, a class of popular per-parameter adaptive methods based on worst-case analysis have been proposed. They include optimization algorithms such as AdaGrad (Adaptive Gradients) (Duchi et al. 2011), RMSProp (Tieleman & Hinton 2012) and Adam (Adaptive Moment Estimation) (Kingma & Ba 2015), which use heuristics that exploit the loss function's geometry to update each parameter independently. These methods operate by taking advantage of local gradient variations at the evaluated points on the loss function. Using gradient variations to adjust the learning rate makes sense because this gradient information is readily available during the execution of SGD and it informs the algorithms on promising future directions for exploration. The main practical drawback with most of these schemes is that they still involve at least one sensitive hyper-parameter (the so-called 'pesky' global learning rate) that must be appropriately configured to obtain satisfactory performance. Some methods have been proposed to counter this, such as vSGD (variance-based SGD) (Schaul et al. 2013) and Hyper-gradient Descent (Baydin et al. 2017). However, the former has not been widely adopted due to either its complexity, practicality or performance relative to the previously described methods, and the latter ignores other available non-gradient information that can be exploited for updating the learning rate. For a detailed survey on stochastic gradient descent algorithms, see Section 4.2, Bottou (1998), Bousquet & Bottou (2008) and Ruder (2016).

In this chapter, we propose an intelligent meta-heuristic method to automatically adjust the global learning rate in a manner that maximally decreases the expected training loss for every epoch of training for vanilla stochastic gradient descent. We demonstrate that this mechanism leads to quicker convergence in a manner that can overcome ill-suited initial global learning rates. We do this by exploiting a spatiotemporal relationship between the temporal learning rates and the response variable of the underlying optimization problem (which is the average training loss for an epoch in our case). We then use adaptive Bayesian optimization (ABO) to sequentially configure the global learning rates

to improve the SGD optimization at every epoch. We also extend the model for dynamic optimization problems (DOPs) to include derivative information that is readily available in this problem.

The merit of our approach is that in addition to taking advantage of any existing gradient-based variations, it learns the time-varying relationship between the global learning rate and the response variable (the average training loss for an epoch) directly, therefore offering additional insights on SGD's performance that can be exploited for learning rate updates at future epochs.

This chapter is structured as follows. Section 4.2 described the background and Section 4.3 describes the problem we address and 4.4 delineates our solution for online adaptation of the global learning rate. Section 4.5 includes experimental analyses conducted on learning models of varying complexity operating on widely-used standard datasets. Section 4.5 also includes the relevant discussions.

4.2 Background

Gradient descent is one of the most popular optimization algorithms and is by far the most commonly-used method for optimizing deep neural networks. The technique aims to minimize an objective function $f(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^D$, by updating a potential solution in the direction of steepest descent defined by the negative gradient at a point, $-\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. We are interested in instances of this algorithm where the objective function is described in terms of training data, $f(\boldsymbol{\theta}; \boldsymbol{x}, y)$, like many training loss functions are. Here, $\boldsymbol{\theta}$ is the parameter of interest and $\{\boldsymbol{x}, y\}$ denotes the training data. There are many types of gradient descent optimization algorithms and we will briefly describe them below. For a more detailed overview, see Ruder (2016).

The purest form of the gradient descent algorithm is one that calculates its gradient using all the available data. This version is called *batch gradient*

descent. The update step is given by

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}), \tag{4.1}$$

where η is the learning rate.

This step is usually expensive, slow and intractable for cases where the whole dataset cannot fit into the memory of the computer implementing it. Additionally, this method is limited in how it can handle new datapoints on-the-fly.

As a result, in practice, the gradient $-\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$ is calculated in an incremental manner by considering its value at various points as determined by the available data in the application in question. The variant of the algorithm using this incremental approximation is called *stochastic gradient descent* (SGD), and its update step is described by

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i), \tag{4.2}$$

where $\boldsymbol{x}_i$ and y_i are training input and target data, respectively.

While batch gradient descent performs redundant computations because it recomputes gradients for similar examples for each update, stochastic gradient descent does away with this redundancy by performing one update at a time. It is, therefore, usually faster and can be used in an online manner. However, the updates usually have a high variance and this negatively affects convergence.

There is an approach that combines the benefits of both the batch and stochastic versions of the gradient descent algorithm, and it is called *mini-batch* gradient descent. This algorithm performs updates using a mini-batch of n training examples as described by the update rule

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}\,;\, \boldsymbol{x}_{i:i+n}\,,\, y_{i:i+n}). \tag{4.3}$$

Using mini-batches helps reduce the variance of the updates and leads to more stable convergence behaviour. In practice, stochastic gradient descent is the name also used to describe this algorithm, and this will be the case for the rest of this thesis.

Stochastic Gradient Descent Algorithms

The main challenges faced by stochastic gradient descent lies with two main issues: (i) the learning rate, and (ii) complex form of the objective function surface.

First, choosing a suitable learning rate is difficult because it should not be too small to cause slow convergence, or to large to prevent it. As a result, learning rate schedulers that adapt the learning rate during training are commonplace. Since they need to be predefined before the algorithm starts, the schedulers do not adapt the learning rate according to characteristics of the training data and perform even worse for sparse data (Robbins & Monro 1951, Darken et al. 1992).

Secondly, a key challenge is that using stochastic gradient descent to optimize highly non-convex surfaces make the optimization process prone to getting stuck in regions with sub-optimal local optima. For more complex forms, it has been argued that the problem is actually the existence of saddle-points that trap the stochastic gradient descent algorithm (Dauphin et al. 2014).

We will now describe a few stochastic gradient descent algorithms that address these issues, abbreviating the notation for the objective function $f(\boldsymbol{\theta}; \boldsymbol{x}_i, y_i)$ as $f(\boldsymbol{\theta})$:

- **Momentum** (Qian 1999). The stochastic gradient descent algorithm does not perform well in ravines because it oscillates between the slopes as it follows the direction of steepest descent but overshoots the base of the ravine. To address this, the momentum algorithms dampen these oscillations by adding a fraction γ of the update vector from the previous time-step to the current update vector

$$\begin{aligned} \boldsymbol{v}_t &= \gamma \boldsymbol{v}_{t-1} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \boldsymbol{v}_t, \end{aligned} \tag{4.4}$$

  where γ is usually set to 0.9 or a similar value. This modification results in faster convergence due to reduced trajectory oscillations.

- **Nesterov Accelerated Gradient (NAG)** (Nesterov 1983). One feature that the momentum algorithm lacks is the ability to pre-emptively speed-up and slow down the optimization in anticipation of changing slopes on the objective function surface. The Nesterov Accelerated Gradient algorithm deals with this nuance by anticipating the next momentum update via an approximation that self-corrects after the update step is completed. This anticipatory update is given by

$$\begin{aligned} \boldsymbol{v}_t &= \gamma \boldsymbol{v}_{t-1} - \eta \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta} - \gamma \boldsymbol{v}_{t-1}), \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \boldsymbol{v}_t. \end{aligned} \tag{4.5}$$

  This modified update rule prevents the algorithm from going too fast and it increases the responsiveness to gradient changes.

- **Adaptive Gradient (Adagrad)** (Duchi et al. 2011). One challenge that has not been addressed by the previously described algorithms is how to adapt the updates of each variable to perform large or small updates depending on the variable's importance. Adagrad addresses this by using a different learning rate for each variable, collated in learning rate vector $\boldsymbol{\eta}$, and modifies it using a gradient-dependent quotient and subsequently updates the parameter $\boldsymbol{\theta}$ at every step using the element-wise rule given by

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\sqrt{G + \epsilon}} \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}). \tag{4.6}$$

  G is a diagonal matrix with a sum of the squared gradients up to the current time-step, $\cdot$ is the element-wise product, and ϵ is a smoothing term that avoids division by zero and that is usually set to $1e$-8. A practical weakness of Adagrad is that for long runs, the accumulation of the squared gradients in the denominator can cause the learning rates to vanish, thus causing the algorithm to not learn anything at all.

- **Root Mean Square Propagation (RMSProp)** (Tieleman & Hinton 2012). To address this problem, RMSprop recursively defines the sum of

gradients as a decaying average of all past squared gradients, $\mathrm{E}(G)$. The update rule is therefore given by

$$\begin{aligned} \mathrm{E}(G) &= \gamma \mathrm{E}(G) + \eta\, G, \\ \boldsymbol{\theta} &= \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\sqrt{\mathrm{E}(G) + \epsilon}} \cdot \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \end{aligned} \tag{4.7}$$

where $\cdot$ is the element-wise product. The momentum term γ is often initialised to 0.9, and the default value of the learning rate η is usually set to 0.1.

- **Adaptive Moment Estimation (Adam)** (Kingma & Ba 2015). The Adam algorithm is another method that uses per-parameter learning rates during the updates like Adagrad and RMSprop. However, in addition to keeping track of an exponentially decaying average of past squared gradients v_t, the Adam algorithm also keeps an exponentially decaying average of past gradients m_t, as done for momentum, defined as

$$\begin{aligned} m_t &= \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\, \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}), \\ v_t &= \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\, (\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}))^2, \end{aligned} \tag{4.8}$$

which are then corrected for bias as follows

$$\begin{aligned} \hat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1 + \beta_1}, \\ \hat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1 + \beta_2}. \end{aligned} \tag{4.9}$$

These bias-corrected terms are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients, respectively, hence the name of the method. The update rule is therefore given by

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \frac{\boldsymbol{\eta}}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon} \cdot \hat{\boldsymbol{m}}_t. \tag{4.10}$$

Kingma & Ba (2015), the authors of the Adam algorithm, propose default values of 0.9 for the β_1 parameter, 0.999 for the β_2 parameter, and 10^{-8} for the ϵ parameter. They empirically demonstrate that Adam works

well in practice and compares favourably to other stochastic gradient descent algorithms with per-parameter learning-rate adaptation mechanisms. Adam is one of the most popular methods used for training neural networks due to its demonstrable positive utility in practice.

4.3 Problem Statement

Many learning systems seek to find a model best suited to a given task. For a particular inference problem, these learning methods search for the best model satisfying some optimality criterion. Each of the models is described by set of parameters $\boldsymbol{\theta} \in \mathcal{P}$, where $\mathcal{P} \subseteq \mathbb{R}^n$ is the space of model parameters. In practice, finding the best model involves describing a cost or loss function $\mathcal{L}(\boldsymbol{\theta};\, \mathcal{D})$ that measures how well a model with parameters $\boldsymbol{\theta}$ performs its prediction or classification task on a set of training data $\mathcal{D}$. For this chapter, we will consider the training cost or loss. Therefore, the optimal model is obtained by searching the parameter space $\mathcal{P}$ for a model that satisfies

$$\boldsymbol{\theta}^{\star} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\ \mathcal{L}(\boldsymbol{\theta};\, \mathcal{D}). \tag{4.11}$$

The training of these learning systems involves the estimation of model parameters using a training dataset. This estimation is usually done using the standard stochastic gradient descent algorithm or its variants. We will use regular SGD for our descriptions in this section as we are interested in vanilla SGD, and any variant amenable to the external tuning of the global learning rate.

The process of training these models involves passing the data in mini-batches through the SGD algorithm. Using mini-batches is practically more efficient and has been shown to yield better performance (Bengio 2012). At every training iteration i, the model parameters $\boldsymbol{\theta}$ are updated by the rule

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta \cdot \nabla_{\boldsymbol{\theta}}\, \mathcal{L}(\boldsymbol{\theta};\, \mathcal{D}), \tag{4.12}$$

where η is the learning rate that determines the degree to which the parameters will be updated in the direction of the optimal parameters. Based on practical experience, if the value of η is very large, the algorithm may skip over the best parameters, and if it is too small, then the algorithm may need many more iterations to converge to the best parameter values. So using an appropriate value of η is critical (Ruder 2016). A suitable value of the learning rate η depends on the geometry of the underlying training loss function $\mathcal{L}$, whose information is encapsulated by the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ and the curvature.

In practice, multiple passes of the data are made through the stochastic gradient descent algorithm. For each epoch, the data is segmented into mini-batches where the data is also shuffled. This shuffling is done to introduce independence between the data samples, thus reducing the variance of the solution, and making sure the learnt models remain general and overfit less. This practice has been shown to help the SGD algorithm converge quicker (Bengio 2012, §2.1).

In this chapter, we are interested in adaptively tuning the global learning rate at the end of each epoch to maximally decrease the training loss associated with the subsequent epoch of training. However, because each epoch comprises a set of N iterations whose operations provide a per-iteration training loss (described by the update rule in Equation 4.12), we need to identify an appropriate optimality measure. Since we are interested in the loss associated with a set of N iterations affiliated with an epoch t, the average training losses for those iterations is a good choice for a data quantity that characterizes the quality of the global learning rate used for that epoch. Therefore, our quantities of interest will be the average training loss l at epoch t given by

$$l_t = \frac{1}{N} \sum_{i=1}^{N} l_i,$$

and the associated learning rate η_t.

Moreover, we also have access to useful hyper-gradients $\nabla_\eta \mathcal{L}$, as proposed in Almeida et al. (1999) and more recently used in Baydin et al. (2017), which

can be obtained by applying the chain rule on the available $\nabla_{\boldsymbol{\theta}} \mathcal{L}$. Assuming a case for a mini-batch update i, we proceed as follows:

$$\begin{aligned} \nabla_{\eta}^{i} \mathcal{L} &= \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}_i;\, \mathcal{D}\right)}{\partial \eta} \\ &= \frac{\partial \mathcal{L}\left(\boldsymbol{\theta}_i;\, \mathcal{D}\right)}{\partial \boldsymbol{\theta}_i} \cdot \frac{\partial \boldsymbol{\theta}_i}{\partial \eta} \\ &= \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}_i;\, \mathcal{D}\right) \cdot \frac{\partial}{\partial \eta}\left(\boldsymbol{\theta}_{i-1} - \eta \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_{i-1};\, \mathcal{D})\right) \\ &= \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}_i;\, \mathcal{D}\right) \cdot \left(-\nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}_{i-1};\, \mathcal{D}\right)\right). \end{aligned} \tag{4.13}$$

We will abbreviate $\nabla_{\eta}^{i} \mathcal{L}\left(\boldsymbol{\theta}_i;\, \mathcal{D}\right)$ as ∇_{η}^{i}, a notation we will use for the rest of the chapter.

The associated average gradient over the epoch, which can be obtained using the sum rule of differentiation, is therefore given by

$$\nabla_{\eta}^{t} = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\eta}^{i}.$$

4.4 Proposed Solution

Our goal is to find and configure suitable global learning rates for all epochs during the training process. The challenge is that the suitable learning rates are not only related to the geometry of the underlying loss function, but also the manner with which the data is presented to the SGD algorithm during a training epoch, and the stage of the training process. This set of challenges is further complicated by the fact that these three factors interact in ways that are difficult to describe rigorously. So to address these challenges, we first notice that we have the following available data

$$\Big[\, \eta_j, j\, ,\, l_j \,\Big]_{j=1}^{t},$$

where η_t and l_t are the learning rates and average training losses at epoch t. If we consider that the suitable learning rate is the minimum of some objective function, then we can re-describe our goal as keeping track of the minimum of

this function throughout the epochs of training. We can use the available data to model this objective function as a relationship of the form

$$f : (\eta, t) \mapsto l, \tag{4.14}$$

where learning rates $\eta \in \mathbb{R}$, epochs $t \in \mathbb{N}_+$ and average training losses $l \in \mathbb{R}$. This objective function f is:

- *Expensive to evaluate*: because each evaluation involves running an entire epoch of training data through the SGD algorithm;
- *Noisy*: the data is presented to the SGD algorithm via shuffled mini-batches that are associated with some variance;
- *Unknown*: we have no explicit characterization of the relationship between the three variables of interest.

These features make the problem of tracking the minimum of f amenable to adaptive Bayesian optimization. As prescribed in Chapter 3, we start by modelling f using a spatiotemporal Gaussian process prior with a kernel of the form

$$K(\{\eta, t\}, \{\eta', t'\}) = K_{\mathrm{L}}(\eta, \eta') \times K_{\mathrm{T}}(t, t') \tag{4.15}$$

where K_{L} is the spatial learning rate kernel and K_{T} is the temporal epoch-based kernel. We now discuss the choices for each covariance function in turn.

A Spatial Covariance Function for Training Curves

Since we want to capture correlations that are both smooth and non-smooth, a suitable choice for the spatial covariance function is the sum of a squared exponential and Matérn $\frac{1}{2}$ covariance function given by

$$\begin{aligned} K_{\mathrm{L}}(\eta, \eta') &= K_{\mathrm{SE}} + K_{\mathrm{M12}} \\ &= \sigma_{\mathrm{f1}}^2 \exp\Big(-\frac{(\eta - \eta')^2}{2l^2}\Big) + \sigma_{\mathrm{f2}}^2 \exp\Big(-\frac{|\eta - \eta'|}{l}\Big) + \sigma_{\mathrm{n}}^2. \end{aligned} \tag{4.16}$$

A Temporal Covariance Function for Training Curves

Prior experience with training curves for most models shows that the curves trend as an exponential decay towards some asymptotic final value if the model is being trained well with a good learning rate as shown in Figure 4.1.

