Practical Bayesian optimisation for hyperparameter tuning

Ahsan S. Alvi

Christ Church

University of Oxford

Submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

Trinity Term 2019

Abstract

Advances in machine learning have had, and continue to have, a profound effect on scientific research and industrial activities. We are able to uncover insights contained within large troves of data and develop models to solve problems that seemed infeasible until recently. Before we can train a model, we have to define high-level properties, also known as the model's hyperparameters. Examples include the architecture of a neural network, the class and parameters of a stochastic optimisation algorithm, the number of trees in a random forest or the kernel in a Gaussian process. One of the key challenges in developing good machine learning models is choosing good values for these hyperparameters.

Evaluating the quality of a set of hyperparameters is usually computationally costly, as it requires us to first train a model specified by the hyperparameters before we can evaluate a score. The computational costs of scoring a configuration means that we want to perform this hyperparameter search with as few trials as possible. Bayesian optimisation (BO) is a general-purpose efficient global optimisation technique that has proven to be remarkably effective for many expensive optimisation tasks, including hyperparameter tuning. This thesis brings together multiple contributions aimed at making BO more practically applicable, focussing on uncertain-input measurements as well as model hyperparameter tuning.

The case of data with input noise is relevant when we are modelling (or finding extremal values of) a quantity defined on a spatial grid. We undertake a comparative study evaluating existing uncertain-input GP methods, and develop a novel approach based on the unscented transform that outperforms existing uncertain-input methods.

We then address the question of parallelising BO to make optimal use of available computing resources. We develop a novel class of methods for asynchronous parallel BO, and demonstrate empirically the benefits of our approach, as well as more generally showing the advantages of asynchronous over synchronous BO.

Finally, we propose a new framework for dealing with a mixture of categorical and continuous hyperparameters. With few exceptions, model hyperparameters consist of both categorical and continuous variables, but BO approaches have mainly focussed on continuous search spaces. We present a novel framework that combines multi-armed bandits with continuous BO methods and leverages each of their strengths to achieve state-of-the-art performance on real-world optimisation tasks.

Statement of Originality

I hereby declare that, except where made explicitly clear, the contents of this dissertation are my own original work, and to the best of my knowledge do not contain materials submitted in whole, or in part, for consideration for any other degree or qualification at the University of Oxford or any other academic or professional institution.

Ahsan Shahzad Alvi
September 17, 2019

Acknowledgements

This thesis is a product of the guidance, support and friendship of many people that I have come to know during my time at Oxford, and to whom I would like to express my gratitude.

First, I want to thank both Steve Roberts for accepting me as his DPhil student, and Mike Osborne for agreeing to co-supervise me. Your steady guidance, unwavering patience and constant support helped me find my feet. Both of you provided me with complete academic freedom, encouraging me as I forged my own path through the jungle that is research. Nevertheless, you kept me on track when deadlines approached, for which I am immensely grateful.

I have been lucky to do this DPhil at the Machine Learning Research Group (MLRG), which is a wonderful collection of brilliant, collaborative researchers, fostering an open yet competitive environment. I had the pleasure of collaborating with Binxin Ru, having discovered our shared interest in Bayesian optimisation, the fruits of which constitute parts of this thesis. Your steadfast work ethic and unwavering enthusiasm were infectious. I was also fortunate to work with Jan Calliess and Vu Nguyen, who brought many ideas and energy to the collaboration.

One of the greatest gifts that I take away from my time at Oxford are the experiences and friendships that I have formed over the years. This list has to include Ivan Kiskin, Adam Cobb, Kyriakos Polymenakos and Arno Blaas. I discovered many new interests, developed new hobbies and learnt an inordinate amount about so many things. I wouldn't have it any other way. You introduced me to so many new ideas, and our long conversations on topics both banal and philosophical remain fond memories. Thank you also to all the members of MLRG that made it so memorable, including Zihao Zhang, Binxin Ru, Edward Wagstaff, Ibrahim Almosallam, Gabriele Abbati, and many more.

Finally, but most importantly, I owe the deepest debt of gratitude to the people closest to me: Abbu-ji, Ammi-ji, Papa-ji, Mama-ji and Mohsan. Your endless love and unwavering encouragement saw me through the highs and lows of this degree. Thank you also to my sister-in-law Amna for our adventures together and bringing even more laughter to our family. You all patiently stood by and supported me every step of the way. This thesis is as much your hard work as it is mine.

Parts of this thesis are based on joint-authored publications, which are listed below.

- Ahsan S. Alvi, Binxin Ru, Jan-Peter Calliess, Stephen Roberts, and Michael A. Osborne. "Asynchronous batch Bayesian optimisation with improved local penalisation". In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of Proceedings of Machine Learning Research, pages 253–262, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/alvi19a.html.

  This publication forms the basis of the work in Chapter 4. Ru and I contributed equally to the theory. I provided all experimental results except for the CIFAR CNN training experiment, which was provided by Ru. Calliess provided guidance on the theory relating to Lipschitz constants. Roberts and Osborne provided general advice on theory and experiments.

- Binxin Ru, Ahsan S. Alvi, Vu Nguyen, Michael A. Osborne, Stephen J Roberts. "Bayesian Optimisation over Multiple Continuous and Categorical Inputs." In *3rd Workshop on Meta-Learning at NeurIPS 2019, Vancouver, Canada*. URL https://arxiv.org/abs/1906.08878

  This publication forms the basis of the work in Chapter 5. Ru and I contributed equally to the theory and experiments. Nguyen and Roberts provided advice on the theory and experiments. Osborne provided general guidance.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Introduction

Advances in artificial intelligence and machine learning (ML) have had, and continue to have, a profound effect on scientific research and industrial activities. With the advent of ever-increasing volumes of data and cheap computational resources, ML is enabling us to uncover insights contained within large troves of data and develop models to solve difficult problems that were infeasible until recently.

The effects of this interest in data-driven modelling and decision-making has led to many exciting developments in different areas of life and scientific research, such as healthcare (Miotto et al., 2017; Ghassemi et al., 2018), genomics (Libbrecht and Noble, 2015), natural language processing (Deng and Liu, 2018), autonomous vehicles (González et al., 2015), manufacturing (Wuest et al., 2016), drug discovery (Lavecchia, 2015), and many more.

With very few exceptions, ML models are accompanied by hyperparameters that define higher-order concepts of the model, such as the model class or structure. Before a model can be trained, the values of these hyperparameters must be selected. These hyperparameters can take many different forms, such as defining the architecture of a neural network (numbers of units, types of activation functions), optimiser choices (Adam, AdaDelta, SGD), kernel hyperparameters in Gaussian processes (GPs), regularisation parameters in support vector machines, etc. Selecting these hyperparameters defines the model's set of parameters (e.g. weights and biases in a neural network) that are then often optimised via gradient-based optimisers.

We can summarise this optimisation of the hyperparameters in the following concise problem statement:

$$\mathbf{x}^* = \max_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x}), \tag{1.1}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the metric we wish to maximise (or minimise) to identify the optimal model, and the hyperparameters are summarised in the d-dimensional vector $\mathbf{x}$ in the bounded hyperparameter space $\mathcal{X}$. For example, $f(\mathbf{x})$ could be the predictive log likelihood, the mean squared error or the classification accuracy of a model, defined by hyperparameters $\mathbf{x}$, evaluated on unseen data.

What makes these hyperparameter selections difficult is the fact that a single model training procedure can take anywhere from seconds to multiple hours or days. This duration depends on various factors such as the model class, the size of the training data, the optimisation algorithm used etc., and generally the training time increases as models become more complex or the data volumes increase. Completing such a training run allows us then to compute the desired performance metric, which estimates the quality of the selected hyperparameter values.

In practice, hyperparameter tuning is often performed manually by the model developers. In such cases a practitioner relies on the intuitions developed from past experience to be able to identify a good configuration of the hyperparameters. This can be a frustrating exercise, and the space of hyperparameter configurations $\mathcal{X}$

is not guaranteed to be explored in a consistent manner, as there are often more hyperparameters than can be visualised effectively by a practitioner.