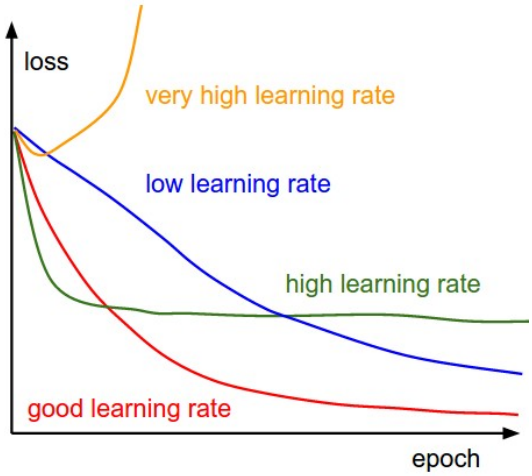

Figure 4.1: Graphical illustration of the quality of the learning process associated with different learning rates. Source: Karpathy (2014).

This decay is of the form $e^{-\lambda t}$, where $t, \lambda \geq 0$. As done in Swersky et al. (2014), we can synthesize a covariance function that captures a wide range of training curve behaviours for values of λ from 0 to ∞. This covariance function is given by

$$\begin{aligned} K_{\mathrm{T}}(t,t') &= \int_0^\infty e^{-\lambda t}\, e^{-\lambda t'} \psi(d\lambda) \\ &= \int_0^\infty e^{-\lambda(t+t')} \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\lambda\beta}(d\lambda) \\ &= \frac{\beta^\alpha}{\Gamma(\alpha)} \int_0^\infty e^{-\lambda(t+t'+\beta)} \lambda^{\alpha-1}(d\lambda) \\ &= \frac{\beta^\alpha}{(t+t'+\beta)^\alpha} \end{aligned} \tag{4.17}$$

where ψ is a a non-negative mixing measure on $\mathbb{R}_+$ having the form of a Gamma distribution with density

$$\psi(\lambda) = \frac{\beta^\alpha}{\Gamma(\alpha)} \lambda^{\alpha-1} e^{-\lambda\beta},$$

with parameters $\alpha, \beta > 0$, and where $\Gamma(\cdot)$ is the Gamma function.

Incorporating Gradient Information

Given that the appropriate learning rate for a particular epoch is related to the geometry of the underlying loss function, taking advantage of the available gradient information can only improve the search for good learning rates. When we include the gradients into our arsenal of available data, we have the following data quantities at our disposal

$$\Big[\, \eta_i, i, l_i, \nabla_\eta^i \,\Big]_{i=1}^{t},$$

where ∇_η^t is the average gradient with respect to the learning rate at epoch t (as described in Section 4.3)[1]. Therefore, we can define two relationships

$$f_{\mathrm{L}} : (\eta, t) \mapsto l_t, \qquad \text{for average training } \textit{losses} \text{ per epoch, as before, and,} \tag{4.18}$$

$$f_{\mathrm{D}} : (\eta, t) \mapsto \nabla_\eta^t \qquad \text{for average } \textit{derivatives} \text{ per epoch.} \tag{4.19}$$

These maps can then be combined into a multiple output relationship

$$f : (\eta, t) \mapsto (l_t, \nabla_\eta^t), \tag{4.20}$$

which we can model using a Gaussian process prior with a co-regional covariance of the form

$$\mathbb{C}\mathrm{ov}(\ f_{\mathrm{L}}(\eta, t),\ f_{\mathrm{D}}(\eta', t')\ ) = K(\ (\eta, t),\ (\eta', t')\ ) \times \mathrm{B}[L, D] \tag{4.21}$$

where B is a positive semi-definite matrix that encodes the dependencies among the outputs. It is of the form

$$B = WW^\top + \mathrm{diag}(\kappa),$$

where L and D are entry indicators which determine if the data belongs to the derivative or training loss relationship, and κ is heuristic diagonal jitter term.

[1] This not the gradient used in the inner loop of SGD, but an average of all the gradients from all the iterations within the specified epoch. See Section 4.3.

The benefit of using this model is that it improves the quality of the predictions for the training losses by using the gradient information. At prediction time during the evaluation of the acquisition function, we can select which output we want to predict for.

We use the same choice of spatial and temporal covariance functions for the derivative map due to the strong correlations between the training losses and gradients.

Learning Rate Adaptation

Now that we have a handle on a suitable meta-objective function, our goal is to exploit the relationship such that, at every epoch t, we perform the optimization

$$\eta_t^\star = \underset{\eta \in F(t)}{\arg\max} f(\eta, t \mid \mathcal{D}_t), \tag{4.22}$$

where $F(t)$ is the part of f's domain which is the searchable feasible set of global learning rates at epoch t. The resulting procedure is shown in Algorithm 5.

Algorithm 5 Global Learning Rate Adaptation with ABO-metaheuristics

Input: T: Number of epochs
Input: GP Prior $\mathcal{GP}(\mu, K)$ for $f : (\eta, t) \mapsto (l, \nabla_\eta)$
Output: $\{\eta_1, ..., \eta_\mathrm{T}\}$ trace of suitable learning rates
Output: $\{l_1, ..., l_\mathrm{T}\}$ trace of training losses
1: **for** $t = 0...T$ **do**
2: $\quad \eta_t = \underset{\eta \in F(t)}{\arg\max}\ a(\eta, t \mid \mathcal{D}_t)$
3: $\quad$ Run training epoch with global learning rate η_t
4: $\quad$ Update data
5: $\quad$ Train GP model using hyperparameter optimization
6: **end for**

4.5 Experiments

We evaluated the impact of the proposed method on vanilla stochastic gradient descent and a variant of the algorithm amenable to the external tuning of the global learning rate. Therefore, the optimizers to which we applied the global

learning rate adaptation are the standard SGD method and SGD with Nesterov Momentum (NEST). The experiments were conducted with these methods operating on several learning models of varying complexity. The metaheuristic-assisted versions (with hyper-gradients) of these methods are denoted with a suffix '-M' (which stands for *metaheuristics*). The comparison experiments were performed on well-known benchmark problems for digit recognition and image classification on both convex and non-convex models, which we describe in the next subsections.

Datsets

We considered two widely-used standard datasets for our experiments. The first is the MNIST digit recognition dataset (LeCun et al. 2010) that consists of 60,000 training examples. It describes an Arabic numeral classification problem with ten classes for each of the digits from 0 to 9. The vectors of input data were of size 784. The second dataset is the CIFAR-10 tiny natural image dataset (Krizhevsky et al. 2014) that consists of 60,000 32×32 tiny colour images comprising of ten object classes, with 6000 images per class.

Learning Models

We conducted the experiments on the following learning models:

- A simple softmax model (a neural network with no hidden layer) with a convex loss on a classification task on the MNIST data;
- A multi-layer neural network, also called a multi-layer perceptron (MLP), with one fully-connected hidden layer with 1000 nodes and a ReLu activation for a classification task on the MNIST data;
- A multi-layer neural network with two fully-connected hidden layers with 1000 nodes, each with a ReLu activation for the classification task on the MNIST dataset;

- The LeNet-5 convolutional neural network (LeCun et al. 1998), whose architecture is shown in Figure 4.2, for a classification task on CIFAR-10 dataset.

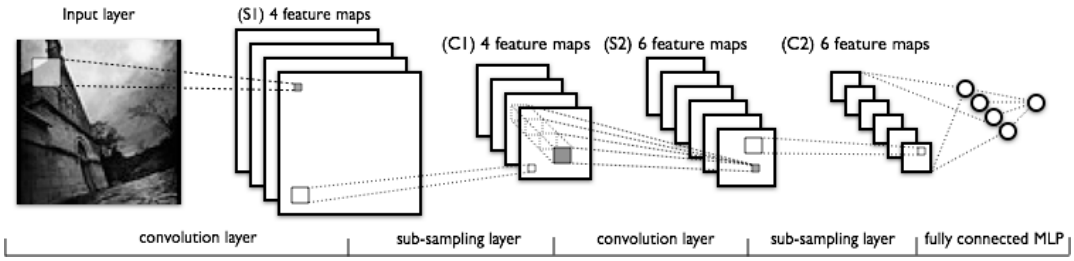

Figure 4.2: Architechure of LeNet-5 Convolutional Neural Network. Source: LeCun et al. (1998).

Formally, a neural network is a composite function $g = j_{\mathrm{H}} \circ ... \circ j_1$ that sequentially processes data $\boldsymbol{x}$ through H hidden layers by applying affine transformations j and an activation $A(\cdot)$, i.e.,

$$y = A(\ W_k \boldsymbol{h}_k + b_k\ ), \qquad\qquad k = 0, 1, ..., H-1.$$

The network output

$$y = W_{\mathrm{H}} \boldsymbol{h}_{\mathrm{H}} + b_{\mathrm{H}}$$

is then fed into a loss function. A commonly-used loss function for classification problems is cross-entropy. It approximates the probability of a class label belonging to a true target class c, and it is described as

$$\mathbb{E}[\ KL(\delta_c || p_y)\ ] = \mathbb{E}[\ -\log(p_y(c))\ ],$$

where δ_c is the target distribution to approximate and

$$p_y = \frac{\exp^{-y(c)}}{\sum_k \exp^{-y(k)}}.$$

Testing Methodology and Implementation Details

For the experiments, we ran the standard methods with different global learning rates from the set $\eta = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. We also ran a version of the methods with an exponential learning rate schedule, parameterized

by with $\gamma = 0.1$ and an initial global learning rate of $\eta = 10^{-1}$. As a benchmark, we used the popular Adam optimizer which performs per-parameter updates by exploiting the training loss function's geometric information with the recommended initial global learning rate of $\eta = 10^{-3}$.

The metaheuristic-assisted methods had their global learning rates changed at every epoch with the first three values being selected from the range $\{\eta : 10^{-5} \leq \eta \leq 10^{-3}\}$ to aid the initial training of the underlying Gaussian process. During the rest of the epochs, the configured learning rates were those proposed by our method.

We used Bayesian optimization with the lower confidence bounds acquisition function (Srinivas et al. 2012). The Gaussian processes and Bayesian optimization methods were implemented using codes from the GPy Gaussian process library (GPy 2012). The neural networks were implemented using the PyTorch machine learning library (Paszke et al. 2017), where necessary modifications to the optimization methods in the *torch.optim* package was made.

For all tests, we used the cross-entropy loss function, mini-batches size of 128 and L2 regularization with 10^{-4}. We also used a momentum factor of $\mu = 0.9$ for the Nesterov accelerated gradient method.

Softmax Model

We fit a logistic regression classifier to the MNIST data where we assigned probabilities of membership to the ten classes. Figure 4.3 shows the general behaviour of the various algorithms over 25 epochs. We observed the metaheuristic-assisted methods consistently brought the training loss closer to the optimum in all cases regardless of the assigned initial global learning rate values, with the aggressiveness of the method tapered off in the momentum variants by the momentum mechanism.

We also observed that the standard algorithms with smaller global learning rates in the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$ did better than the others, behaving consis-

tently with the common wisdom that learning rates that are too large or too small are not optimal.

Multi-layer Neural Network

We also tested the effectiveness of our method on training multi-layer neural networks on the MNIST dataset. Figures 4.4 and 4.5 show that our method proposes global learning rates that result in training losses that are close to the optimum value. As with Softmax regression, runs of the standard methods with learning rates from the set $\{10^{-1}, 10^{-2}, 10^{-3}\}$ performed better. In all these cases, our algorithm demonstrated its ability to adaptively tune the global learning rate in a manner that is commensurate with the need to maximally reduce the training loss in every epoch.

Convolutional Neural Network

Lastly, to investigate the performance of our method on deep architectures, we tested our method on training the LeNet convolutional neural network on the CIFAR-10 dataset. Figure 4.6 shows the traces of the training loss over 100 epochs that demonstrated that our algorithm consistently tunes the global learning rate to result in training losses close to the optimum. For a closer inspection of the training curves due to their longer trajectories, a magnified version of the training loss traces is shown in Figure 4.7.

A Remark on Adam

The Adam optimizer slightly converged quicker than our proposed method in all the experiments. Adam, an algorithm that performs per-parameter learning rate updates in every iteration, was at an advantage against our approach that only tunes the global learning rate every epoch. Considering that the difference in convergence performance is not too great, and that Adam has an advantage, the results demonstrate the ability of our algorithm to make good use of the

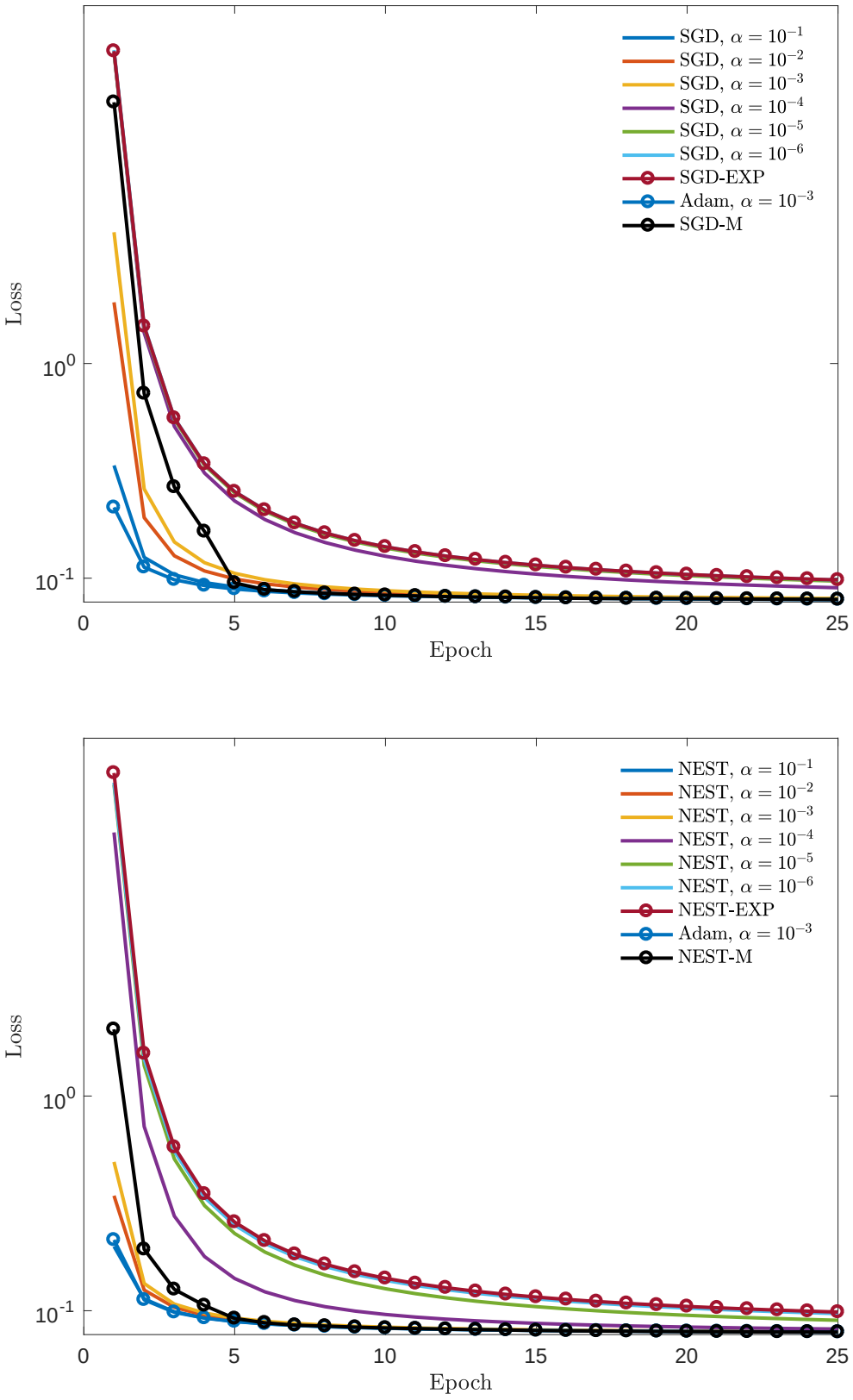

Figure 4.3: Adaptive tuning of the global learning rate for softmax regression on MNIST data.

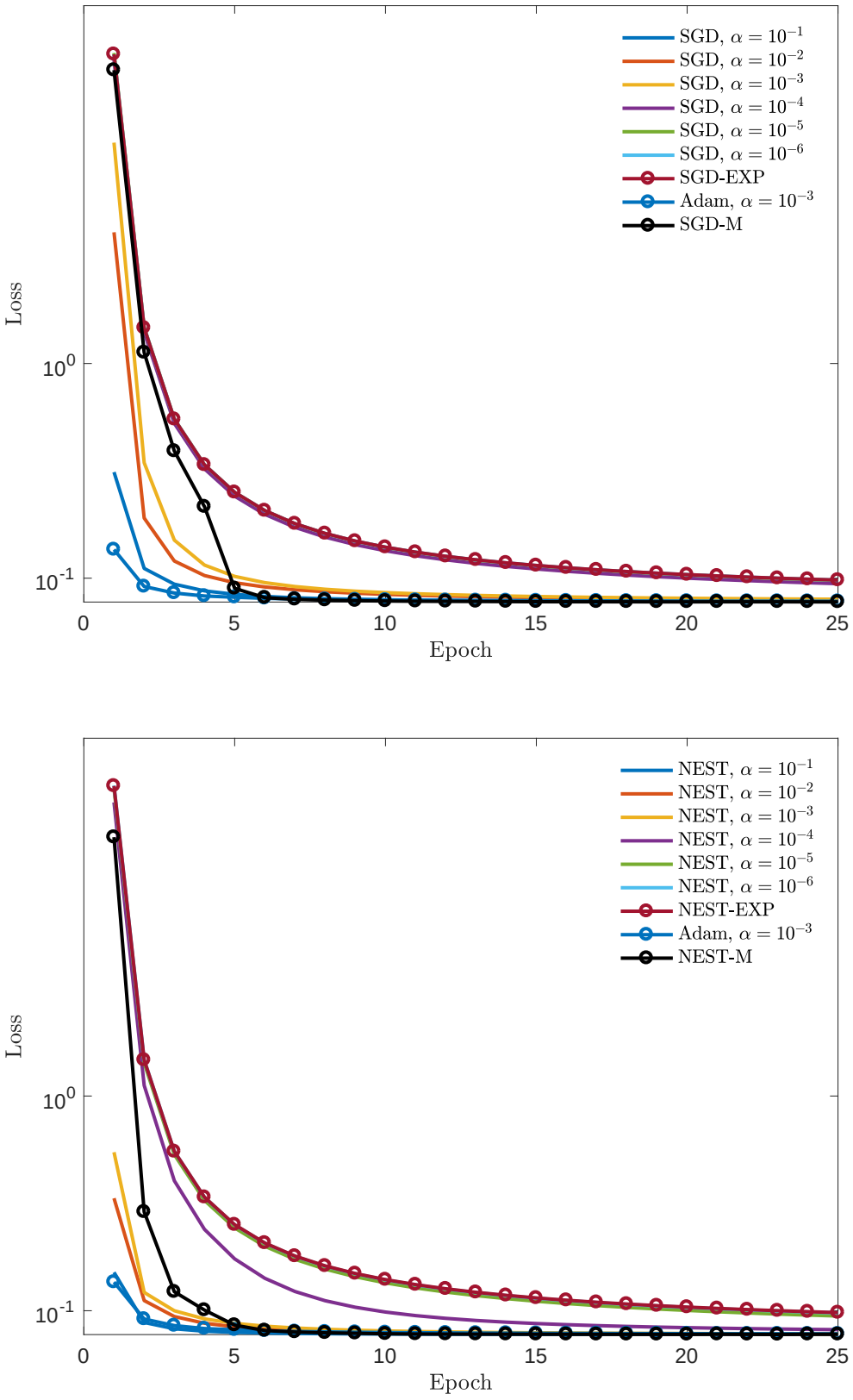

Figure 4.4: Adaptive tuning of the global learning rate for the fully-connected network with one hidden layer on MNIST data.

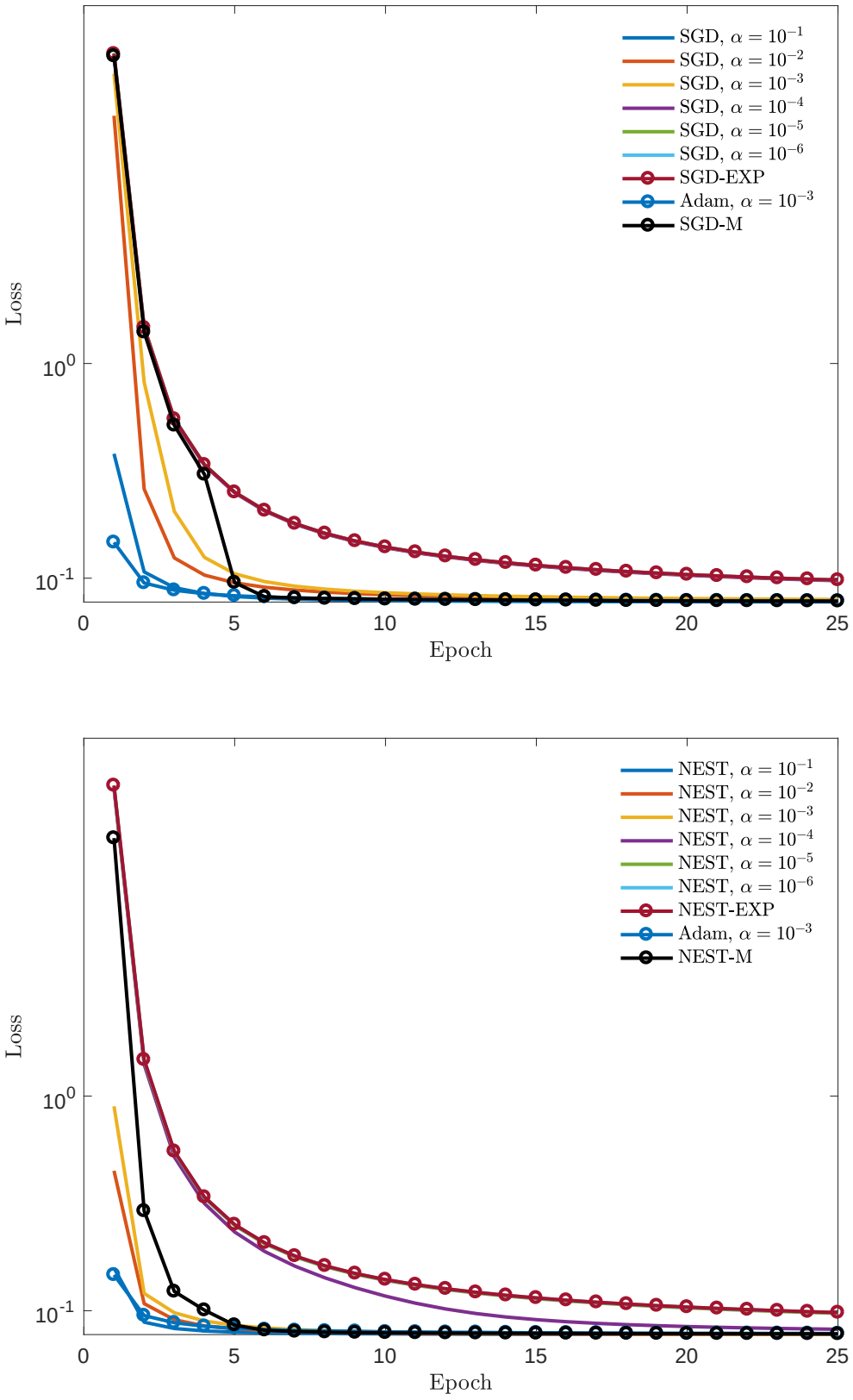

Figure 4.5: Adaptive tuning of the global learning rate for fully-connected network with two hidden layers on MNIST data.

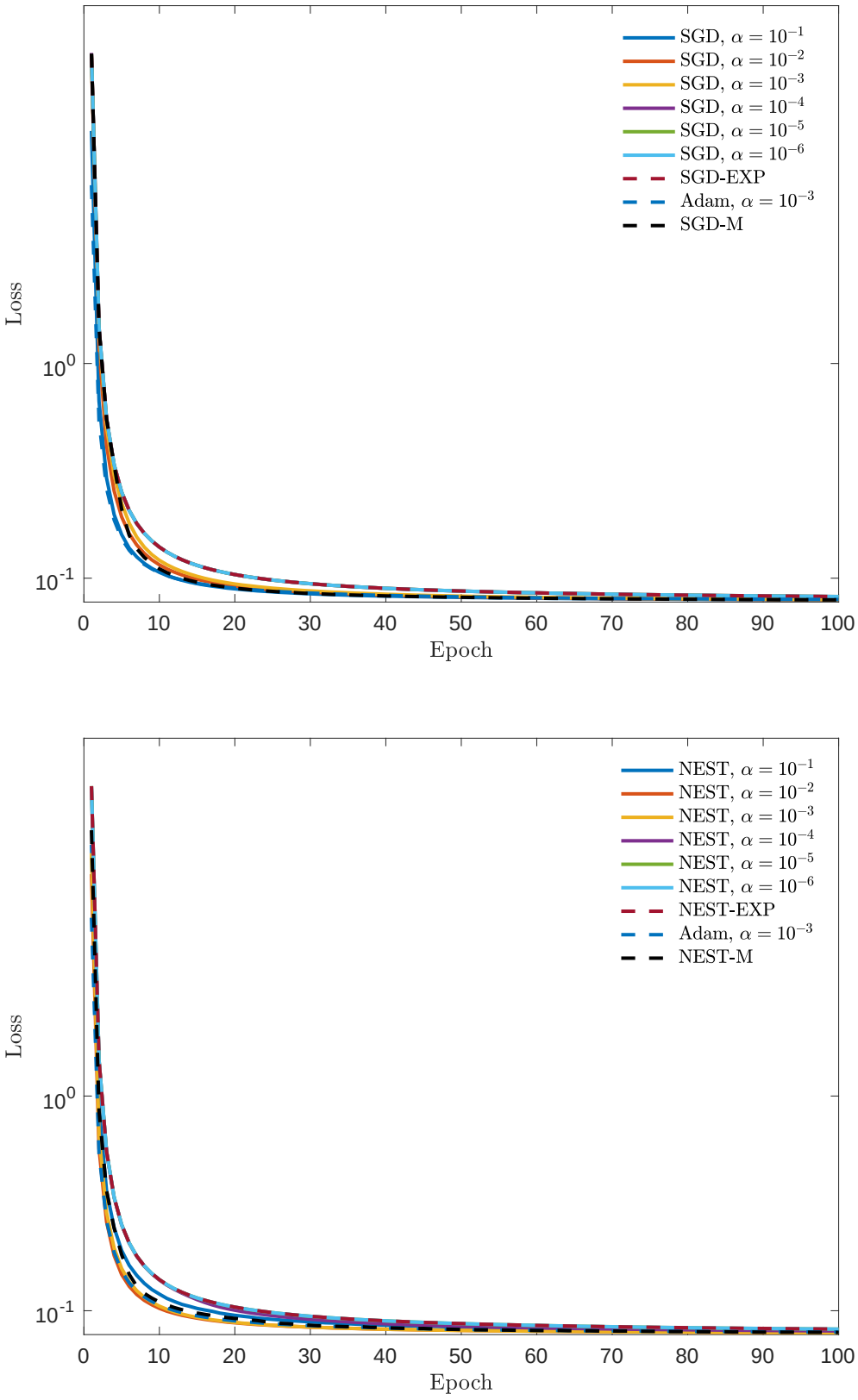

Figure 4.6: Adaptive tuning of the global learning rate for convolutional neural network (LeNet) on CIFAR-10 data.

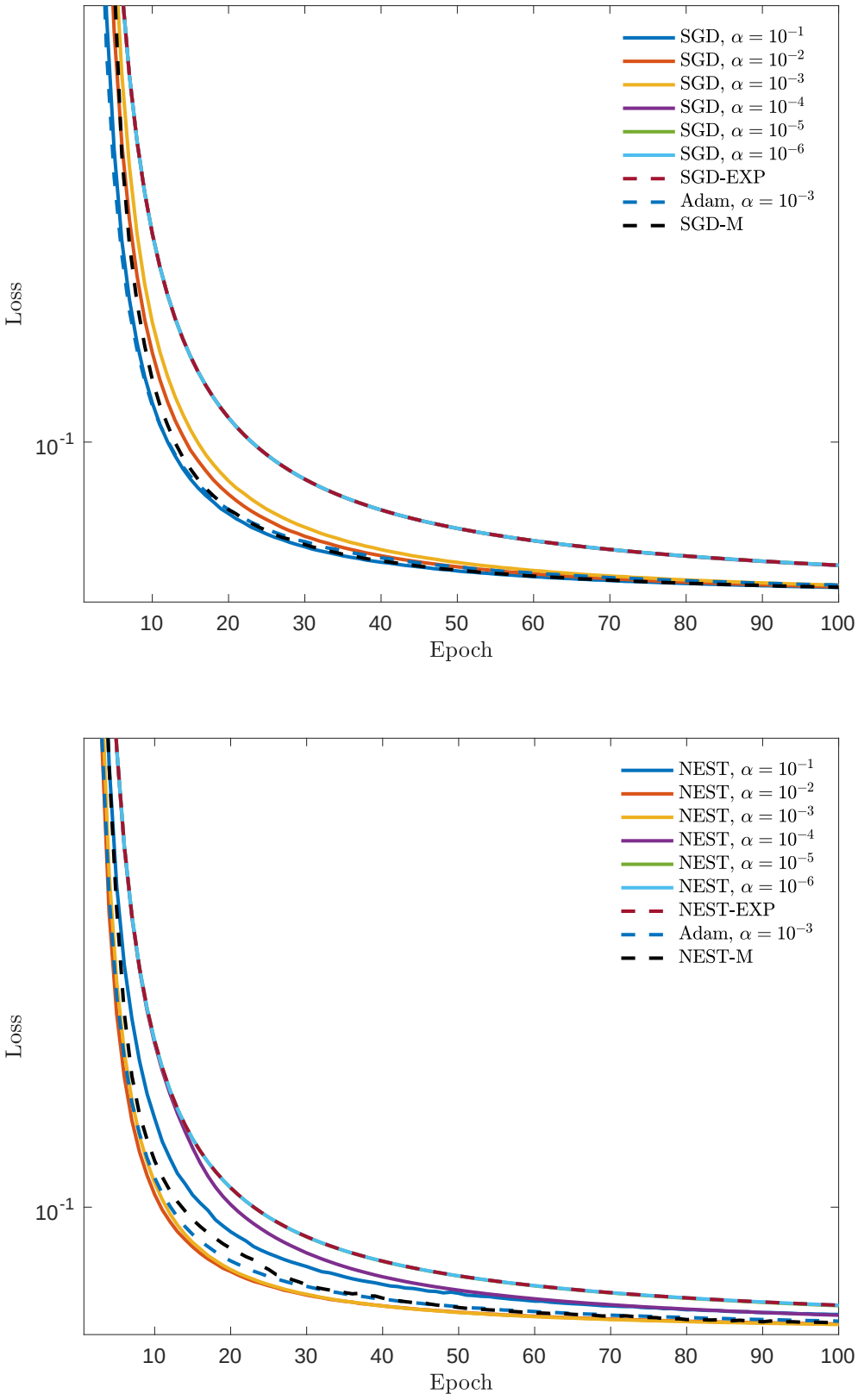

Figure 4.7: Magnified plots of the results for convolutional neural network on CIFAR-10 data.

learnt model to optimally and automatically select good global learning rates for vanilla SGD.

Final Remarks

While Bayesian optimization is a powerful method for performing the search for the extremum of latent objective functions, the goal of our problem is keeping track of it. As stated in Chapter 3, the need to ensure that the exploration steps do not drastically interfere with our intention of tracking the optimum is essential, and even more crucial in this learning rate problem.

One way of doing this is ensuring that we set bounds on the non-temporal variables that are motivated by the prior art to ensure that the ABO algorithm does not stray too far in its exploration steps. For learning rates, we know that values that are too large or too small are sub-optimal, so setting bounds to avoid the extreme values is reasonable. Doing this was valuable for our results.

An interesting avenue for future work is to adopt an automatic mechanism for intelligently learning these constraints such as the method of safety constraints in Berkenkamp et al. (2016), except our 'safety' characterization would be about parts of the search space that would lead to adverse tracking behaviour.

Another avenue for future work is to use our algorithm to perform the learning rate tuning using the generalization error rather than the training error as done in this chapter. As generalization error is a more useful metric to measure the performance of a machine learning model in practice, doing this would make our method more useful.