Popular naïve approaches for selecting model hyperparameters automatically, such as grid-search, are likely to take too long, as we train a lot of models to cover as many possible combinations of hyperparameters as we can (exponential in d). It also stands to reason that evaluating many models with incrementally differing hyperparameters is an inefficient approach to exploring the hyperparameter space $\mathcal{X}$, since we would reasonably expect models with similar hyperparameters to perform similarly. This motivates the use of efficient global optimisation approaches (Jones et al., 1998), that more effectively search the hyperparameter space $\mathcal{X}$, while taking as few samples of f as possible to minimise the time taken to arrive at the best model configuration. One such optimisation method is Bayesian optimisation (BO) (Shahriari et al., 2016).

BO was recently shown to be a powerful method for optimising ML model hyperparameters in Snoek et al. (2012), and many further successes have been recorded since. Arguably one of the most highly-publicised recent developments in ML is AlphaGo, the agent developed by Google DeepMind that achieved super-human ability at the board game Go (DeepMind, 2016). The developers of AlphaGo also used BO to tune its hyperparameters to improve the agent in self-play games (Chen et al., 2018).

The desire to capitalise on the wide-spread interest in ML methods has also lead to the emergence of AutoML platforms (Quanming et al., 2018). The value in these platforms lies in the fact that they enable non-experts to leverage ML methods to tackle problems without needing a ML education. For such systems to work, they need to include techniques that automate the steps of a data scientist's modelling process in order to create models with minimal human input. A key skill in such a system is the ability to find optimal settings for model hyperparameters to solve a user's task. This is an expensive, and often black-box, optimisation task, which is an application domain for which BO is particularly suitable.

Published research in BO relies on empirically demonstrating the quality of proposed algorithms, which is also reflected in the work presented in this thesis. This is due to the fact that developing formal guarantees without making significant assumptions, such as fixing the hyperparameters of the BO surrogate (see Section 2.3), has proven to be a challenging task. In practice, the hyperparameters of the BO surrogate are often optimised throughout the procedure, so guarantees for fixed hyperparameters may not be relevant.

Taking this into account, we will evaluate the performance our methods on synthetic problems widely-used in the BO literature, as well as on machine learning hyperparameter tuning tasks, which are the primary motivator for this work.

1.2 Thesis outline and summary of contributions

Drawing our motivation from the breadth of applications of BO, this thesis brings together multiple pieces of work addressing practical challenges related to BO, in particular when considering noisy domains and hyperparameter optimisation.

In Chapter 2 we introduce relevant concepts of probability theory, GPs and BO and bandit algorithms.

In Chapter 3 we investigate GP regression with input uncertainty. GPs are the dominant surrogate model used in BO and by default only model uncertainty on the observed function values. They do not consider input uncertainty in the standard formulation. Being able to deal with input uncertainty is relevant to situations where we are modelling (or finding extremal values of) a quantity over a spatial grid, and where the positional data is derived from noisy methods such as triangulation or GPS. We undertake a comparative study of existing uncertain-input GP methods and the standard GP, and also develop a novel approach based on the unscented transform. We find that a standard GP with a stationary kernel is able to model input-corrupted data as well as, if not better than, existing GP variants that explicitly incorporate input noise. Our method based on the unscented transform outperforms existing uncertain-input GP methods, as well as the standard GP.

We then proceed to discuss questions that arise when applying BO in practice. In Chapter 4 we address a practical question of applying BO, while making optimal use of parallel computing resources. Many parallel BO methods have been developed recently, primarily driven by the success of BO for optimising ML hyperparameters. These approaches predominantly consider synchronous evaluation of jobs, which has disadvantage in terms of resource utilisation in cases for which the runtimes of evaluations differ from each other. We make the case for asynchronous parallel BO, and show the benefits of asynchronous over synchronous BO. We also propose a novel method for asynchronous BO that outperforms existing synchronous and asynchronous methods in terms of both optimisation runtime and sample efficiency.

The majority of BO research considers the optimisation of a black-box function with a bounded continuous input space. This is a reasonable assumption for many applications, but not for ML systems, where the parameter space is, almost without exception, made up of both continuous and categorical variables. In Chapter 5 we present a new approach for explicitly dealing with both categorical and continuous variables by bringing together multi-armed bandits and continuous BO methods for both sequential BO as well as batch evaluation.

Finally, in Chapter 6 we conclude the thesis and present possible future research directions that relate to the work presented here.

2 Background

Contents

2.1 Bayesian probability theory

We begin our survey of background methods with the philosophy that underpins all of the work in this thesis: Bayesian statistics. The theorem underpinning Bayesian statistics is Bayes' rule, which provides a framework for performing inference about unobserved random variables and observed data. Assume A and

B are two (discrete) random variables corresponding to two different events. The joint probability of these two random variables can be decomposed using their conditional distributions in two equivalent ways:

$$P(A, B) = P(A)P(B|A) = P(B)P(A|B), \tag{2.1}$$

by using the sum and product rules of probability (Jaynes, 2003), as well as symmetry $P(A, B) = P(B, A)$. Rearranging Equation (2.1) gives us Bayes' rule:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}. \tag{2.2}$$

The denominator can be expressed solely in terms of quantities in the numerator, i.e.

$$P(A) = \sum_B P(A|B)P(B), \tag{2.3}$$

so we can view this term as a normalising constant, ensuring that the resulting distribution sums to one.

For continuous quantities, we define a probability density function, $p(x)$, where x is a continuous random variable:

$$P(a \leqslant x \leqslant b) = \int_a^b p(x)\mathrm{d}x, \tag{2.4}$$

for which we also require:

$$p(x) \geqslant 0, \tag{2.5}$$

$$\int_{-\infty}^{\infty} p(x)\mathrm{d}x = 1. \tag{2.6}$$

This allows us to use Bayes' rule to perform inference on continuous spaces, such as selecting model parameters, which is a key objective in our work.

A Bayesian approach to inferring a model's parameters proceeds as follows. Consider a model class $\mathcal{M}$ with parameters θ, as well as observed data $\mathcal{D}$. Applying Bayes' rule to this scenario, we can write

$$p(\theta|\mathcal{D}, \mathcal{M}) = \frac{p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})}{p(\mathcal{D}|\mathcal{M})}. \tag{2.7}$$

Let us discuss each term in turn. Before we have observed any data, the model class allows us to define the list of parameters and their likely ranges, e.g. from

past experience on similar types of problems or by engaging a domain expert for advice. This is summarised in the *prior* distribution $p(\theta|\mathcal{M})$, as it encodes all of our beliefs *prior to observing any data.*

The second term in the numerator, $p(\mathcal{D}|\theta, \mathcal{M})$, can have two different meanings depending on the context. In the case where we have not yet observed any data, we can sample from this distribution to generate data for different values of θ. Once we've observed data, we consider the data fixed and this term is considered the *likelihood* of the parameters. In this case, the likelihood is a function of the parameters θ and represents how well different parameter choices explain the observed data.

The denominator, $p(\mathcal{D}|\mathcal{M})$, is called the *marginal likelihood* or *model evidence* and is defined as

$$p(\mathcal{D}|\mathcal{M}) = \int p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M})\mathrm{d}\theta. \tag{2.8}$$

The model evidence can also be used as measure of how well the model class $\mathcal{M}$ is suited to the modelling question. Comparing the model evidences for different model classes allows us to perform model selection by evaluating $p(\mathcal{D}|\mathcal{M}_1), p(\mathcal{D}|\mathcal{M}_2), \ldots$ for model classes $\mathcal{M}_1, \mathcal{M}_2, \ldots$ and choosing the best one.

The left-hand side, $p(\theta|\mathcal{D}, \mathcal{M})$, is our *posterior* distribution over the parameters θ, which updates our prior (expert) knowledge with the observed data.

Moments of distributions

Two properties of probability distributions that we will make use of in this thesis are the mean and the covariance. They are also referred to as the first and second moment of a distribution respectively.