4.6 Summary

We presented an intelligent metaheuristic approach based on adaptive Bayesian optimization for global learning rate adaptation in vanilla stochastic gradient descent to solve the problem of pathologically chosen initial learning rates. We

extended adaptive Bayesian optimization to consider derivative information using co-regional spatiotemporal Gaussian process priors for the surrogate model. Lastly, we empirically tested the method on the problem of adaptively tuning the global learning rate for the standard stochastic gradient descent algorithm and the version with Nesterov accelerated gradients. We demonstrated that the resulting algorithm adapts the learning rate well to achieve competitive training error rates, beating methods not using the meta-heuristic adaptation scheme.

5

Adaptive Configuration Method for Online Portfolio Selection Methods

We can only see a short distance ahead, but we can see plenty there that needs to be done.
— Alan Turing, 1950

Financial markets are complex environments that produce enormous amounts of noisy and nonstationary data. One fundamental problem in quantitative finance is online portfolio selection, the goal of which is to exploit this data to sequentially select portfolios of assets to achieve positive investment outcomes while managing risks. Various algorithms have been proposed for solving this problem in fields such as finance, statistics and machine learning, among others. Most of the methods have parameters that are estimated from backtests for good performance. Since these algorithms operate on nonstationary data that reflects the complexity of financial markets, we posit that adaptively and intelligently tuning these parameters is a remedy for dealing with this complexity. In this paper, we model the mapping between the parameter space and the space of performance metrics using a Gaussian process prior. We then propose a method based on adaptive Bayesian optimization for automatically and

adaptively configuring online portfolio selection methods. We test the efficacy of our solution on algorithms operating on equity and index data from various financial markets.

5.1 Introduction

There has been a resurgence of interest in portfolio choice problems over the last decade due to the availability of large amounts of financial data and cheap computing power. The empirical evidence from the accumulated data has motivated trading strategies that attempt to exploit the behaviour of predictable components in the data. One variant of portfolio selection that has become increasingly popular is the online portfolio selection (OLPS) problem, where one seeks to sequentially select portfolios over a fixed time horizon to maximize some investor-specified criterion. For a survey of online portfolio selection techniques, see Section 5.2 and Li & Hoi (2014).

Most online portfolio selection methods estimate their parameters from backtests on data, or by hand-tuning. These parameters are then used for the rest of the out-of-sample trading periods. However, there are some limitations of using this approach for parameter estimation.

First, the methods used to estimate parameters tend to involve a lot of human intervention and are often time-consuming. For cases where the algorithm is particularly sensitive to the parameters, the configuration activity becomes a critical part of determining whether a method will succeed, often overriding the importance of the underlying assumptions of the strategy.

Secondly, financial data is often nonstationary, and because online portfolio selection algorithms operate sequentially, the nonstationarity in the data affects the effectiveness of the methods. An algorithm configured on backtested data may start out doing well on out-of-sample scenarios, but its performance may deteriorate after several trading periods.

Lastly, because trading on a financial market is a costly endeavour, the nonstationarity in the market data affects the trading efficacy and introduces new risks relating to the parameters used for the portfolio selection method. Since these risks are not managed directly or adequately exploited in many of the existing approaches, they can negatively affect the trading algorithm's performance.

We address these issues by utilizing a Bayesian nonparametric approach to modelling a relationship between the parameters and the performance criterion in every trading period using a spatiotemporal Gaussian process (GP) prior. We then propose a mechanism for using adaptive Bayesian optimization for tuning the parameters of online portfolio selection algorithms. The technique enables the methods to adapt to changing market conditions by having a handle on the uncertainty of the model, parameters and data. We consider a range of investment strategies driven by widely-accepted trading characteristics. Our formulation thus provides a method for tuning existing algorithms and has the advantage of delivering interpretable performance improvements. This interpretability offers useful algorithmic insights to practitioners.

This chapter is structured as follows. Section delves into the background, Section 5.3 outlines the problem setting and Section 5.4 describes the adaptive configuration method. Lastly, Section 5.5 includes experiments and discussions.

5.2 Background

Portfolio selection is the allocation of wealth for investment across a set of financial assets with the goal of achieving some objectives, usually to increase profits and reduce the risk of losing money. Online Portfolio Selection (OLPS) concerns a specific aspect of portfolio selection where the goal is to choose optimal weights of asset allocations in a dynamic financial market over multiple investment periods (within a finite horizon). This problem is characterized by the feature that the relevant optimal portfolios change over time because the

underlying market is inherently dynamic. As a result, this financial market environment requires an investor to readjust and select optimal portfolios that adapt to the changing market conditions with the goal of acclimatizing to it.

Problem Setting

Our formalization of the online portfolio selection problem is based on the survey by Li & Hoi (2014). We consider a financial market with a universe of M assets. We invest our wealth in a smaller subset of m assets, where $m < M$, for a sequence of n trading periods. During this time, the market prices of the assets in the portfolio change (these price differences are called the asset *returns*). These price changes are represented by a return vector denoted by

$$\big\{\boldsymbol{r}_t \in \mathbb{R}_+^m : t = 1, 2, ..., n\big\}.$$

Element i of return vector $\boldsymbol{r}_t$, denoted as $r_{t,i}$, is the ratio of the asset price at the end of period t to the price at the end of the previous period $(t-1)$ for asset i. Thus, an investment in asset i in trading period t increases by a factor of $r_{t,i}$. Due to these market fluctuations, we need to select the relevant optimal portfolios for every trading period that we will denote using $\boldsymbol{x}_t$. This approach is different from other classical formulations found in financial engineering studies where they look at asset return signals for one asset, and they aim to predict future values. They do so to determine when to buy and sell that singular asset. In the online portfolio selection formulation, however, the goal is to optimize some criterion to select the best asset allocation for a set of assets, which is done for every investment period. The success of the algorithm is judged by how much money it makes at the end of the horizon of investment periods. The process by which online portfolio selection methods operate is described in Algorithm 6.

At the beginning of the period t, an investment is specified by a portfolio vector $\boldsymbol{x}_t$. Element i of this portfolio, denoted by $x_{t,i}$, represents the proportion of capital invested in the asset i. We assume a portfolio is self-financed and

Algorithm 6 Online Portfolio Selection

Input: $\{\boldsymbol{r}_1, ..., \boldsymbol{r}_n\}$ historical asset return data
Output: $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\}$ sequence of portfolios
Output: $\{\boldsymbol{x}_1^\top \boldsymbol{r}_1, ..., \boldsymbol{x}_n^\top \boldsymbol{r}_n\}$ sequence of portfolios profits/losses
Procedure:
Initialisation: $\boldsymbol{x} = \frac{1}{m}$
for $t = 1, 2, ..., n$ **do**
Investor learns portfolio $\boldsymbol{x}_t$
Market reveals returns $\boldsymbol{r}_t$
Portfolio incurs period profits/losses $\boldsymbol{x}_t^\top \boldsymbol{r}_t$
Investor updates portfolio selection model
end for

no short-selling is allowed. Therefore, the portfolio $\boldsymbol{x}$ satisfies the constraint that each entry is non-negative and all entries sum up to one, $\boldsymbol{x}^\top \mathbf{1} = 1$. A *portfolio strategy* represents the investments procedure from period 1 to n that generates a sequence of portfolios

$$\big\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\big\}.$$

In every trading period, an investment manager apportions capital as described by portfolio vector $\boldsymbol{x}_t$ at the beginning of the trading period, and holds the portfolio until the end of the investment period. For an asset return of $\boldsymbol{r}_t$, the associated wealth of the portfolio of assets will increase by a factor of $\boldsymbol{x}_t^\top \boldsymbol{r}_t$. This process is repeated for all trading periods. For easy analysis, it is assumed that the initial wealth is \$1. This assumption allows the total wealth at the end of the trading horizon to be calculated multiplicatively.

Additionally, the following domain-specific assumptions are made when conducting theoretical analyses of online portfolio selection methods:

1. **Transaction Cost**: we assume no transaction costs or taxes in the model;

2. **Market Liquidity**: we assume that one can buy and sell any quantity of any asset in its closing prices;

3. **Impact Cost**: we assume market behaviour is not affected by any portfolio selection strategy.

Taxonomy of Online Portfolio Selection Methods

Online portfolio selection algorithms fall into three main classes that we will briefly describe.

1. **Follow the Winner (FW)** algorithms aim to asymptotically achieve the same growth rate as that of an optimal strategy, which is often based on the description postulated by the Capital Growth Theory (Li & Hoi 2014). This theory focusses on multi-period portfolio selection and aims to maximize the expected growth rate of the returns generated by an investment strategy (Kelly 1956).

2. **Follow the Loser (FL)** methods transfer the wealth from winning assets to losers based on the assumption of mean reversion. This assumption seems contradictory to common sense but empirically often achieves significantly better performance.

3. **Pattern Matching (PM)** approaches predict the next market distribution based on a sample of historical data and explicitly optimize the portfolio based on the sampled distribution.

For performance comparisons, the most common benchmark approaches used in online portfolio selection literature are:

1. **Uniform Buy-and-Hold (UBAH)**: This benchmark uses an equally-weighted portfolio at the initial trading period and holds it for the whole trading horizon without re-balancing (the portfolio is implicitly changed over subsequent time steps following market fluctuations) (Li & Hoi 2014, Li et al. 2012).

2. **Best Buy-and-Hold (BBAH)**: In hindsight, based on known returns over the time horizon, we find the best portfolio to hold at the initial time to get the maximum cumulative wealth (Li & Hoi 2014, Li et al. 2012).

3. **Uniform Constant Rebalanced Portfolio (UCRP)**: This benchmark maintains an equally-weighted portfolio at each trading period in the finite investment horizon (by re-balancing it at every period) (Li & Hoi 2014, Li et al. 2012).

4. **Best Constant Rebalanced Portfolio (BCRP)**: This uses the known returns over the trading horizon to find the best constant rebalanced portfolio to maintain at every period that maximizes the final cumulative wealth (Li & Hoi 2014, Li et al. 2012), i.e. a fixed portfolio $\boldsymbol{x}$ that the strategy rebalances to after every step. The rebalancing is required because when the price changes occur, the portfolio weights change. BCRP aims to rebalance to the same portfolio weights in every period, where this fixed portfolio is calculated on the full data (assumes all future data is available).

The Table 5.1 below shows a summary of examples of algorithms that fall into the classes described above and their associated references.

Table 5.1: General classification of the state-of-the-art online portfolio selection algorithms and benchmarks.

Classification	Algorithm	Reference
Benchmarks	Equally weighted	
	Best-stock	
	Constant Rebalanced Portfolios (CRP)	Cover (1991)
Follow Winner	Universal Portfolios (UP)	Cover (1991)
	Exponential Gradient (EG)	Helmbold et al. (1998)
	Follow Regularised Leader (ONS)	Agarwal et al. (2006)
Follow Loser	Anti-Correlation (AntiCor)	Borodin et al. (2003)
	Passive Aggressive Mean Reversion (PAMR)	Li et al. (2012)
	Confidence-Weighted Mean Reversion (CWMR)	Li et al. (2013)
	Online Moving Average Reversion (OLMAR)	Li & Hoi (2012*b*)
Pattern Matching	Nonparametric Kernel (B^K)	Gyorfi et al. (2006)
	Nonparametric Nearest-Neighbour (B^{NN})	Gyorfi et al. (2008)
	Correlation-driven Nonparametrics (CORN)	Li et al. (2011)

5.3 Problem Setting

Let us consider an investment task over a complex financial market with m assets for T investment periods. In each period, the asset prices change, and this affects any investment in those assets by a factor proportional to the price changes (also called *returns*).

We describe an investment decision in every trading period using a *portfolio* of assets, mathematically specified by a portfolio vector $\boldsymbol{w} \in \mathbb{R}^m$, whose elements represent the proportion of wealth invested in each asset. We assume that a portfolio's investment is self-financed such that no margin or short-selling is allowed, i.e. there is no money market. So $\boldsymbol{w}$ is vector such that $\sum_{i=1}^{m} w_i = 1$ and $w_i \geq 0$.

An investment strategy is represented by a procedure that produces a sequence of portfolios, one for each trading period, over the horizon of investment periods. The strategy reinvests the portfolio after every period over the horizon. After T trading periods, a strategy is evaluated by how much money it has made or lost, how well it managed risk and how resilient it was to market

fluctuations. This procedure is shown in Algorithm 7.

Algorithm 7 Online Portfolio Selection (OLPS) Problem

1: **for** $t = 1, 2, ..., T$ **do**
2: Strategy learns portfolio $\boldsymbol{w}_t^\star$
3: Market reveals asset returns
4: Portfolio $\boldsymbol{w}_t^\star$ incurs period profits or losses
5: Strategy updates portfolio selection model
6: **end for**

We are concerned with the step that calculates the best portfolio for period t highlighted in Line 2, marked in blue, which we will define as

$$\boldsymbol{w}_t^\star = \arg\max \; g(\boldsymbol{w}_t; \mathcal{D}, \boldsymbol{\theta}), \tag{5.1}$$

where $\boldsymbol{w}_t$ is the portfolio at trading period t, $\mathcal{D}$ is a collation of all the market data and information from previous investment decisions, and $\boldsymbol{\theta}$ is a parameter vector for the strategy criterion function g.

The function g defines trading strategies that use market characteristics to exploit market conditions. These investment strategies are based on well-understood market phenomena and empirical observations. The strategies described by g depend on data $\mathcal{D}$ assembled from a myriad of sources, selected for how informative they are for unveiling exploitable trading opportunities. These strategies often have parameters $\boldsymbol{\theta}$ resulting from the construction of the method or theoretical analyses. These parameters often dictate the mode or degree to which certain strategy-centric operations are performed. Examples of these parameters include, but are not limited to, learning rates, regularization parameters, window sizes, scaling multipliers, tolerances and heuristic parameters.

The optimization of strategy g shown in Equation 5.1 is usually performed using standard off-the-shelf optimizers. This optimization seeks to use the rules of the strategy to find the best portfolio at time t when data $\mathcal{D}$ are encountered, all under the strategic context described by parameters $\boldsymbol{\theta}$. These parameters are usually estimated from backtests or diligently hand-tuned, both done to

select parameters with the best performance on past training data before the strategy is executed out-of-sample. These estimated parameters are then used for the optimization in Equation 5.1 in all the future trading periods. However, as shown by Algorithm 7, noisy and nonstationary market information is sequentially revealed to the strategy in every out-of-sample trading period. The problem that arises is that the conditions and assumptions that held true on the backtest data on which $\boldsymbol{\theta}$ was estimated on do not always hold true for future time periods. This discrepancy can negatively affect the algorithm's performance.

We assert that a suitable remedy for this predicament is to adaptively tune $\boldsymbol{\theta}$ in an online manner as it encounters new data. This configuration should be done with the goal of maximizing the utility of using a strategy g for selecting portfolios in each trading period. A suitable criterion for this utility is the unseen future return of the portfolio estimated by g. The difficulty with doing this is that the objective function that describes this scheme is

- *Unknown*: because we have no analytical form of the function, and it depends on unseen quantities whose relationship we have not yet discerned;
- *Expensive to evaluate*: because one has to trade on a financial market and incur some cost after deciding what parameters to use to obtain the corresponding value of the utility; and
- *Noisy*: as a consequence of being dependent on financial data which is notorious for having a low signal-to-noise ratio (Ghoshal & Roberts 2016).

In this chapter, we propose a suitable model for learning this objective function, and a mechanism for keeping track of the best parameters that maximize the utility of using a strategy g in all the trading periods. The main advantage of solving this problem is that it would provide an intelligent and effective method for making a strategy g acclimatize to different information regimes as it encounters new data.

5.4 Adaptive Configuration Method

In this section, we introduce our novel approach to adaptive parameter configuration for an online portfolio selection strategy. To address the problem introduced in the previous section, we start by proposing a *parameter configuration map* f that defines a relationship between the compact parameter space $\mathcal{P} \subseteq \mathbb{R}^D$, the space of trading periods $\mathcal{T} \subseteq \mathbb{N}_+$, and the space of performance metrics $\mathcal{M} \subseteq \mathbb{R}$. We define this map as

$$f : \mathcal{P} \times \mathcal{T} \mapsto \mathcal{M}, \tag{5.2}$$

where parameters $\boldsymbol{\theta} \in \mathcal{P}$ and performance metrics $m \in \mathcal{M}$. The idea behind this relationship is predicated on the notion that the sequential nonstationary data induces the temporal evolution of the best parameters of strategy g. This parameter configuration map f captures a temporal-parametric relationship that can be exploited to generate optimal parameters over a horizon of trading periods. Learning the map f would also allow us to gain insights into the behaviour of the underlying trading approach and exploit those insights to adaptively tune $\boldsymbol{\theta}$ for good performance.