For a probability distribution $p(\mathbf{a})$ for a vector-valued random variable $\mathbf{a}$ the mean is defined as

$$\mathbb{E}[\mathbf{a}] = \int \mathbf{a}p(\mathbf{a})\mathrm{d}\mathbf{a}, \tag{2.9}$$

and the covariance is

$$\mathbb{C}[\mathbf{a}] = \mathbb{E}\Big[(\mathbf{a} - \mathbb{E}[\mathbf{a}])(\mathbf{a} - \mathbb{E}[\mathbf{a}])^T\Big]. \tag{2.10}$$

2.1.1 Parameter inference

In many cases, full Bayesian inference of a model's parameters may not be possible, as computing the posterior distribution may be too challenging. This could be caused by posterior being analytically intractable or computationally too expensive to compute accurately. Instead of performing exhaustive sampling over the domain of θ, two simplifications are often used in practice. *Maximum likelihood estimation* (MLE) optimises the likelihood, or equivalently its logarithm, to find the mode of the likelihood:

$$\hat{\theta}_{\text{MLE}} = \arg\max_{\theta} \Big(\log p(\mathcal{D}|\theta, \mathcal{M}) \Big). \tag{2.11}$$

This approximates the posterior by placing all of the probability mass on the MLE estimate $\hat{\theta}_{\text{MLE}}$. Since the logarithm is a monotonic transformation, the locations of the extrema of $p(.)$ and $\log p(.)$ are the same, and performing a log transform may improve the numerical stability of such an optimisation.

The second option is *maximum a posteriori* (MAP), where the (optionally log-transformed) joint distribution is optimised:

$$\hat{\theta}_{\text{MAP}} = \arg\max_{\theta} \Big(\log p(\mathcal{D}|\theta, \mathcal{M}) + \log p(\theta|\mathcal{M}) \Big). \tag{2.12}$$

MAP inference is widely utilised, as it allows us to incorporate the prior into the parameter selection, while also leveraging popular gradient-based methods in the optimisation.

2.1.2 Priors

A key ingredient of the Bayesian approach is the choice of prior distribution $p(\theta|\mathcal{M})$. This distribution encodes our knowledge about the domain we are reasoning about, so it should be chosen carefully. As a general rule, in the small-data domain that we are investigating in this thesis, it is always preferable to provide as much useful information to the inference as possible. On the opposite end of the spectrum, if we know little about the application domain, we should take care to define priors that do not inadvertently mislead the model. This reliance on the prior is also the

main criticism of Bayesian approaches, since different practitioners can arrive at different results given the same data as a result of differing priors.

The prior choice can also be influenced by the mathematical form of the likelihood. If we wish to arrive at an analytic posterior $p(\theta|\mathcal{D}, \mathcal{M})$, then we can adapt our prior and likelihood by choosing a *conjugate* pair, which would result in a posterior distribution of the same functional form as the prior. This also gives us the functional form of the marginal likelihood $p(\mathcal{D}|\mathcal{M})$ for "free", though the values of its parameters may still be expensive to compute. Examples of conjugate pairs are a Gaussian likelihood with a Gaussian prior, or Bernoulli likelihood with a Beta prior.

If we do not require a normalised posterior distribution, then we have more flexibility with the choice of priors as we do not need to compute the denominator in Equation (2.7). This situation is relevant to MLE and MAP inference, where we are interested only in the location of the mode of the distribution, or if we decide to use a sampling technique for unnormalised distributions, such as slice sampling (Murray et al., 2010; Murray and Adams, 2010).

Informative and noninformative priors

Regardless of whether we use a conjugate prior or not, it is important to carefully construct the prior, starting with deciding between using an informative or noninformative prior (Bishop, 2006). If we do not have a strong intuition about the quantity we wish to perform inference about, but want to constrain the variable within a range, then we can use a uniform prior over a bounded region, which assigns zero probability mass outside the bounded region and equal probability to values within. This allows us to "let the data speak for themselves" (Bishop, 2006).

If we have an intuition about the likely range for a scalar parameter θ, which could be derived from past experience or access to an expert, then we may decide to use an informative prior. Examples include using a normal distribution (if $\theta \in \mathbb{R}$) or gamma distribution (if $\theta \in \mathbb{R}^+$). The spread of the prior distribution allows us to encode the strength of our belief, as it affects how much data will be needed to move away from the prior and accept an alternative hypothesis.

Regularisation and priors

There exists an interesting link in the ML literature between priors and regularisation that may help further develop an intuition of the effect of priors. Consider a model f with parameters θ being fit to data $\{\mathbf{x}_i, y_i\}_{i=1}^N$ using a least squares loss function. We can write the error as

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i, \theta) - y_i\Big)^2. \tag{2.13}$$

In order to ensure that the model does not overfit[1], we often add an additional regularisation term to the loss (Bishop, 2006), e.g. l_2-regularisation:

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i, \theta) - y_i\Big)^2 + \frac{\lambda}{2}\|\theta\|^2. \tag{2.14}$$

Now consider the case where we are performing MAP optimisation to find a model's optimal parameters θ by using a Gaussian likelihood $\mathcal{N}(y_i; f(\mathbf{x}_i, \theta), \sigma^2)$ and a Gaussian prior $\mathcal{N}(\theta; 0, 1)$:

$$p(\theta|\mathcal{D}, \mathcal{M}) \propto p(\mathcal{D}|\theta, \mathcal{M})p(\theta|\mathcal{M}) \tag{2.15}$$

$$= \prod_{i=1}^{N}\mathcal{N}(y_i; f(\mathbf{x}_i, \theta), \sigma^2)\mathcal{N}(\theta; 0, 1). \tag{2.16}$$

Taking logarithms and collecting terms independent of θ in C, we can write

$$\log\Big(p(\theta|\mathcal{D}, \mathcal{M})\Big) = \frac{1}{2\sigma^2}\sum_{i=1}^{N}\Big(f(\mathbf{x}_i, \theta) - y_i\Big)^2 + \frac{1}{2}\|\theta\|^2 + C. \tag{2.17}$$

When we compare the terms in Equations (2.14) and (2.17), we see that l_2-regularised optimisation of the squared error is equivalent to MAP inference with a Gaussian likelihood and Gaussian prior. There is a similar link between l_1-regularisation and the zero-mean Laplacian prior that can be used to enforce sparsity in θ. So MAP estimation of θ is equivalent to regularised minimisation of a training loss, where the likelihood is matched with the loss function and the prior with the regularisation term.

[1]We use the term "overfit" to refer loosely to the problem of the parameters being fit to non-generalisable properties, such as the noise in the training data.

2.2 Gaussian processes

Gaussian processes (GPs) (Rasmussen and Williams, 2006) are a class of models that allow us to perform Bayesian inference over functions. We will focus our discussion on GPs for regression, which is the most popular surrogate model used in BO.

A GP is defined as a *collection of random variables, any finite number of which have a joint Gaussian distribution* (Rasmussen and Williams, 2006, Sec. 2.2). A GP is fully specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \tag{2.18}$$

where

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})], \tag{2.19}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbb{E}[f(\mathbf{x}) - m(\mathbf{x})(f(\mathbf{x}') - m(\mathbf{x}'))]. \tag{2.20}$$

Note that we are considering a scalar function $f : \mathbb{R}^d \to \mathbb{R}$ here, as this is relevant to the applications in this thesis. GPs defined over vector-valued outputs are discussed in e.g. Journel and Huijbregts (1978).

The mean and covariance functions allow us to define prior beliefs about the function $f(\mathbf{x})$ that are then combined with observed data, resulting in a posterior distribution over f. Note that the GP is a distribution over *functions*, analogous to how the Gaussian distribution is a distribution for a random variable. Setting the mean function to zero for notational simplicity, the joint distribution between the function values at observed locations $\mathbf{X}$ and a new location $\mathbf{x}^*$ can then be written as

$$\begin{bmatrix} f(\mathbf{X}) \\ f(\mathbf{x}^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) & k(\mathbf{X}, \mathbf{x}^*) \\ k(\mathbf{x}^*, \mathbf{X}) & k(\mathbf{x}^*, \mathbf{x}^*)) \end{bmatrix}\right), \tag{2.21}$$

where $k(\mathbf{X}, \mathbf{X})$ is the matrix made up by evaluating the covariance function, or kernel, on all pairs of input locations in the training data and $k(\mathbf{X}, \mathbf{x}^*)$ is the matrix of kernel evaluations between the training data and the new location. The kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that defines the similarity of two locations $\mathcal{X}$ and allows us to define descriptive concepts about the functions being modelled, such

as smoothness, periodicity, stationarity etc. Many kernels have hyperparameters that allow us to define these properties a priori, or alternatively learn them from observed data. We will introduce some popular kernels in the following section.