We assume that the parameter configuration map f is of the functional form

$$f : (\boldsymbol{\theta}, t) \mapsto m, \tag{5.3}$$

where we take the metric m to be the strategy's return at period $t \in \mathcal{T}$ while operating with parameters $\boldsymbol{\theta}$. We can take $\log f$ be a sample from a zero-mean Gaussian process with covariance function K

$$\log f \sim \mathcal{GP}(\mathbf{0}, K). \tag{5.4}$$

We consider K to be a separable spatiotemporal covariance function given by

$$K(\{\boldsymbol{\theta}, t\}, \{\boldsymbol{\theta}', t'\}) = K_{\mathrm{P}}(\boldsymbol{\theta}, \boldsymbol{\theta}') \times K_{\mathrm{T}}(t, t'). \tag{5.5}$$

The parameter-space kernel K_{P} is taken to be a separable product Rational Quadratic (RQ) kernel

$$K_{\mathrm{P}}(\boldsymbol{\theta}, \boldsymbol{\theta}') = \sigma_f^2 \prod_{i=1}^{D} \left(1 + \frac{(\theta_i - \theta_i')^2}{2\alpha_i l_i^2}\right)^{-\alpha_i}, \tag{5.6}$$

where the hyperparameters $\sigma_f, l_i, \alpha_i > 0$. The choice for this kernel was inspired by the fact that we assume the assets that we trade are *liquid* (easily bought or sold), so the effect of any small variations in the proportion of a particular asset on the value of the portfolio will be smooth. Since each asset will have a different characterization of this smoothness, the RQ kernel provides the advantage of being a scaled mixture of the squared exponential (SE) kernel with varying length-scales (Rasmussen & Williams 2006, §4.2).

We take the temporal kernel K_{T} to be

$$K_{\mathrm{T}}(t, t') = \exp\left(-\frac{(|t - t'|}{l}\right) + \left(1 + \frac{(t - t')^2}{2\alpha l^2}\right)^{-\alpha}, \tag{5.7}$$

where $l_1, l_2, \alpha > 0$. This temporal kernel was selected to accommodate both abrupt and smooth temporal changes. The Matérn $\frac{1}{2}$ component captures the more abrupt changes and the RQ component captures the smoother ones.

Now that we have a description of the parameter configuration map, the next step is how to use it for adaptively tuning the settings. To choose the best configuration $\boldsymbol{\theta}$ for the online portfolio selection strategy g at every trading step t, we are interested in solving

$$\boldsymbol{\theta}_t^\star = \operatorname*{arg\,max}_{\boldsymbol{\theta} \in \mathcal{S}_t} f(\boldsymbol{\theta}, t), \tag{5.8}$$

where the parameter configuration map $f(\boldsymbol{\theta}, t)$ is the meta-objective function of the online portfolio selection strategy g.

To keep track of the best parameters using the meta-objective function, we resort to adaptive Bayesian optimization, which will allow us to choose *what* parameters to use in *every* investment period as the sequential OLPS algorithm encounters new data and changing market conditions. We make

use of the fixed-frequency evaluation version of adaptive Bayesian optimization where the times of interest are known. This is because we often know the trading frequency of the OLPS algorithms *a priori*. We intelligently evaluate the learnt Gaussian process model using smart search heuristics, which are encoded by the acquisition function $a(\cdot)$, that facilitate the selection of the best parameters to use in the relevant trading periods. The advantage of using predefined fixed-frequency iterations with an optimistic bandit-based search heuristic is that there are convergence guarantees as long as we evaluate f fast enough to capture changes as shown in Bogunovic et al. (2016). The resulting parametric method is highlighted in red in Algorithm 8, which shows an updated online portfolio selection procedure. The part highlighted in blue still shows where the OLPS algorithm learns the portfolio $\boldsymbol{w}^\star$ using strategy g.

Algorithm 8 Online Portfolio Selection with the Configuration Method

Input: T: number of trading periods in trading horizon
Input: $g(\cdot\,;\boldsymbol{\theta})$: OLPS strategy with parameters $\boldsymbol{\theta}$
Input: m: performance metric
Input: GP Prior $\mathcal{GP}(\mu, K_{\mathrm{P}} \times K_{\mathrm{T}})$ for $f : (\boldsymbol{\theta}, t) \mapsto m$
Output: $\{\boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_t\}$ trace of parameters, where $t \leq T$
Output: $\{m_1, ..., m_t\}$ trace of performance metrics, where $t \leq T$

1: **for** $t = 1, 2, ..., T$ **do**
2: Train GP model $\mathcal{GP}(\mu, K)$
3: Set bounds of $\mathcal{S}_t$ using information from $K_{\mathcal{T}}$
4: Select parameters $\boldsymbol{\theta}_t^\star$ from

$$\boldsymbol{\theta}_t^\star = \underset{\boldsymbol{\theta}_t \in \mathcal{S}_t}{\arg\max}\ a_{\mathrm{UCB}}(\boldsymbol{\theta}, t)$$

5: Strategy learns portfolio $\boldsymbol{w}_t^\star$ by

$$\boldsymbol{w}_t^\star = \arg\max\ g(\boldsymbol{w}_t; \mathcal{D}, \boldsymbol{\theta}_t^\star)$$

6: Market reveals asset returns
7: Portfolio $\boldsymbol{w}_t^\star$ incurs period profits or losses
8: Strategy updates portfolio selection model
9: Update data for GP model
10: **end for**

Strategic Insights

The most critical part of the adaptive configuration method is the parameter configuration map f. The importance of f is more ubiquitous than the configuration task as it generates insights about the OLPS problem in terms of the modelled parameters. Learning f leverages observable information and uncovers intricate patterns for a wide variety of OLPS strategies. What the method does in its configuration task is to use these insights to propose the best parameters as time-evolving market conditions are revealed to the OLPS strategy.

Using performance metrics as the output space of the configuration map is crucial for uncovering insights from the data. The performance metrics operate as an optimality criterion that discriminates good parameters from bad while encoding user-specific objectives. Many practitioners have excellent prior knowledge about the settings for their OLPS methods based on their experience of using them. This knowledge will usually include the details such as the optimal range of values for good performance and scenarios when particular configurations work excellently. All this information can be incorporated into the method via the specification of an appropriate Gaussian process prior and by setting constraints on the non-temporal variables during the acquisition function optimization.

The temporal length-scales provide additional insights on whether an OLPS approach would receive performance gains from the adaptive tuning of its parameters. If the learnt temporal length-scale is large when compared to the horizon of trading periods under consideration, then there is no time variation in the best parameters. On the other hand, if the temporal length-scale is small, then the best parameters evolve with time.

Since BO explores and exploits f to search for the best configurations, sub-optimal performance may be incurred in some exploration steps which can be costly in the OLPS setting. However, there is a remedy for this. The

practitioner-based prior art on OLPS configuration becomes useful in introducing constraints for the search that leads to less risky performance.

5.5 Experiments

In this section, we present an extensive set of empirical analyses of our configuration method and compare it with the recommended static settings of some popular online portfolio selection methods. For all the experiments, the portfolios are rebalanced on a daily basis. At the end of each trading day, we reconfigure the OLPS methods, and they calculate the target portfolio for the following market opening. The true market data is only revealed to the OLPS algorithms after the relevant portfolios have been constructed, so all the adaptive parameter configurations and portfolio calculations are performed out-of-sample. The method operates by picking daily configurations that are aimed at maximizing the daily return of the generated portfolio. We first describe the experimental protocols, the nature of the online portfolio selection algorithms that we configure and then delve into the results.

Comparison Measures

In our experiments, we compare various OLPS methods with their recommended settings as prescribed by their respective authors and variants adaptively tuned by our method. We compare the performance of the methods under criteria that fall into the following three classes:

- Absolute returns: How much money the strategy makes over a pre-defined time horizon;
- Risk: How much financial risk the strategy induces over that time horizon. It is often based on the standard deviation of the return of a strategy over the horizon; and

- Risk-adjusted returns: How profitable the strategy is when risk is taken into account.

The specific measures under these classes are shown in Table 5.2. We assume there are no transaction costs for all our experiments because we are only interested in gauging the performance improvements resulting from using our configuration method.

Table 5.2: Comparison measures used in the experiments and their classes.

Criteria	Measures	
Absolute return	Cumulative wealth	Annualized percentage yield
Risk	Annualized standard deviation	Maximum drawdown
Risk-adjusted return	Annualized Sharpe ratio	Calmer ratio

We briefly define the measures shown in Table 5.2 below:

- Cumulative wealth: This is the total wealth a strategy accumulates assuming an initial investment of $1. This is the most common measure for comparing different trading strategies, and we use it as our main comparator. In general, higher values of cumulative wealth indicate better trading algorithms. Since portfolio returns are multiplicative, the total wealth represents the multiple or percentage increment to the original wealth. This makes cumulative wealth interpretable.

- Annualized percentage yield: This is the annual rate of return that accounts for compounding interest. As with total wealth, higher values indicate better performance.

- Annualized standard deviation: This is the standard deviation of the portfolio returns multiplied by the square root of the number of trading periods in one year, and it is commonly used as a measure of risk. Since it has the same units as the returns, the annualized standard deviation is

ordinarily represented as a percentage. It is also called the *volatility risk*. Unlike cumulative wealth, lower values indicate better risk management.

- Maximum drawdown: A drawdown is a difference between the maximum and minimum monetary values of a portfolio during an investment period. The maximum drawdown is the largest difference in portfolio values over a time horizon. It is used to measure the downside risk of a trading strategy. The maximum drawdown is defined as:

$$\text{MDD} = \frac{v_{\text{highest}} - v_{\text{lowest}}}{v_{\text{highest}}}, \tag{5.9}$$

where v_{highest} is the highest value of the investment portfolio during the investment horizon, and v_{lowest} the lowest value in that same horizon. As with volatility risk, it is usually represented as a percentage, and lower values indicate good performance.

- Annualized Sharpe ratio: This is the average strategy return earned more than the risk-free rate per unit of total risk. This is a popular measure of risk-adjusted performance. In general, as with total wealth, higher values of the Sharpe ratio indicate better trading algorithms. The Sharpe ratio is defined as:

$$S = \frac{\mathbb{E}[r_s] - r_f}{\sqrt{\sigma_s}}, \tag{5.10}$$

where r_p is the strategy return, r_f is the risk-free return (the return from a benchmark risk-free investment such as a treasury bill) and σ_p is the standard deviation of the strategy returns.

- Calmer ratio: This measures a strategy's performance relative to its risk. It is calculated by dividing the average annual rate of return of the strategy by the associated maximum drawdown. It is also called the *drawdown ratio*. As with Sharpe ratio, higher values indicate better risk-adjusted performance. The Calmer ratio is given by:

$$C = \frac{\bar{r}_{\text{annual}}}{\text{MDD}}, \tag{5.11}$$

Table 5.3: Trading strategies used for experiments and their representative references.

Algorithm	Reference
Market	–
Best-stock (BS)	–
Best Constant Rebalanced Portfolios (BCRP)	(Cover 1991)
Exponential Gradient (EG)	(Helmbold et al. 1998)
Online Newton Step (ONS)	(Agarwal et al. 2006)
Passive Aggressive Mean Reversion (PAMR)	(Li et al. 2012)
Confidence-Weighted Mean Reversion (CWMR)	(Li et al. 2013)
Online Moving Average Reversion (OLMAR)	(Li & Hoi 2012*b*)

where $\bar{r}_{\text{annual}}$ the average annual rate of return and MDD is the previously defined maximum drawdown.

Tested Strategies

Table 5.3 shows the online portfolio selection strategies that were used to test the efficacy of the adaptive configuration method. Table 5.3 also shows the abbreviations we use for the approaches. We add a suffix '-ABO' for the version of the methods that are adaptively tuned by our method. Additionally, Table 5.4 also shows the classes of trading ideas that the strategies belong to and includes the number of available tunable parameters configured during the experiments.

Datasets

For our empirical analyses, we use publicly available and historical daily prices of stock and index data from diverse markets in different time periods. The data used in the experiments is shown in Table 5.5. These datasets were collected by authors of the various online portfolio selection methods, including some of the authors of the methods that we test in this paper, and are commonly used for experiments in the OLPS literature. For more details on the datasets, their

Table 5.4: Classifications and the number of tunable parameters for the trading strategies.

Classification	Algorithm	No. of Params
Benchmarks	Market	0
	BS	0
	BCRP	0
Momentum	EG	1
	ONS	2
Mean reversion	CWMR	1
	OLMAR	2
	PAMR	1

sources, selection criteria and histories of their use, see Li & Hoi (2012*a*) and Li & Hoi (2014).

Table 5.5: Summary of four real market datasets.

Dataset	# Assets	# Days	Time Frame
DJIA	30	507	Jan/2001 - Jan/2013
SP500	25	1276	Jan/1998 - Jan/2003
TSE	88	1258	Jan/1994 - Dec/1998
MSCI	24	1043	Apr/2006 - Mar/2010

Implementation Details

The OLPS toolbox (Li et al. 2015) was used for the implementations of the adaptively configured strategies used for the experiments. Bayesian optimization with the upper confidence bounds acquisition function (Srinivas et al. 2012) was incorporated into the configuration steps within the toolbox. Acquisition function optimization was performed using Particle Swarm Optimization (PSO) (Poli et al. 2007). The Gaussian process priors used were those specified in Section 5.4 and trained using *maximum a posteriori probability* (MAP) estimation

with log-normal hyper-priors on all parameter space kernel hyper-parameters, and Gamma hyper-priors on the temporal space kernel hyper-parameters.

The parameters for the first ten trading periods were randomly generated using space-filling latin hypercube sampling (Stein 1987) with bounds defined by the parameter range constraints. To deal with the computational complexity of the GP as the number of datapoints increases, for all experiments, we used the sparsification method described in Section 3.6 with a maximum number of datapoints of 300, and a reset block size of 200 datapoints.

All the parameters for the standard versions of the online portfolio selection methods were set according to their original empirical studies whose references are stated in Table 5.3.

Absolute Returns

Table 5.6 illustrates the main results for cumulative wealth on the four datasets achieved by the standard and adaptively-tuned online portfolio selection approaches. The results show that the adaptively-tuned methods achieved better performance than their standard counterparts on most of the tests. Figure 5.1 also shows the annualized percentage yields of the approaches and shows the same trend.

For the few adaptively-tuned OLPS methods that performed below their standard counterparts, we observed that after training their models, the temporal length-scales were longer than their corresponding trading time horizons. This meant that the was no time-varying behaviour learnt by the GP model of the parameter configuration map. Consequently, the exploration steps induced by adaptive Bayesian optimization lead to suboptimal returns in some trading periods which in turn caused the slightly lower overall performance.

Additionally, all the tested methods did not perform well on the DJIA dataset relative to performances on the other datasets. This was because the characteristics inherent to the DJIA dataset did not closely match the assumptions made by the tested strategies. The relatively poorer performance on the

Table 5.6: Cumulative wealth for different OLPS methods.

Methods	DJIA	SP500	TSE	MSCI
Market	0.76	1.34	1.61	0.91
BS	1.19	3.78	6.28	1.50
BCRP	1.24	4.04	6.78	1.51
EG	0.81	1.63	1.59	0.93
EG-ABO	0.81	1.63	1.61	0.92
ONS	1.53	3.34	1.61	0.86
ONS-ABO	1.53	3.25	1.62	0.99
PAMR	0.68	5.09	264.86	15.23
PAMR-ABO	1.18	6.73	274.24	15.37
CWMR	0.68	5.90	332.62	17.28
CWMR-ABO	1.34	9.12	357.24	17.44
OLMAR	1.20	8.63	678.44	22.51
OLMAR-ABO	1.63	9.59	714.36	28.49

DJIA dataset is also reflected in the smaller improvements due to our method on tests involving that dataset.

The results also show that the degree to which adaptive tuning affects the total wealth produced by a strategy on a particular dataset is proportional to how much wealth is generated by the standard version of that strategy relative to the other methods. This behaviour is particularly prevalent when we compare the relative improvements of the momentum-based algorithms (EG and ONS) to those gained by the mean reversion methods (CWMR, PAMR and OLMAR), whose improvements are much greater.