Equation (2.21) describes the case where we have access to perfect realisations of the function f at the input locations $\mathbf{X}$, which is rarely the case. In most cases we interact with f via a noisy observation process. This observation process is referred to as the *link function*, and describes how our N observations $\mathbf{y} = \{y_1, \ldots, y_N\}$ at inputs $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ relate to their respective true function values $\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_N)\}$. Assuming additive Gaussian noise on the observations, i.e. $\mathbf{y} = f(\mathbf{x}) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$, the joint distribution becomes:

$$\begin{bmatrix} \mathbf{y} \\ f(\mathbf{x}^*) \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I & k(\mathbf{X}, \mathbf{x}^*) \\ k(\mathbf{x}^*, \mathbf{X}) & k(\mathbf{x}^*, \mathbf{x}^*)) \end{bmatrix}\right), \tag{2.22}$$

Using standard results of Gaussian random variables (Rasmussen and Williams, 2006), we can write the distribution over the function values $f(\mathbf{x}^*)$ conditioned on the observations as

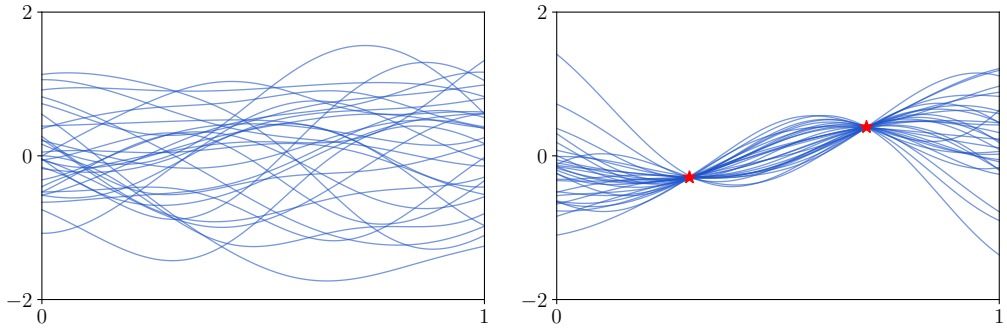
$$p(f(\mathbf{x}^*)|\mathbf{x}^*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}\Big(\mu(\mathbf{x}^*), \mathbf{C}(\mathbf{x}^*)\Big), \tag{2.23}$$

$$\mu(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I)^{-1}\mathbf{y}, \tag{2.24}$$

$$\mathbf{C}(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{x}^*) - k(\mathbf{x}^*, \mathbf{X})(k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I)^{-1}k(\mathbf{X}, \mathbf{x}^*). \tag{2.25}$$

Assuming additive Gaussian noise allows us to write the posterior in closed form, but if the observation process is non-Gaussian, e.g. sigmoidal for classification, then the posterior may not necessarily have a closed-form solution and would require approximations or sampling (Rasmussen and Williams, 2006, Ch. 3).

Figure 2.1 demonstrates how a GP updates its beliefs as a result of observing data. The prior in Figure 2.1a encodes the class of functions, before any data has been observed. Conditioning on the observations has the effect of making certain functions more probable than others, by "clamping" the posterior to the observed values, which is reflected in the shape of the sampled functions in Figure 2.1b.



(a) Prior **(b)** Posterior

Figure 2.1: Sampled functions from a GP (a) before and (b) after conditioning on two observations (red). After observing the data, the posterior is restricted to functions that pass through (or close to) the observed data.

2.2.1 Covariance kernels

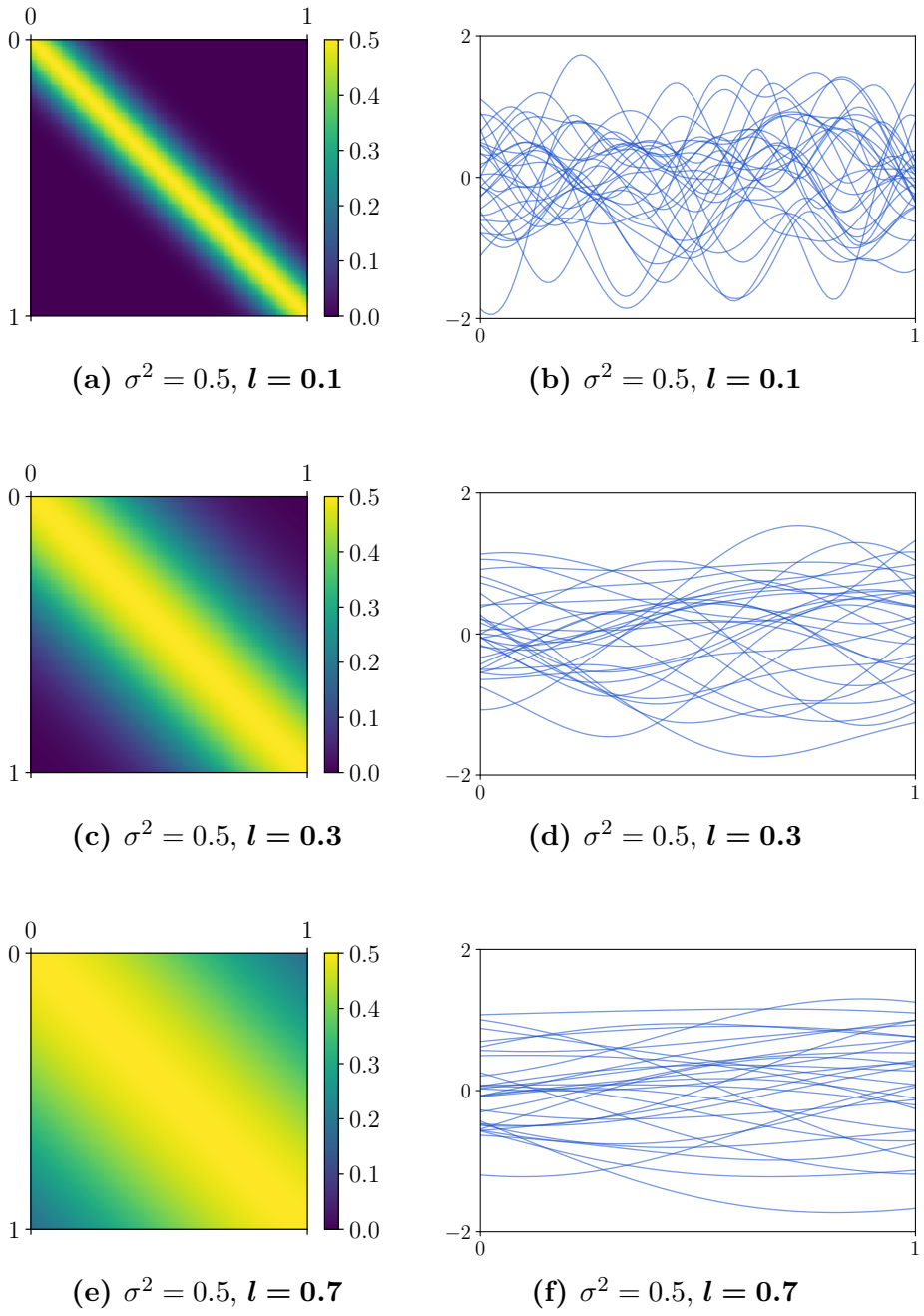
The covariance kernel is the key component that allows us to provide domain-specific information to the GP, as it encodes the class of functions being modelled. It allows us to encode higher-order properties about the function we are modelling, such as smoothness, periodicity/seasonality, range of function values, etc. There are a few kernels that are particularly popular, so we will discuss them here. We will focus on *stationary* kernels, which are all parameterised by the distance between input locations, and hence are invariant to translation in the input space. These kernels make use of the Mahalanobis distance r (Mahalanobis, 1936), which for two locations $\mathbf{x}$ and $\mathbf{x}'$ is defined as

$$r^2 = (\mathbf{x} - \mathbf{x}')^\top \boldsymbol{\Lambda}^{-1} (\mathbf{x} - \mathbf{x}'), \tag{2.26}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with the squared lengthscale hyperparameters $[l_1^2, \ldots, l_d^2]$ for each input dimension. If we assume that $l_i = l$ for $i = 1, \ldots, d$, then $\boldsymbol{\Lambda} = l^2 I$ giving us an isotropic kernel. Kernels with unique values for each dimension are also known as *automatic relevance determination* kernels (Rasmussen and Williams, 2006, Ch. 4), as the lengthscales also simultaneously provide feature relevance information.