Table 5.7 lists the t-test statistics of the better-performing adaptively-tuned momentum-based algorithms. We use the method for measuring the t-statistics of achieving above-market performance as described in Grinold & Kahn (1999). This test identifies the *active returns* that are due to the skill of the adaptively-tuned methods and excludes any return that is a function of the market's move-

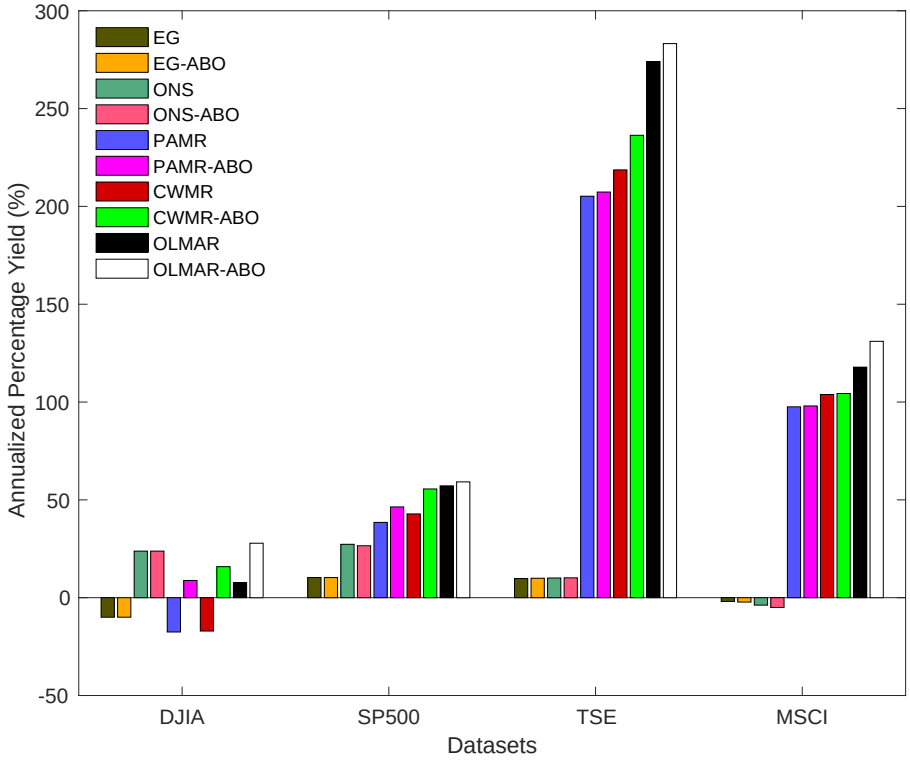

Figure 5.1: Annualized Percentage Yield.

ment. Therefore, the t-statistic provides a test of the statistical significance of the strategy's above-market performance being due to the skill of the algorithm. As rule of thumb, a t-statistic of two or more indicates that the performance of the OLPS strategy is due to skill rather than chance. This means that the probability of observing such an active return by luck is only 5%. The statistics in Table 5.7 validate the improvements via our method for those methods and show that they were very less likely due to chance, whose probabilities are at most 8.68% on the non-comporting DJIA dataset. Therefore, the improvements due to our method occurred with high confidence.

Risk and Risk-Adjusted Returns

An evaluation of the volatility and drawdown risks for the benchmarks and OLPS approaches are shown in Figure 5.2.

Table 5.7: Statistical t-test results of the performance by the better-performing methods configured by the method.

Method	Stat. Attr.	DJIA	SP500	TSE	MSCI
PAMR-ABO	t-statistics	1.3628	2.3966	2.3966	6.1694
	p-value	0.0868	0.0083	0.0083	0.0000
CWMR-ABO	t-statistics	1.6768	2.7228	3.9580	6.4074
	p-value	0.0471	0.0033	0.0000	0.0000
OLMAR-ABO	t-statistics	1.8263	2.4599	3.9305	6.3824
	p-value	0.0342	0.0070	0.0000	0.0000

The associated risk-adjusted returns in terms of annualized Sharpe and Calmer ratios are also shown in Figure 5.3. The results show that the higher the cumulative wealth, the higher the associated risk. This outcome is expected as higher rewards, which our method maximized for, are always associated with higher risk. However, other measures that take risk into account can be used for the GP model within the method to achieve desired the outcomes within prescribed risk appetites.

Comparison with Time-varying Bandits

This subsection contains additional experiments that involve an extra BO-based configuration method having the Gaussian process model proposed in the work on Time-varying bandits (TVB) in Bogunovic et al. (2016). These models involve a temporal kernel that is a Matérn $\frac{1}{2}$ covariance function and the spatial kernel is the same as the one used in the previous experiments (the rational quadratic covariance function). The goal of these experiments is to test the efficacy of the simple temporal evolution model of TVB with the more flexible approach proposed in this chapter. The experiments involving the TVB model are denoted with the '-TVB' suffix. Table 5.8 shows the results.

The results show that the method proposed in this chapter that uses the more flexible modelling approach afforded by adaptive Bayesian optimization

Table 5.8: Cumulative wealth for different OLPS methods with additional results for time-varying bandit.

Methods	DJIA	SP500	TSE	MSCI
Market	0.76	1.34	1.61	0.91
BS	1.19	3.78	6.28	1.50
BCRP	1.24	4.04	6.78	1.51
EG	0.81	1.63	1.59	0.93
EG-ABO	0.81	1.63	1.61	0.92
EGO-TVB	0.81	1.63	1.60	0.92
ONS	1.53	3.34	1.61	0.86
ONS-ABO	1.53	3.25	1.62	0.99
ONS-TVB	1.47	3.23	1.60	0.87
PAMR	0.68	5.09	264.86	15.23
PAMR-ABO	1.18	6.73	274.24	15.37
PAMR-TVB	0.75	4.07	180.42	15.86
CWMR	0.68	5.90	332.62	17.28
CWMR-ABO	1.34	9.12	357.24	17.44
CWMR-TVB	1.01	12.55	217.95	16.07
OLMAR	1.20	8.63	678.44	22.51
OLMAR-ABO	1.63	9.59	714.36	28.49
OLMAR-TVB	1.14	10.06	269.33	12.38

(ABO) performs better than TVB in all but three of the experiments. The cases where TVB performed well demonstrated that the temporal evolution of the best parameters was not smooth. This is a consequence of the ability to flexibly model the temporal evolution of the DOP in a manner that accommodates both smooth and abrupt changes in the parameters through the temporal covariance function.

Final Remarks

In practice, investment professionals will often have a predefined set of market characteristics that they want to exploit and associated criteria that they want

to maximise as they operate within a financial market. This choice of what characteristics an investor will choose to consider will depend on how informative they are about market opportunities. Therefore, parameter configuration maps can be constructed to match these preferences for both the characteristics and performance metrics.

Additionally, one of the benefits of our method is that there is a lot of prior art and experience on how existing strategies are configured that can be incorporated into the GP prior of the parameter configuration map. One aspect of the prior art that is commonplace with OLPS strategies is parameter sensitivity analysis. It is a time-consuming process that seeks to study how much a strategy's performance changes with different parameters. Our method of adaptive configuration is a natural and fine-tuned extension of the existing scope of parameter sensitivity practices. It provides the benefit of being able to determine if a strategy will benefit from adaptive parameter configuration (or not) as seen in our experiments. Our method is more general and thus subsumes the benefits of parameter sensitivity studies.

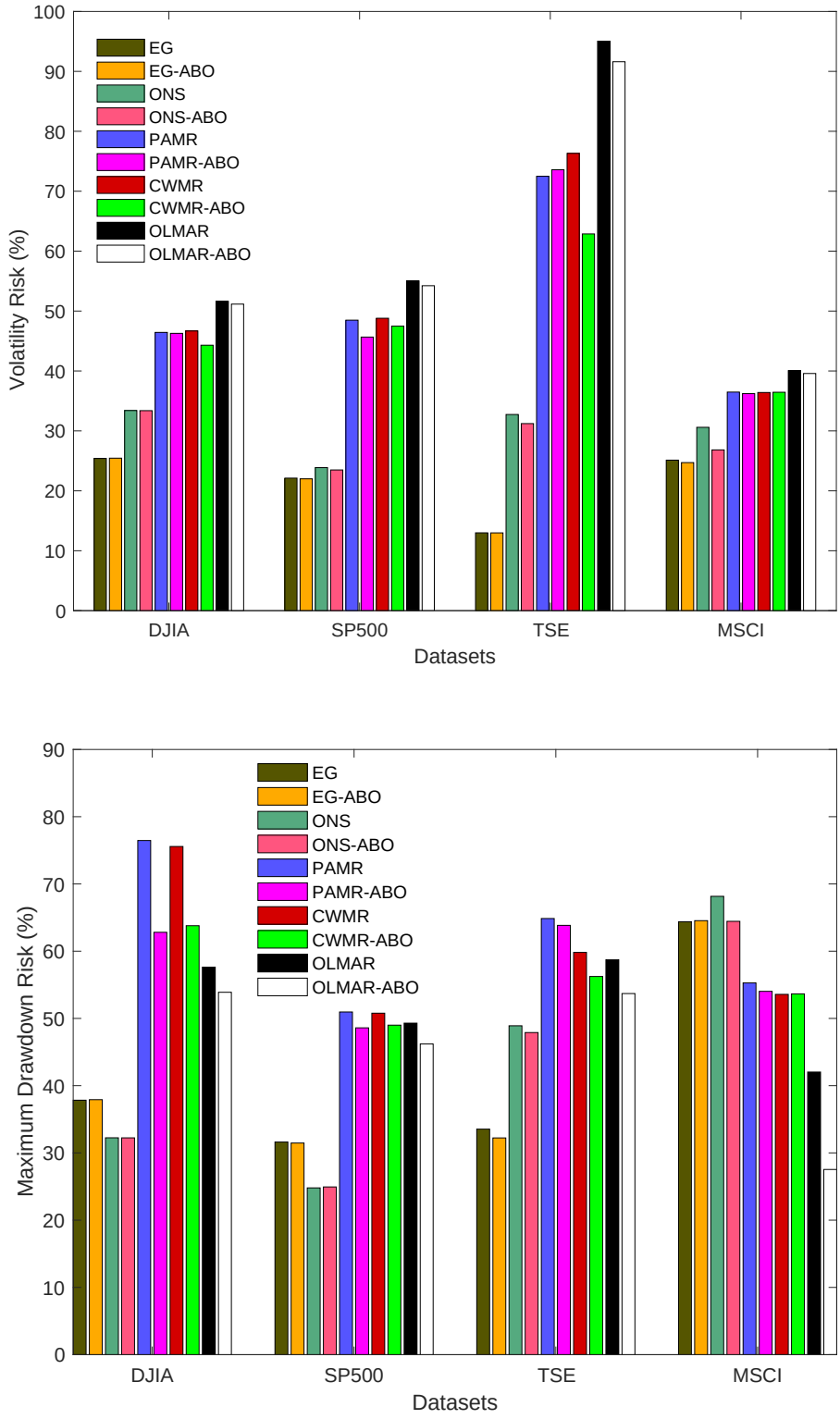

Figure 5.2: Top: Volatility Risk. Bottom: Drawdown Risk.

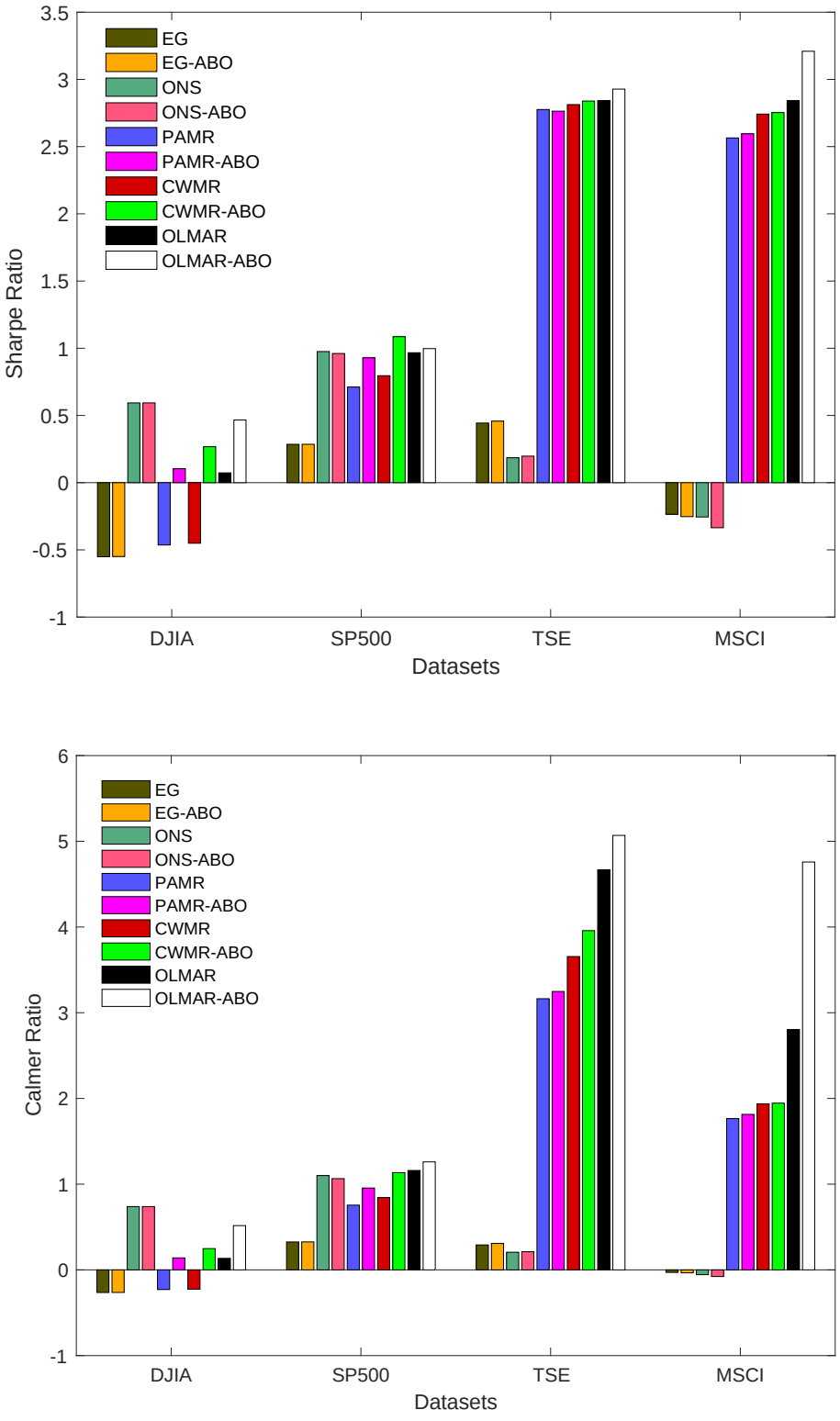

Figure 5.3: Top: Annualized Sharpe Ratio. Bottom: Calmer Ratio.

5.6 Summary

We described a method for adaptively tuning the parameters of OLPS methods. The method is supported by a parameter configuration map that is expressed in terms of a spatiotemporal GP prior. The model is intelligently searched using adaptive Bayesian optimization to generate a sequence of parameters for every trading period to maximize the return in that period. We show the efficacy of our method by comparing the performance gains of using it on known OLPS approaches operating on a diverse set of financial datasets. Our experimental analyses also showed that the method can determine if a method would benefit from adaptive parameter tuning.

6
Discussion

Some people say: how can you live without knowing?
I do not know what they mean.
I always live without knowing. That is easy.
How you get to know is what I want to know.

— Richard Feynman, 1963

In this chapter, we critically examine our findings in light of the current state of the literature on time-varying function optimization as outlined in the background survey in Chapter 2 and make judgments on what has been learnt. We also suggest avenues for future research.

Our goal in this thesis was to develop a mechanism for tracking the minimum of a temporally evolving latent, noisy and expensive function using Bayesian optimization. We adopted spatiotemporal Gaussian process priors as our surrogate model and utilized the learnt insights to dictate how to search the function intelligently. The resulting practical extensions to standard Bayesian optimization induced the tracking of the nonstationary minimum and a subset-of-data sparsification approach for cases where inference was expensive. We used the developed method to propose metaheuristics for adaptively tuning the

global learning rate in stochastic gradient descent, while also providing some further extensions to our approach with regards to using multi-task learning to deal with the existence of additional useful signals. We also use our technique to develop an adaptive configuration oracle for online portfolio selection methods.

6.1 Dynamic Optimization Problems

In Chapter 3, we modelled the temporally evolving latent, noisy and expensive function as a dynamic optimization problem. We assumed that the dynamics were native to the problem and not caused by the function being evaluated (i.e., the function has no state (Hennig et al. 2017)). This meant that the function was not modified by being evaluated by an optimization algorithm. These specifications were based on our need to solve the aforementioned minimum tracking problem that usually occurs in many meta-systems where the primary system operates in a nonstationary environment.

However, the notion of 'dynamics' can apply to optimization problems beyond the scope of what we defined. Fu et al. (2014) argue that a more general definition of dynamic optimization problems must include the requirement of multiple-decision-making, in contrast to single-decision-making for static optimization problems. They further used this more general definition to draw connections between dynamic optimization problems and reinforcement learning, where they assumed that the objective functions had a state. This is unlike the more specific problem we have studied in this thesis.

For future work, there is need to define a useful taxonomy of dynamic optimization problems classified according to

1. the nature of the objective functions,

2. the essence and cause of the dynamics,

3. what solving the problem means and its interpretation in practice.

There is also a need to prescribe infallible mechanisms for comparing the performance of methods solving these problems. This comparator will facilitate a more systematic study of dynamic optimization problems.

6.2 Optimizing for *where* and *when*

Our proposed adaptive Bayesian optimization method uses the learnt length-scales of the underlying Gaussian process to set temporal bounds of the feasible region for the search. The method exploits the learnt insights to determine where and when to sample the objective function intelligently. We showed that this mechanism allows the automatic adjustment of the budget of steps to use within a fixed time horizon, a utility that enables the allocation of an appropriate budget of steps as determined by the learnt surrogate model and a user-defined threshold.

The main problem we identified with using the standard set of search heuristics is that some of the exploration steps were suboptimal and did not adequately track the minimum. However, these suboptimal steps were valuable for the Bayesian updates critical for learning the dynamic optimization problem well.

To address this, for future work, we suggest creating acquisition functions that explore and exploit the dynamic optimization problem while maintaining an automatic tolerance of how suboptimal the explore steps can be. One feasible approach of incorporating these tolerances is the idea of safety constraints (Sui et al. 2015, Berkenkamp et al. 2016) that can be used to describe the tolerated worst-case tracking performance.

6.3 Intelligent Meta-solutions and Exploiting Prior Knowledge

In Chapters 4 and 5 we used adaptive Bayesian optimization to improve the performance of algorithms operating in nonstationary environments. Our ap-

proach provided significant performance improvements despite the fact that the meta-problems solved by adaptive Bayesian optimization were relatively less complex than the primary problem addressed by the principal algorithm (in terms of the effort and resources required to solve them). This result demonstrates the importance of intelligent meta-solutions in helping to improve systems operating on complex problems involving nonstationary data.

The importance of making efforts in perceptively solving meta-learning problems is further reflected in the recent surge in research demonstrating the ubiquity of the resulting solutions on large-scale learning systems. For example, in Ha et al. (2016), they propose the use of one neural network, which they call a hypernetwork, to generate the weights for another network which solves the primary task. Their abstraction is inspired by nature, where the hypernetworks provide an abstraction that is comparable to the relationship in the study of genetics between an encoded genotype—the hypernetwork—and an observable phenotype—the main network. They demonstrate that this approach is faster than the standard approach for training the networks. This work has inspired other more intricate formulations such as the use of an auxiliary hypernetwork to accelerate the selection of the architecture for the main network in Brock et al. (2017). In Lorraine & Duvenaud (2018), they use hypernetworks to perform a joint stochastic optimization of the weights and hyperparameters of the main network. These examples, among many others, demonstrate that there are opportunities to exploit for meta-models to improve a variety of primary systems.

Many meta-problems existing within complex systems have a lot of prior art accumulated by practitioners. In Chapters 4 and 5, we utilized the prior knowledge to define appropriate Gaussian process priors and search constraints for the acquisition function. This was vital in achieving significant performance improvements. Since many meta-problems often have a lot of domain-based prior information, adaptive Bayesian optimization is well suited to solving them.

As stated in Chapter 5, most of the parameters in meta-systems often fall into several categories such as learning rates, regularization parameters, window sizes, scaling multipliers, tolerances and heuristic parameters, among others. For future work, it would be interesting to see if it was possible to collate a taxonomy of these parameters and describe the best prior distributions for modelling meta-problems involving them.

7
Conclusions

We must know.
We will know.
— David Hilbert, 1930

This thesis proposed a method for tracking the minimum of a dynamic function. It showed that solving this problem is necessary for critical meta-problems that exist within larger systems operating in complex and nonstationary environments.

In Chapter 3, we proposed practical extensions of Bayesian optimization, which resulted in a method that we called adaptive Bayesian optimization, by modelling the dynamic objective function as a spatiotemporal Gaussian process and exploiting the learnt insights to guide the search. We also constructed a mechanism for approximating the surrogate model for cases where the number of datapoints became too expensive for Bayesian nonparametric inference. The experiments showed that the resulting practical extensions facilitated the tracking of a temporally evolving minimum and induced an appropriate budget of steps based on the learnt model and a user-defined threshold.

In Chapter 4, we demonstrated the ubiquity of adaptive Bayesian optimization by addressing the global learning rate adaptation problem in stochastic gradient descent optimization, a very important algorithm for training complex learning systems operating on large amounts of data. We used the existing domain experience to construct useful Gaussian process models that provided key insights into producing intelligent meta-heuristics for improving performance, demonstrated empirically via experiments on training neural networks.

In Chapter 5, we used our method to intelligently solve a meta-problem relating to a task for which suboptimal performance can be financially costly. We used adaptive Bayesian optimization to exploit hidden patterns in financial signals for adaptively configuring the parameters of various online trading algorithms. Experiments on equity and index financial data showed notable performance improvements on existing trading algorithms. Further analysis revealed that the improvements were due to the skill gained from new insights rather than by chance, a useful finding for practitioners.

Many new types of problems have arisen as a result of the massive amounts of data that are produced on a daily basis, and they require complex systems to solve them. Since real-world data is often nonstationary, intelligently solving the meta-problems that help these systems navigate the uncertainties associated with this nonstationarity will become ever more crucial.

A

Cholesky Updates and Downdates

In this appendix, we provide a brief description of Cholesky updates and downdates as used for efficient updates to the covariance matrix in sequential Gaussian process regression. Our description in this appendix is based on that given in Osborne (2010) and the equivalent formulations in the classic textbook (Press et al. 1992, §2.7).

A.1 Cholesky Decomposition

Matrix decomposition is the general process of factorizing a matrix into a product of matrices. Cholesky decomposition is the factorization of a matrix A into

$$A = R^\top R, \tag{A.1}$$

where R is an upper triangular matrix, and $A \in \mathbb{R}^{D\times D}$ is a symmetric and positive definite matrix satisfying the condition

$$\boldsymbol{v}^\top A \boldsymbol{v} > 0, \qquad \forall \boldsymbol{v} \in \mathbb{R}^D. \tag{A.2}$$

A.2 Cholesky Updates

One problem that arises in practice is the need to update a Cholesky decomposition. This occurs when we already have a decomposition $A = R^\top R$, and the rows and columns of A have been updated. One example of this happens in the case of sequential Gaussian process regression when the covariance matrix is updated after a new datapoint is added to the training set.

Assuming we have a symmetric and positive definite matrix A denoted in its block form as

$$\begin{bmatrix} A_{11} & A_{13} \\ A_{13}^\top & A_{33} \end{bmatrix},$$

then its known Cholesky factor is given by

$$\begin{bmatrix} R_{11} & R_{13} \\ 0 & R_{33} \end{bmatrix}.$$

If a new row and column is inserted into A to yield the following block matrix

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{12}^\top & A_{22} & A_{23} \\ A_{13}^\top & A_{23}^\top & A_{33} \end{bmatrix},$$

then the Cholesky factor of this new matrix is given by

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} \\ 0 & S_{22} & S_{23} \\ 0 & 0 & S_{33} \end{bmatrix},$$

where the blocks are calculated using the following equations:

$$S_{11} = R_{11} \tag{A.3}$$

$$S_{12} = R_{11}^\top \setminus A_{12} \tag{A.4}$$

$$S_{13} = R_{13} \tag{A.5}$$

$$S_{22} = \mathrm{chol}(A_{22} - S_{12}^\top S_{12}) \tag{A.6}$$

$$S_{23} = S_{22}^\top \setminus (A_{23} - S_{12}^\top S_{13}) \tag{A.7}$$

$$S_{33} = \mathrm{chol}(R_{33}^\top R_{33} - S_{23}^\top S_{23}). \tag{A.8}$$

For notations, we define $\boldsymbol{x} = A \setminus \boldsymbol{b}$ as a solution to $A\boldsymbol{x} = \boldsymbol{b}$, and $\mathrm{chol}(A)$ is the Cholesky decomposition of a matrix A.

A.3 Cholesky Downdates

For the reverse, we consider a case where a row and column of the matrix A has been removed. We start, like before, with a block matrix given by

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{12}^\top & A_{22} & A_{23} \\ A_{13}^\top & A_{23}^\top & A_{33} \end{bmatrix},$$

with a pre-calculated Cholesky factor given by

$$\begin{bmatrix} S_{11} & S_{12} & S_{13} \\ 0 & S_{22} & S_{23} \\ 0 & 0 & S_{33} \end{bmatrix},$$

We want the Cholesky factor of the updated version of the block matrix given by

$$\begin{bmatrix} A_{11} & A_{13} \\ A_{13}^\top & A_{33} \end{bmatrix}.$$

Its downdated upper Cholesky factor is given by

$$\begin{bmatrix} R_{11} & R_{13} \\ 0 & R_{33} \end{bmatrix},$$

where the blocks are calculated by the following formulas:

$$R_{11} = S_{11} \tag{A.9}$$

$$R_{13} = S_{13} \tag{A.10}$$

$$R_{33} = \text{chol}(S_{33}^\top S_{33} + S_{23}^\top S_{23}). \tag{A.11}$$

Bibliography

Abramowitz, M. & Stegun, I. A. (1965), 'With Formulas, Graphs, and Mathematical Tables', *National Bureau of Standards Applied Mathematics Series e* **55**, 953.

Agarwal, A., Hazan, E., Kale, S. & Schapire, R. E. (2006), Algorithms for Portfolio Management based on the Newton Method, *in* 'Proceedings of the 23rd International Conference on Machine Learning', ICML '06, ACM, New York, NY, USA, pp. 9–16.
URL: *http://doi.acm.org/10.1145/1143844.1143846*

Almeida, L. B., Langlois, T., Amaral, J. D. & Plakhov, A. (1999), Parameter Adaptation in Stochastic Optimization, *in* 'On-line Learning in Neural Networks', Cambridge University Press, pp. 111–134.

Amari, S.-I., Park, H. & Fukumizu, K. (2000), 'Adaptive Method of Realizing Natural Gradient Learning for Multilayer Perceptrons', *Neural Computation* **12**(6), 1399–1409.

Archetti, F. & Schoen, F. (1984), 'A Survey on the Global Optimization Problem: General Theory and Computational Approaches', *Annals of Operations Research* **1**(2), 87–110.
URL: *http://dx.doi.org/10.1007/BF01876141*

Bardenet, R., Brendel, M., Kégl, B. & Sebag, M. (2013), Collaborative hyperparameter tuning, *in* 'International Conference on Machine Learning', pp. 199–207.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M. & Wood, F. (2017), 'Online Learning Rate Adaptation with Hypergradient Descent', *arXiv preprint arXiv:1703.04782* .

Bengio, Y. (2012), Practical Recommendations for Gradient-based Training of Deep Architectures, *in* 'Neural Networks: Tricks of the Trade', Springer, pp. 437–478.

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization', *Journal of Machine Learning Research* **13**(Feb), 281–305.

Berkenkamp, F., Krause, A. & Schoellig, A. P. (2016), 'Bayesian Optimization with Safety Constraints: Safe and Automatic Parameter Tuning in Robotics', *ArXiv e-prints* .

Blei, D. M., Kucukelbir, A. & McAuliffe, J. D. (2017), 'Variational inference: A review for statisticians', *Journal of the American Statistical Association* **112**(518), 859–877.

Bodik, P., Hong, W., Guestrin, C., Madden, S., Paskin, M. & Thibaux, R. (2004), 'Intel Berkeley Research Lab Data'.
URL: *http://db.csail.mit.edu/labdata/labdata.html*

Bogunovic, I., Scarlett, J. & Cevher, V. (2016), 'Time-Varying Gaussian Process Bandit Optimization', *arXiv preprint arXiv:1601.06650* .

Bornn, L., Shaddick, G. & Zidek, J. V. (2012), 'Modeling Nonstationary Processes Through Dimension Expansion', *Journal of the American Statistical Association* **107**(497), 281–289.
URL: *https://doi.org/10.1080/01621459.2011.646919*

Borodin, A., El-Yaniv, R. & Gogan, V. (2003), Can we learn to beat the Best Stock?, *in* 'Advances in Neural Information Processing Systems (NIPS)'.

Bottou, L. (1998), 'Online Learning and Stochastic Approximations', *On-line Learning in Neural Networks* **17**(9), 142.

Bottou, L. (2012), Stochastic Gradient Descent Tricks, *in* 'Neural Networks: Tricks of the Trade', Springer, pp. 421–436.

Bousquet, O. & Bottou, L. (2008), The Tradeoffs of Large Scale Learning, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 161–168.

Branke, J. (1999), Memory Enhanced Evolutionary Algorithms for Changing Optimization Problems, *in* 'Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on', Vol. 3, IEEE, pp. 1875–1882.

Brochu, E., Cora, V. M. & de Freitas, N. (2010), 'A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning', *CoRR* .

Brock, A., Lim, T., Ritchie, J. M. & Weston, N. (2017), 'SMASH: One-shot Model Architecture Search through Hypernetworks', *arXiv preprint arXiv:1708.05344* .

Bui, T. D., Yan, J. & Turner, R. E. (2017), 'A unifying framework for sparse Gaussian process approximation using power expectation propagation', *Journal of Machine Learning Research* .

Bull, A. D. (2011), 'Convergence Rates of Efficient Global Optimization Algorithms', *Journal of Machine Learning Research* **12**(Oct), 2879–2904.

Calandra, R., Seyfarth, A., Peters, J. & Deisenroth, M. P. (2016), 'Bayesian Optimization for Learning Gaits under Uncertainty', *Annals of Mathematics and Artificial Intelligence* **76**(1-2), 5–23.

Carpentier, A. & Munos, R. (2012), Bandit theory meets compressed sensing for high dimensional stochastic linear bandit, *in* 'Artificial Intelligence and Statistics', pp. 190–198.

Carpin, M., Rosati, S., Khan, M. E. & Rimoldi, B. (2015), 'UAVs using Bayesian optimization to Locate Wifi Devices', *CoRR* **abs/1510.03592**. **URL:** *http://arxiv.org/abs/1510.03592*

Chalupka, K., Williams, C. K. & Murray, I. (2013), 'A Framework for Evaluating Approximation Methods for Gaussian Process Regression', *Journal of Machine Learning Research* **14**(Feb), 333–350.

Chen, B., Castro, R. & Krause, A. (2012), 'Joint optimization and variable selection of high-dimensional gaussian processes', *arXiv preprint arXiv:1206.6396* .

Chen, Y., Davis, T. A., Hager, W. W. & Rajamanickam, S. (2008), 'Algorithm 887: CHOLMOD, supernodal sparse Cholesky Factorization and Update/Uowndate', *ACM Transactions on Mathematical Software (TOMS)* **35**(3), 22.

Cisco Visual Networking Index (2017), 'The Zettabyte era – Trends and Analysis', *Cisco White Papers* .

Cover, T. M. (1991), 'Universal Portfolios', *Mathematical Finance* **1**(1), 1–29.

Cruz, C., González, J. R. & Pelta, D. A. (2011), 'Optimization in Dynamic Environments: A Survey on Problems, Methods and Measures', *Soft Computing* **15**(7), 1427–1448.

Darken, C., Chang, J. & Moody, J. (1992), Learning Rate Schedules for Faster Stochastic Gradient Search, *in* 'IEEE Workshop on Neural Networks for Signal Processing', IEEE, pp. 3–12.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. & Bengio, Y. (2014), Identifying and Attacking the Saddle Point Problem in High-dimensional Non-Convex Optimization, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 2933–2941.

Duchi, J., Hazan, E. & Singer, Y. (2011), 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization', *Journal of Machine Learning Research (JMLR)* **12**(Jul), 2121–2159.

Duvenaud, D. K., Nickisch, H. & Rasmussen, C. E. (2011), Additive Gaussian Processes, *in* 'Advances in Neural Information Processing Systems', pp. 226–234.

Feurer, M., Springenberg, J. T. & Hutter, F. (2015), Initializing Bayesian Hyperparameter Optimization via Meta-Learning., *in* 'AAAI', pp. 1128–1135.

Fletcher, R. (1987), 'Practical Methods of Optimization', *New York* **80**.

Fu, H., Lewis, P. R., Sendhoff, B., Tang, K. & Yao, X. (2014), What are Dynamic Optimization Problems?, *in* 'IEEE Congress on Evolutionary Computation (CEC)', IEEE, pp. 1550–1557.

Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q. & Cunningham, J. P. (2014), Bayesian Optimization with Inequality Constraints, *in* 'ICML', pp. 937–945.

Garnett, R., Osborne, M. A. & Roberts, S. J. (2010), Bayesian Optimization for Sensor Set Selection, *in* 'IPSN 2010 Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks', pp. 209–219.

Gelbart, M. A., Snoek, J. & Adams, R. P. (2014), 'Bayesian optimization with unknown constraints', *arXiv preprint arXiv:1403.5607* .