RBF kernel

The radial-basis-function (RBF, squared exponential, exponentiated quadratic, Gaussian) kernel is by far the most popular kernel used in the GP literature.



(a) $\sigma^2 = 0.5$, $\boldsymbol{l = 0.1}$ **(b)** $\sigma^2 = 0.5$, $\boldsymbol{l = 0.1}$

(c) $\sigma^2 = 0.5$, $\boldsymbol{l = 0.3}$ **(d)** $\sigma^2 = 0.5$, $\boldsymbol{l = 0.3}$

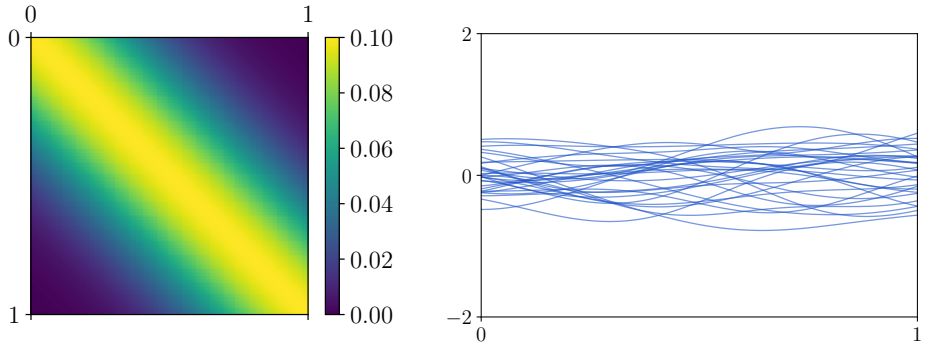
(e) $\sigma^2 = 0.5$, $\boldsymbol{l = 0.7}$ **(f)** $\sigma^2 = 0.5$, $\boldsymbol{l = 0.7}$

Figure 2.2: Kernel matrix $k(\mathbf{X}, \mathbf{X})$ (left) and sample functions (right) drawn from RBF kernels with different lengthscales on an equally-spaced grid $\mathbf{X} \in [0, 1]$. As the lengthscale increases, the variability of the functions reduces and more long-term trends are sampled.

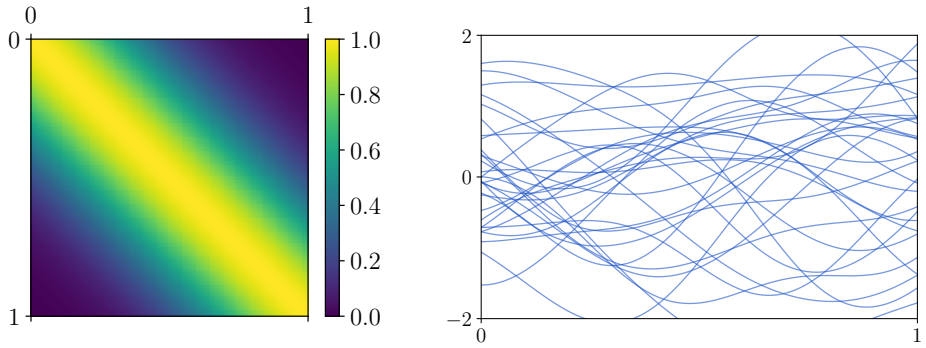
It is defined as

$$k_{\text{rbf}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2}r^2\right). \tag{2.27}$$

When we use the RBF kernel, we are asserting that f is a smooth, infinitely-differentiable function. This is rarely accurate in cases where f is a real-world phenomenon, but despite that the RBF remains the default kernel for many users.



(a) $\boldsymbol{\sigma^2 = 0.1}$, $l = 0.3$ **(b)** $\boldsymbol{\sigma^2 = 0.1}$, $l = 0.3$



(c) $\boldsymbol{\sigma^2 = 1.0}$, $l = 0.3$ **(d)** $\boldsymbol{\sigma^2 = 1.0}$, $l = 0.3$

Figure 2.3: Kernel matrix $k(\mathbf{X}, \mathbf{X})$ (left) and sample functions (right) drawn from RBF kernels with different variances on an equally-spaced grid $\mathbf{X} \in [0, 1]$. Note the different ranges of values between the two kernel matrices.

The RBF kernel in one dimension has two hyperparameters. The variance, σ^2, controls the output scale of the function, and the lengthscale, l, controls the degree of variability of the function across the input domain. Figure 2.2 shows how the lengthscale of the RBF kernel leads to different types of functions by drawing samples from the prior on a grid in the range $[0, 1]$. Figures 2.2a, 2.2c and 2.2e show the

values of the kernel matrix $k(\mathbf{X}, \mathbf{X})$ for lengthscales $l \in \{0.1, 0.3, 0.7\}$ respectively. We drew sample functions from each kernel and show them in Figures 2.2b, 2.2d and 2.2f for the three lengthscales. Figure 2.3 similarly shows the effect of the variance parameters $\sigma^2 \in \{0.1, 1\}$.

A high lengthscale leads to slowly-varying functions, as the kernel's similarity measure between points is larger (the diagonal in Figure 2.2e is much wider than in Figure 2.2a). As the lengthscale decreases, the functions vary more in the same input range. The effect of the variance parameter is straightforward, as it simply scales the output value of the sampled functions.

Matérn kernel

Another popular class of kernels is the Matérn kernel. In addition to the lengthscale and variance, the Matérn kernel also has a smoothness parameter ν and is defined as

$$k_\nu(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu r}\right)^\nu K_\nu \left(\sqrt{2\nu r}\right), \tag{2.28}$$

where $K_\nu(\cdot)$ is a modified Bessel function of the second kind.

By choosing different values for ν we can control the smoothness of the function: $\nu \in \{3/2, 5/2\}$ correspond to functions that are once- and twice-differentiable respectively, and those variants are referred to as the Matérn-3/2 and Matérn-5/2 kernels:

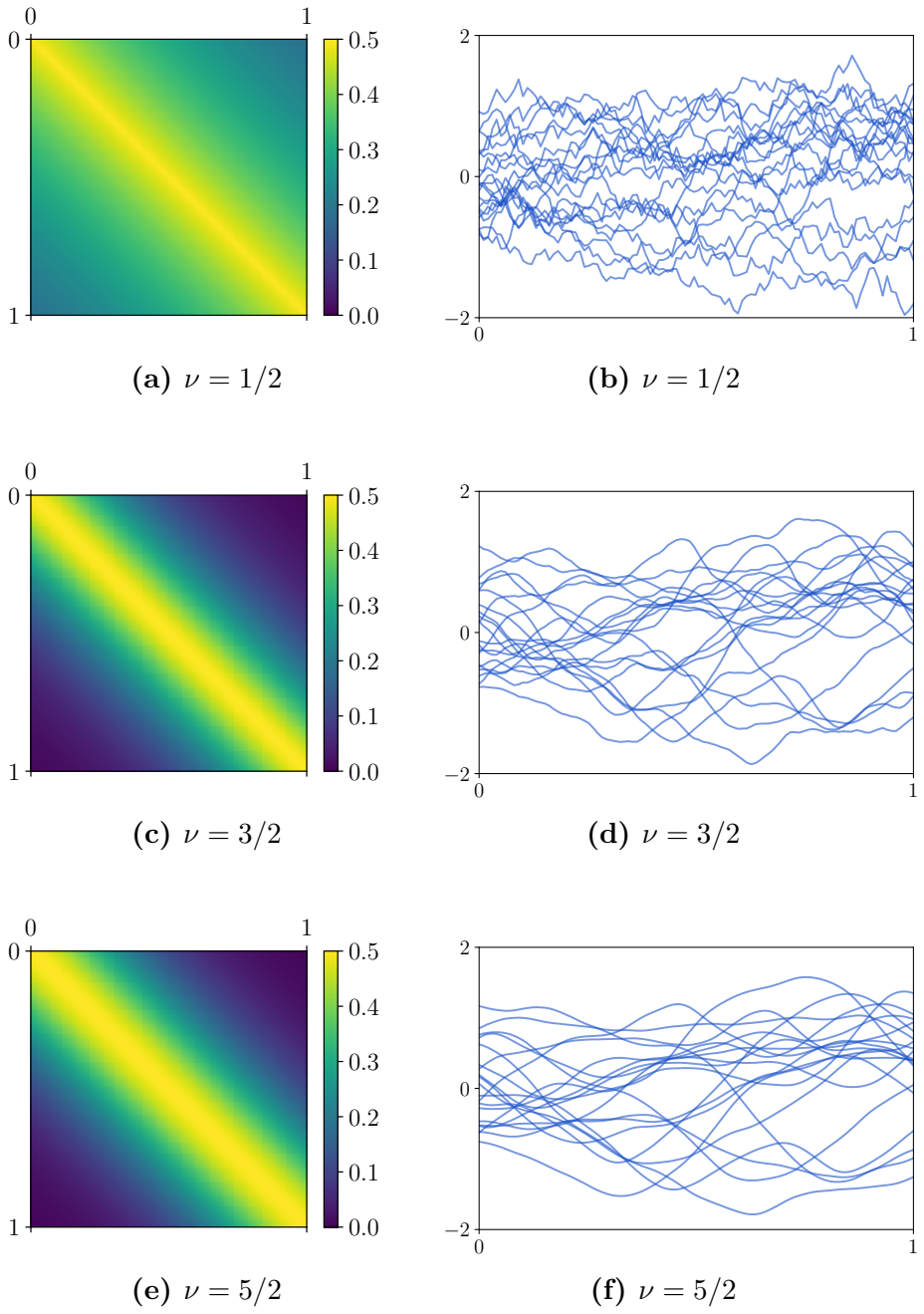
$$k_{\frac{3}{2}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\sqrt{3}r)(1 + \sqrt{3}r), \tag{2.29}$$

$$k_{\frac{5}{2}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-\sqrt{5}r)\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right). \tag{2.30}$$

These kernels are often used to model physical phenomena, due to the limited differentiability and smoothness properties of these kernels. These properties make them a popular choice GP surrogates when used in the context of a Bayesian optimisation surrogate (see Section 2.3).

Setting $\nu = 1/2$ results in the Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) for $d = 1$

$$k_{\frac{1}{2}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp(-r), \tag{2.31}$$



(a) $\nu = 1/2$ **(b)** $\nu = 1/2$

(c) $\nu = 3/2$ **(d)** $\nu = 3/2$

(e) $\nu = 5/2$ **(f)** $\nu = 5/2$

Figure 2.4: Kernel matrix $k(\mathbf{X}, \mathbf{X})$ (left) and sample functions (right) drawn from Matérn kernels with different values of ν on an equally-spaced grid $\mathbf{X} \in [0, 1]$. The functions sampled from the Maté-32 and Maté-52 kernels are less smooth than the RBF samples, due to their limited differentiability.

and when $\nu \to \infty$ the kernel becomes the RBF kernel

$$k_\infty(\mathbf{x}, \mathbf{x}') = k_{\text{rbf}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{1}{2} r^2\right). \tag{2.32}$$

Figure 2.4 shows example functions drawn from the Matérn kernel for different values of the smoothness parameter ν. Note how, as ν increases, the functions become smoother. The Matérn kernel is often used when real-world phenomena are being modelled, as its smoothness and finite differentiability more closely model the data in question.

2.2.2 Inference of GP hyperparameters

The discussion until now focussed on GP inference with a known kernel, and hence known hyperparameters. If enough information is known about the problem being modelled, then the kernel parameters may be able to be defined a priori, but this is rarely the case in practice. Particularly in the case of black-box optimisation, where we might only know that the unknown function f is smooth. In such cases, we can utilise Bayesian techniques to perform inference over the hyperparameters θ by solving the following equation:

$$p(\theta|\mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{X}, \theta)p(\theta)}{p(\mathbf{y}|\mathbf{X})}. \tag{2.33}$$

Evaluating the term $p(\mathbf{y}|\mathbf{X})$ in order to perform full Bayesian inference is challenging, as it involves integrating out the kernel hyperparameters

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{X}, \theta)p(\theta)\mathrm{d}\theta. \tag{2.34}$$

Finding a conjugate prior that would simplify computation of $p(\mathbf{y}|\mathbf{X})$ is also not possible, due to the complex dependency of $p(\mathbf{y}|\mathbf{X}, \theta)$ on the hyperparameters via the inverse of the kernel matrix. We can instead focus on using the numerator to find the optimal values for θ, as the denominator is a fixed quantity independent of the hyperparameters. Taking logarithms and focussing on terms that are dependent on θ, we can write

$$\log p(\theta|\mathbf{y}, \mathbf{X}) \propto \log p(\mathbf{y}|\mathbf{X}, \theta) + \log p(\theta), \tag{2.35}$$

where $\log p(\mathbf{y}|\mathbf{X}, \theta)$ is referred to as the marginal (log) likelihood (Rasmussen and Williams, 2006), and $p(\theta)$ is the hyperparameter prior. The marginal likelihood

is a Gaussian distribution in the observations $\mathbf{y}$

$$\mathcal{L} = \log p(\mathbf{y}|\mathbf{X}, \theta) \tag{2.36}$$

$$= -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi, \tag{2.37}$$

where $K_y = k(\mathbf{X}, \mathbf{X}) + \sigma_n^2 I$ is the covariance matrix including the observation noise process. In the presence of an uninformative prior, e.g. a uniform distribution for $p(\theta)$, finding the maximum of the marginal log likelihood is equivalent to finding the MLE estimate of the hyperparameters. If we have an informative prior, then we simply maximise the joint distribution to perform MAP inference:

$$\hat{\theta}_{\text{MLE}} = \arg\max_\theta \Big(\log p(\mathbf{y}|\mathbf{X}, \theta) \Big), \tag{2.38}$$

$$\hat{\theta}_{\text{MAP}} = \arg\max_\theta \Big(\log p(\mathbf{y}|\mathbf{X}, \theta) + \log p(\theta) \Big). \tag{2.39}$$

The maximisation of the MLE or MAP objectives is often performed using gradient-based optimisers, which require the derivative of Equation (2.37) w.r.t. each hyperparameter θ_j (Rasmussen and Williams, 2006, Equation 5.9):

$$\frac{\partial}{\partial \theta_j}\log p(\mathbf{y}|X, \boldsymbol{\theta}) = \frac{1}{2}\mathbf{y}^\top K_y^{-1}\frac{\partial K_y}{\partial \theta_j}K_y^{-1}\mathbf{y} - \frac{1}{2}\operatorname{tr}\left(K_y^{-1}\frac{\partial K_y}{\partial \theta_j}\right) \tag{2.40}$$

$$= \frac{1}{2}\operatorname{tr}\left(\left(\boldsymbol{\alpha}\boldsymbol{\alpha}^\top - K_y^{-1}\right)\frac{\partial K_y}{\partial \theta_j}\right), \tag{2.41}$$

where $\boldsymbol{\alpha} = K_y^{-1}\mathbf{y}$.

2.3 Bayesian optimisation

When building machine learning systems, we invariably make choices about the class of models being used as well as their hyperparameters, before providing data to the models to train their parameters. We differentiate between parameters and hyperparameters. We will use "parameters" to refer to variables such as weights of neurons in neural networks, or the weights in linear or logistic regressors, and "hyperparameters" to refer to higher-level decisions that affect the structure of the model, such as the number of neurons and layers of a neural network, the number of trees in a random forest or the choice of optimisation routine or preprocessing

algorithms. In general we can view hyperparameters as choices that are made first, as they define the parameter space of the model. For hyperparameter optimisation, we are ultimately interested in the equation

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} f(\mathbf{x}), \tag{2.42}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the performance metric that we wish to optimise and $\mathbf{x}$ are the d hyperparameters.

While gradient-based local optimisation methods are often used for training model parameters, hyperparameter optimisation is less amenable to such methods. Gradient-based methods are suitable for functions that evaluate quickly, which is rarely the case when we are evaluating a selection of hyperparameters, as we need to train the model and then compute the score.

This motivates the use of efficient global optimisation techniques, that are also efficient in the number of required samples, as this would allow us to identify an optimum while minimising the number of training runs of the model. Bayesian optimisation (BO) (Osborne et al., 2009; Brochu et al., 2010; Shahriari et al., 2016) is one such efficient global optimisation technique.

Global optimisation tasks exist in many different areas of research and industry, but BO has recently enjoyed particular success in the domain of machine learning hyperparameter optimisation (Snoek et al., 2012; Swersky et al., 2013; Thornton et al., 2013; Bergstra et al., 2013; Swersky et al., 2014; Gardner et al., 2014; Klein et al., 2016). For example, BO found better hyperparameter settings for deep convolutional neural networks evaluated on the CIFAR-10 (Krizhevsky, 2009) task without data augmentation (Snoek et al., 2012; Domhan et al., 2015). It was also used to select hyperparameters of Google Deepmind's AlphaGo agent (Chen et al., 2018). In such cases, the function f being sampled is the accuracy score of a model, after training it on a (usually large) training set and then evaluating the model performance on a held-out test set.

The BO algorithm, described in pseudocode in Algorithm 1, depends on two components, a probabilistic surrogate model and an acquisition function. The

surrogate is a regression model that combines our prior beliefs with the information gained from observations of f. The acquisition function leverages this surrogate model to select the configuration that will provide the most information about the location of the optimum of $f(\mathbf{x})$. It is therefore important to choose a surrogate model, most often a GP, that is able to reason well about f, as well as selecting an acquisition function that can identify a good sequence of experiments to perform.

It is important to note that, while (global) hyperparameter optimisation is a challenging task, the task is made even more challenging if we optimising many different hyperparameters (e.g. more than 10). Learning useful GP models using the popular Matérn kernel, and many other stationary kernels, can be challenging in a high-dimensional space. We present some approaches that were developed to mitigate this weakness in Section 2.3.2.

Algorithm 1 Generic Bayesian optimisation algorithm

1: **Input:** A black-box function f, observation data $\mathcal{D}_0$, acquisition function α, budget T
2: **Output:** The best recommendation $\mathbf{x}^*$
3: **for** $t = 1, \ldots, T$ **do**
4: Create surrogate model given current observations $\mathcal{M}_{t|t-1} = p(f|\mathcal{D}_{t-1})$
5: Select the next best evaluation $\mathbf{x}_t = \arg\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{M}_{t|t-1})$
6: Evaluate $y_t = f(\mathbf{x}_t)$
7: $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{\mathbf{x}_t, y_t\}$
8: **end for**
9: **return** $\mathbf{x}^*$ corresponding to the best value in $\mathcal{D}_T$

2.3.1 Acquisition functions

Having decided to use a GP as the surrogate, we need to choose the acquisition function, $\alpha(\mathbf{x})$. The acquisition function scores each potential input location and its maximum corresponds to the configuration that will provide the most information about the optimum:

$$\mathbf{x}_t = \arg\max_{\mathbf{x}} \alpha(\mathbf{x}|\mathcal{D}_{t-1}). \tag{2.43}$$

There exist a number of different acquisition functions that are derived by minimising the cumulative or simple regret, which (loosely defined) is a measure for how much

worse the algorithm performs compared to the best possible outcome, or satisfy other desirable properties. The key requirements for an acquisition function are that it should be cheap to compute, and that it should balance exploration and exploitation.

The acquisition function needs to be cheap to compute, as it is optimised over the search space of $\mathbf{x}$. The acquisition function surface is often highly non-convex with multiple local optima, so popular global optimisers for this task include DIRECT (Jones et al., 1993) and multi-started gradient descent (GPyOpt authors, 2016).

Along with being cheap to compute, the acquisition function also needs to identify configurations that provide information about the location of the optimum over the course of the algorithm. This can be in the form of configurations that lie close to a region we have identified as likely to contain the optimum ("exploitation"), in order to refine our knowledge of f in that region. Alternatively, it could select configurations that lie far away from existing data, in order to discover potentially new fruitful regions in the search space ("exploration").

Exploration and exploitation are useful, but fundamentally opposite behaviours, so the task of balancing them falls to the acquisition function. In the best case, we would expect a BO procedure to spend most of the early iterations exploring the search space, and then gradually shifting to a more exploitative behaviour as we approach the end of the optimisation algorithm. We now introduce some popular acquisition functions below.

Probability of improvement

Arguably the earliest acquisition function is the probability of improvement (PI) (Kushner, 1964)

$$\alpha_{\mathrm{PI}}(\mathbf{x}) = p(f(\mathbf{x}) \geq \eta) = \Phi\left(\frac{\mu(\mathbf{x}) - \eta}{\sigma(\mathbf{x})}\right), \tag{2.44}$$

where η is the current incumbent best function value and Φ is the cumulative standard Gaussian distribution. This acquisition function simply looks for values that may have a larger value than the incumbent best with high probability, regardless of how large the potential improvement may be. This may lead to

locations that are highly likely to be slightly larger than the incumbent to be selected over locations where the value is less certain, but potentially more favourable.

Expected improvement

Building on the formulation of PI, the expected improvement (EI) criterion (Jones et al., 1998) takes into account the magnitude of the improvement as well as the certainty:

$$\alpha_{\mathrm{EI}}\left(\mathbf{x}; \mathcal{D}_n\right) = E\left[\max\left(f(\mathbf{x}) - \eta\right); \mathcal{D}_n\right] \tag{2.45}$$

$$= \left(\mu(\mathbf{x}) - \eta\right) \Phi\left(\frac{\mu(\mathbf{x}) - \eta}{\sigma(\mathbf{x})}\right) + \sigma(\mathbf{x}) \phi\left(\frac{\mu(\mathbf{x}) - \eta}{\sigma(\mathbf{x})}\right), \tag{2.46}$$

where ϕ is the standard Gaussian probability distribution. This is one of the most popular acquisition functions, perhaps in part because it has no parameters that need to be chosen by the user. The functional form of the EI acquisition function shows how exploration and exploitation are balanced automatically. The first term scores locations where the posterior mean is is more promising (exploitation of our current knowledge) and the second term scores unexplored, uncertain areas (exploration). In the early iterations of a BO procedure, the uncertainty (second term) will dominate the acquisition function's value, and lead to locations in unexplored regions being selected. As more data is acquired, the function's predicted value (first term) becomes increasingly dominant, as the uncertainty across the search space decreases, gradually shifting the selection towards a more exploitative behaviour.

Despite this balance of exploration and exploitation, it has been shown to weight its selections more towards exploitation (Calandra et al., 2014). A recent adaptation called ScaledEI (Noè and Husmeier, 2018) was developed that may discourage this overly exploitative behaviour. The authors defined their acquisition function to take into account both the expectation of the improvement (as in Equation (2.46)) as well as the variance of the improvement, which encouraged areas of higher uncertainty more than the standard EI formulation.