George, A. P. & Powell, W. B. (2006), 'Adaptive Step-sizes for Recursive Estimation with Applications in Approximate Dynamic Programming', *Machine Learning* **65**(1), 167–198.

Ghahramani, Z. & Rasmussen, C. E. (2003), Bayesian Monte Carlo, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 505–512.

Ghoshal, S. & Roberts, S. (2016), 'Extracting predictive information from heterogeneous data streams using Gaussian Processes', *Algorithmic Finance* **5**(1-2), 21–30.

Gneiting, T., Genton, M. G. & Guttorp, P. (2006), 'Geostatistical Space-time Models, Stationarity, Separability, and Full Symmetry', *Monographs On Statistics and Applied Probability* **107**, 151.

Golub, G. H. & Van Loan, C. F. (2012), *Matrix Computations*, Vol. 3, JHU Press.

Gonzalez, T. F. (1985), 'Clustering to Minimize the Maximum Intercluster Distance', *Theoretical Computer Science* **38**, 293–306.

Goovaerts, P. (1997), 'Geostatistics for natural resources evaluation', *Geostatistics for natural resources evaluation. Oxford Univ. Press, New York.* .

GPy (2012), 'GPy: A Gaussian Process Framework in Python', http://github.com/SheffieldML/GPy.

Gramacy, R. B., Gray, G. A., Le Digabel, S., Lee, H. K., Ranjan, P., Wells, G. & Wild, S. M. (2016), 'Modeling an augmented lagrangian for blackbox constrained optimization', *Technometrics* **58**(1), 1–11.

Grinold, R. C. & Kahn, R. N. (1999), *Active Portfolio Management: A Quantative Approach for Producing Superior Returns and Selecting Superior Money Managers*, 2nd edn, McGraw Hill, New York, pp. 487–490.

Guntsch, M., Middendorf, M. & Schmeck, H. (2001), An Ant Colony Optimization Approach to Dynamic TSP, *in* 'Proceedings of the 3rd

Annual Conference on Genetic and Evolutionary Computation', Morgan Kaufmann Publishers Inc., pp. 860–867.

Gyorfi, L., Lugosi, G. & Udina, F. (2006), 'Nonparametric Kernel-based Sequential Investment Strategies', *Mathematical Finance* **16**(2), 337–357.

Gyorfi, L., Udina, F. & Walk, H. (2008), 'Nonparametric Nearest Neighbor based Empirical Portfolio Selection Strategies', *Statistics and Decisions* **26**(2), 145–157.

Ha, D., Dai, A. & Le, Q. V. (2016), 'Hypernetworks', *arXiv preprint arXiv:1609.09106* .

Hansen, N. (2006), 'The CMA Evolution Strategy: A Comparing Review', *Towards a New Evolutionary Computation* pp. 75–102.

Hassabis, D. (2017), 'Explorations at the Edge of Knowledge'.
URL: *https://www.youtube.com/watch?v=ZyUFy29z3Cw*

Helmbold, D. P., Schapire, R. E., Singer, Y. & Warmuth, M. K. (1998), 'On-Line Portfolio Selection Using Multiplicative Updates', *Mathematical Finance* **8**(4), 325–347.

Hennig, P., Osborne, M. A. & Mark, G. (2017), *Probabilistic Numerics: Uncertainty in Computation*, Cambridge University Press.

Hennig, P. & Schuler, C. J. (2012), 'Entropy search for information-efficient global optimization', *Journal of Machine Learning Research (JMLR)* **13**(Jun), 1809–1837.

Herbrich, R., Lawrence, N. D. & Seeger, M. (2003), Fast Sparse Gaussian Process Methods: The Informative Vector Machine, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 625–632.

Hernández-Lobato, J. M., Hoffman, M. W. & Ghahramani, Z. (2014), Predictive Entropy Search for Efficient Global Optimization of Black-box Functions, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 918–926.

Horst, R., Pardalos, P. & Thoai, N. (2000), 'Introduction to Global Optimization', *Nonconvex Optimization and its Applications* **48**.

Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2011), Sequential model-based optimization for general algorithm configuration, *in* 'International Conference on Learning and Intelligent Optimization', Springer, pp. 507–523.

Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2013), Identifying key algorithm parameters and instance features using forward selection, *in* 'International Conference on Learning and Intelligent Optimization', Springer, pp. 364–381.

Jones, D. R., Perttunen, C. D. & Stuckman, B. E. (1993), 'Lipschitzian Optimization without the Lipschitz Constant', *Journal of Optimization Theory and Applications* **79**(1), 157–181.

Karpathy, A. (2014), 'CS231n: Convolutional Neural Networks for Visual Recognition', http://cs231n.github.io/neural-networks-3/.

Keerthi, S. & Chu, W. (2006), A Matching Pursuit Approach to Sparse Gaussian Process Regression, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 643–650.

Kelly, J. J. (1956), 'A New Interpretation of Information Rate', *Bell Systems Technical Journal* **35**(917-926).

Kim, M., Ding, Y., Malcolm, P., Speeckaert, J., Siviy, C. J., Walsh, C. J. & Kuindersma, S. (2017), 'Human-in-the-loop Bayesian optimization of

Wearable Device Parameters', *PLOS ONE* **12**(9), 1–15.
URL: *https://doi.org/10.1371/journal.pone.0184054*

Kimeldorf, G. & Wahba, G. (1971), 'Some Results on Tchebycheffian Spline Functions', *Journal of Mathematical Analysis and Applications* **33**(1), 82–95.

Kingma, D. P. & Ba, J. (2015), Adam: A method for Stochastic Optimization, *in* 'International Conference on Learning Representations (ICLR)'.

Krause, A. & Ong, C. S. (2011), Contextual Gaussian Process Bandit Optimization, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 2447–2455.

Krizhevsky, A., Nair, V. & Hinton, G. (2014), 'The CIFAR-10 Dataset', *online: http://www. cs. toronto. edu/kriz/cifar. html* .

Lancaster, J., Lorenz, R., Leech, R. & Cole, J. H. (2018), 'Bayesian Optimization for Neuroimaging Pre-processing in Brain Age Classification and Prediction', *Frontiers in Aging Neuroscience* **10**, 28.
URL: *https://www.frontiersin.org/article/10.3389/fnagi.2018.00028*

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based Learning Applied to Document Recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

LeCun, Y., Cortes, C. & Burges, C. (2010), ' MNIST Handwritten Digit Database ', *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist* **2**.

Li, B. & Hoi, S. (2012*a*), 'A Database for On-line Portfolio Selection', http://www.mysmu.edu.sg/faculty/chhoi/olps/datasets.html. Accessed: 2015-02-15.

Li, B., Hoi, S. C. & Gopalkrishnan, V. (2011), 'CORN: Correlation-driven Nonparametric Learning Approach for Portfolio Selection', *ACM Transactions on Intelligent Systems and Technology* **2**(3), 21:1–21:29.

Li, B. & Hoi, S. C. H. (2012*b*), On-Line Portfolio Selection with Moving Average Reversion, *in* 'Proceedings of the International Conference on Machine Learning'.

Li, B. & Hoi, S. C. H. (2014), 'Online Portfolio Selection: A Survey', *ACM Comput. Surv.* **46**(3), 35:1–35:36.
URL: *http://doi.acm.org/10.1145/2512962*

Li, B., Hoi, S. C., Zhao, P. & Gopalkrishnan, V. (2013), Confidence Weighted Mean Reversion Strategy for On-Line Portfolio Selection, *in* 'ACM Transactions on Knowledge Discovery from Data'.

Li, B., Sahoo, D. & Hoi, S. (2015), 'OLPS: A toolbox for online portfolio selection', *Journal of Machine Learning Research (JMLR)* .

Li, B., Zhao, P., Hoi, S. & Gopalkrishnan, V. (2012), 'PAMR: Passive aggressive mean reversion strategy for portfolio selection', *Machine Learning* **87**(2), 221–258.

Lizotte, D., Wang, T., Bowling, M. & Schuurmans, D. (2007), Automatic Gait Optimization with Gaussian Process Regression, *in* 'International Joint Conference on Artificial Intelligence (IJCAI)'.

Lorraine, J. & Duvenaud, D. (2018), 'Stochastic Hyperparameter Optimization through Hypernetworks', *arXiv preprint arXiv:1802.09419* .

Luenberger, D. G., Ye, Y. et al. (1984), *Linear and Nonlinear Programming*, Vol. 2, Springer.

Mack, Y., Goel, T., Shyy, W. & Haftka, R. (2007), Surrogate Model-based Optimization Framework: A Case Study in Aerospace Design, *in*

'Evolutionary computation in dynamic and uncertain environments', Springer, pp. 323–342.

Mahendran, N., Wang, Z., Hamze, F. & De Freitas, N. (2012), Adaptive MCMC with Bayesian Optimization., *in* 'International Conference on Artificial Intelligence and Statistics (AISTATS)', Vol. 22, pp. 751–760.

Michalewicz, Z., Schmidt, M., Michalewicz, M. & Chiriac, C. (2007), Adaptive Business Intelligence: Three Case Studies, *in* 'Evolutionary Computation in Dynamic and Uncertain Environments', Springer, pp. 179–196.

Minka, T. P. (2001), Expectation propagation for approximate Bayesian inference, *in* 'Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence', Morgan Kaufmann Publishers Inc., pp. 362–369.

Mockus, J. (1975), 'On Bayesian Methods for Seeking the Extremum', *Lecture Notes in Computer Science* **27**, 400–404.

Moser, I. & Chiong, R. (2013), Dynamic Function Optimization: The Moving Peaks Benchmark, *in* 'Metaheuristics for Dynamic Optimization', Springer, pp. 35–59.

Moulines, E. & Bach, F. R. (2011), Non-Asymptotic Analysis of Stochastic Approximation Algorithms for Machine Learning, *in* 'Advances in Neural Information Processing Systems (NIPS)', pp. 451–459.

Nesterov, Y. E. (1983), A method for solving the convex programming problem with convergence rate o (1/kˆ 2), *in* 'Dokl. Akad. Nauk SSSR', Vol. 269, pp. 543–547.

Osborne, M. (2010), Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature, PhD thesis, PhD thesis, University of Oxford.

Osborne, M. A., Garnett, R. & Roberts, S. J. (2009), 'Gaussian Processes for Global Optimization', *International Conference on Learning and Intelligent Optimization* .

Paszke, A., Gross, S., Chintala, S. & Chanan, G. (2017), 'PyTorch: Tensors and Dynamic Neural Networks in Python with Strong GPU Acceleration'.

Pelta, D., Cruz, C. & Verdegay, J. L. (2009), 'Simple Control Rules in a Cooperative System for Dynamic Optimisation Problems', *International Journal of General Systems* **38**(7), 701–717.

Poli, R., Kennedy, J. & Blackwell, T. (2007), 'Particle Swarm Optimization', *Swarm intelligence* **1**(1), 33–57.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992), 'Numerical Recipes in C++', *The Art of Scientific Computing* .

Qian, N. (1999), 'On the momentum term in gradient descent learning algorithms', *Neural networks* **12**(1), 145–151.

Quiñonero-Candela, J. & Rasmussen, C. E. (2005), 'A Unifying View of Sparse Approximate Gaussian Process Regression', *Journal of Machine Learning Research* **6**(Dec), 1939–1959.

Rainforth, T., Le, T. A., van de Meent, J.-W., Osborne, M. A. & Wood, F. (2016), Bayesian Optimization for Probabilistic Programs, *in* 'Advances in Neural Information Processing Systems (NIPS) (NIPS)', pp. 280–288.

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

Robbins, H. & Monro, S. (1951), 'A Stochastic Approximation Method', *The Annals of Mathematical Statistics* pp. 400–407.

Roux, N. L. & Fitzgibbon, A. W. (2010), A Fast Natural Newton Method, *in* 'Proceedings of the 27th International Conference on Machine Learning (ICML)', Citeseer, pp. 623–630.

Ruder, S. (2016), 'An Overview of Gradient Descent Optimization Algorithms', *arXiv preprint arXiv:1609.04747* .

Samo, Y.-L. K. & Roberts, S. J. (2016), 'String and Membrane Gaussian Processes', *The Journal of Machine Learning Research* **17**(1), 4485–4571.

Särkkä, S. & Hartikainen, J. (2012), Infinite-Dimensional Kalman Filtering Approach to Spatio-Temporal Gaussian Process Regression, *in* 'International Conference on Artificial Intelligence and Statistics (AISTATS)'.

Schaul, T., Zhang, S. & LeCun, Y. (2013), No More Pesky Learning Rates, *in* 'International Conference on Machine Learning (ICML)', pp. 343–351.

Schraudolph, N. N. (1999), 'Local Gain Adaptation in Stochastic Gradient Descent'.

Schraudolph, N. N. (2002), 'Fast Curvature Matrix-vector Products for Second-order Gradient Descent', *Neural Computation* **14**(7), 1723–1738.

Seeger, M., Teh, Y.-W. & Jordan, M. (2005), Semiparametric latent factor models, Technical report.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. & de Freitas, N. (2015), Taking the Human out of the Loop: A Review of Bayesian Optimization, Technical report, Universities of Harvard, Oxford, Toronto, and Google DeepMind.
URL:
http://www.cs.ox.ac.uk/people/nando.defreitas/publications/BayesOptLoop.pdf

Slivkins, A. & Upfal, E. (2008), Adapting to a Changing Environment: the Brownian Restless Bandits, *in* 'Conference on Learning Theory', pp. 343–354.

Snoek, J., Larochelle, H. & Adams, R. P. (2012), Practical Bayesian Optimization of Machine Learning Algorithms, *in* 'Advances in Neural Information Processing Systems (NIPS)'.

Snoek, J. R. (2013), Bayesian optimization and semiparametric models with applications to assistive technology, PhD thesis, University of Toronto.

Srinivas, N., Krause, A., Kakade, S. & Seeger, M. (2012), 'Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting', *Information Theory, IEEE Transactions on* **58**(5), 3250–3265.

Stein, M. (1987), 'Large Sample Properties of Simulations using Latin Hypercube Sampling', *Technometrics* **29**(2), 143–151.

Sui, Y., Gotovos, A., Burdick, J. & Krause, A. (2015), Safe Exploration for Optimization with Gaussian Processes, *in* 'International Conference on Machine Learning', pp. 997–1005.

Swersky, K., Snoek, J. & Adams, R. P. (2013), Multi-task Bayesian optimization, *in* 'Advances in neural information processing systems', pp. 2004–2012.

Swersky, K., Snoek, J. & Adams, R. P. (2014), 'Freeze-thaw Bayesian Optimization', *arXiv preprint arXiv:1406.3896* .

Tezuka, M., Munetomo, M. & Akama, K. (2007), Genetic Algorithm to Optimize Fitness Function with Sampling Error and its Application to Financial Optimization Problems, *in* 'Evolutionary Computation in Dynamic and Uncertain Environments', Springer, pp. 417–434.

Tieleman, T. & Hinton, G. (2012), 'Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude', *COURSERA: Neural networks for machine learning* **4**(2), 26–31.

Titsias, M. (2009), Variational learning of inducing variables in sparse Gaussian processes, *in* 'Artificial Intelligence and Statistics', pp. 567–574.

Torn, A. & Zilinskas, A. (1989), *Global Optimization*, Springer-Verlag New York, Inc.

Trojanowski, K. & Wierzchoń, S. T. (2009), 'Immune-based Algorithms for Dynamic Optimization', *Information Sciences* **179**(10), 1495–1515.

Van Vaerenbergh, S., Santamaría, I. & Lázaro-Gredilla, M. (2012), Estimation of the Forgetting Factor in Kernel Recursive Least Squares, *in* 'IEEE International Workshop on Machine Learning for Signal Processing (MLSP)', IEEE, pp. 1–6.

Wang, Z., Zoghi, M., Hutter, F., Matheson, D. & de Freitas, N. (2013), Bayesian Optimization in High Dimensions via Random Embeddings, *in* 'International Joint Conference on Artificial Intelligence'.

Willetts, D. (2013), *Eight Great Technologies*, Policy Exchange & the UK Department for Business, Innovation & Skills.
URL: *https://www.gov.uk/government/speeches/eight-great-technologies*

Xu, W. (2011), 'Towards Optimal One-pass Large-scale Learning with Averaged Stochastic Gradient Descent', *arXiv preprint arXiv:1107.2490* .

Yamashita, T., Sato, N., Kino, H., Miyake, T., Tsuda, K. & Oguchi, T. (2018), 'Crystal Structure Prediction Accelerated by Bayesian Optimization', *Phys. Rev. Materials* **2**, 013803.
URL: *https://link.aps.org/doi/10.1103/PhysRevMaterials.2.013803*

Yogatama, D. & Mann, G. (2014), Efficient transfer learning method for automatic hyperparameter tuning, *in* 'Artificial Intelligence and Statistics', pp. 1077–1085.