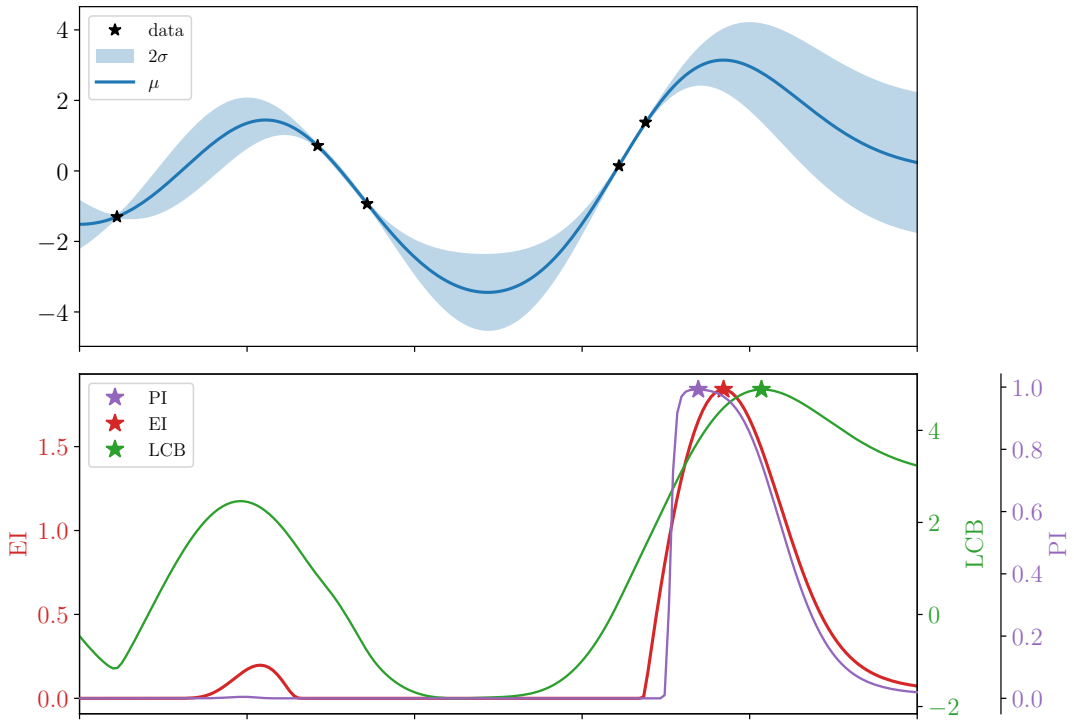


Figure 2.5: PI, EI and LCB ($\kappa = 2$) in action. Top: Surrogate and current data. Bottom: Acquisition functions. Note how PI and EI choose a location quite close to the data (exploitative) and LCB is more exploratory. The acquisition surfaces for PI and EI are characteristically defined by flat regions, making them challenging to optimise.

Upper/Lower confidence bound

Cox and John (1992) introduced an acquisition function termed lower confidence bound (LCB), which is defined as

$$\alpha_{\text{LCB}} = \mu(\mathbf{x}) + \kappa\sigma(\mathbf{x}), \tag{2.47}$$

where the parameter κ controls affects how much the algorithm will explore vs exploit. Higher values of κ increase the effect of the posterior uncertainty, and hence lead to more exploratory behaviour.

In the case of minimisation, the acquisition function is referred to as upper confidence bound (UCB) and is defined as

$$\alpha_{\text{UCB}} = \mu(\mathbf{x}) - \kappa\sigma(\mathbf{x}). \tag{2.48}$$

An added benefit of using LCB is the fact that strong theoretical results regarding convergence have been developed (Srinivas et al., 2010; De Freitas et al., 2012).

Srinivas et al. (2010) showed that the convergence rate of BO is $\mathcal{O}\left(\sqrt{\frac{d \ln T}{T}}\right)$ where T is the number of samples, and the optimal convergence rate for fixed surrogate hyperparameters is shown to be $O(T^{-\nu/d})$, where ν is the smoothness parameter of the Matérn kernel (Bull, 2011).

Figure 2.5 shows the EI and LCB acquisition functions in practice. The top figure shows the data and the surrogate, and the bottom plot shows the values of the two acquisition functions across the search domain. The shape of EI in the figure is often observed in practice, namely it is mostly flat across most of the search space with one pronounced mode. This can be particularly challenging to optimise. The shape of the LCB function is more smoothly varying across the domain, but also illustrates the highly non-convex nature of the optimisation problem mentioned earlier.

Information-theoretic acquisition functions

In contrast to the methods introduced so far, information-theoretic acquisition functions consider the posterior over the location of the optimum value $p(\mathbf{x}^*|\mathcal{D})$. Our belief over f in the form of the probabilistic surrogate model implicitly induces this posterior over $\mathbf{x}^*$. Entropy search (Hennig and Schuler, 2012) selects locations in order to minimise the entropy of the optimum's posterior distribution. However, this distribution $p(\mathbf{x}^*|\mathcal{D})$ is not analytically tractable, so approximations must be made. Extensions to this idea have been made by Hernández-Lobato et al. (2014) and Wang and Jegelka (2017), but in general, these methods are more challenging to compute than the previously-mentioned alternatives, so EI and LCB remain the default choice for acquisition functions.

Choosing the acquisition function

We have presented a number of different acquisition functions, but this raises the question of how to choose the best acquisition function for a given task? This remains an open question. The choice of acquisition function often depends on the personal preference of a practitioner, or on properties of the shape of the acquisition function that may make it a better fit for a certain application. We make one such choice in our work in Chapter 4. We require a multi-modal acquisition function

for the BO algorithm to identify a diverse set of configurations to test in parallel, so we selected the UCB acquisition function.

Similarly, the choice of surrogate model, whether GP, Random forest, neural network or a different class of models altogether, also falls to personal preferences at this stage. Developing guidelines for when, given an optimisation task, certain acquisition functions and surrogate models are more suitable than others is an open question, but one with the potential for significant impact in this area of research. The work by McLeod et al. (2018) touches upon this question indirectly by developing an algorithm that chooses between BO and local optimisation.

2.3.2 Research trends

Having provided an overview of BO, we will now highlight some trends and open questions in the research area.

Stopping criterion Our discussion so far has focussed on applying BO given a fixed budget of T evaluations. Popular gradient-based local optimisation algorithms that are used to optimise machine learning parameters often contain early-stopping criteria, that interrupt the optimisation when no further benefit is likely to be achieved by continuing the algorithm. Developing such a stopping criterion for BO remains an open question, with some strides having been made towards this goal by McLeod et al. (2018), where the authors propose a criterion to switch from global BO to local gradient-based optimisation.

High-dimensional BO BO algorithms in most cases make use of a GP surrogate. GPs have many properties that make them well-suited for this task, but they often struggle in cases where the input dimensionality is high. The use of the Mahalanobis distance in stationary kernels means GP surrogates are particularly sensitive to the "curse of dimensionality", which states that the complexity of a search space, and hence also the distances therein, grows exponentially as we increase the dimensionality. With larger distances between points, this means that the kernel will struggle to learn useful relationships

from the data and will therefore be less useful. Some recent approaches tackle this problem using random embeddings (Wang et al., 2016) and searching within a sampled low-dimensional space, learning additive structures of the hyperparameters (Kandasamy et al., 2015; Gardner et al., 2017; Rolland et al., 2018) to reduce the distances computed inside the kernel, or optimising subsets of the parameter space (Li et al., 2018).

Non-myopic BO BO is a sequential, myopic optimisation algorithm. Maximising the EI or PI criterion assumes that the current sample is the last one. Srinivas et al. (2010) proposed a schedule for the tradeoff parameter in LCB that reduces the contribution of the posterior variance as the algorithm progresses, but in most practical applications, the LCB parameter is fixed to the same value for all iterations. Using knowledge of the remaining budget when selecting locations seems natural, as this would provide additional motivation for exploratory or exploitative behaviour towards the beginning and end of the BO algorithm respectively. González et al. (2016) proposed a lookahead BO procedure called GLASSES that considers not only the current iteration, but extends the decision horizon beyond the myopic to consider "dozens of future evaluations".

Scaling BO to larger sample sizes BO is often applicable when the function f is very expensive to compute, such that we do not expect large volumes of data to be provided to the optimisation algorithm. This has meant that much research effort has been focussed on the acquisition function and related algorithmic concerns, while accepting the use of a GP surrogate. Using a GP does however unnecessarily restrict the applicability of BO exclusively to the small-data domain due to GP posterior inference scaling cubically with the number of training points, unless we resort to approximations (Hensman et al., 2013). There has been some work that explored the use of neural networks as alternative surrogates (Snoek et al., 2015) and the more recent interest in

Bayesian neural networks (Gal and Ghahramani, 2016) may provide fruitful directions for research.

Multi-task BO In this thesis we are concerned with the optimisation of a single objective function, f, but in other applications the objective may combine multiple different cost functions together, e.g. if we wish to find a model architecture that performs well on more than one task (Swersky et al., 2013). The existence of multiple cost functions that may conflict with each other turns the task from finding a single optimum location into one of discovery of a Pareto front, which is the set of all Pareto optimal points (Paria et al., 2018).

2.4 Multi-armed bandits

We now introduce another class of algorithms concerned with balancing exploration and exploitation, which will be relevant to our discussion in Chapter 5: the multi-armed bandit (MAB) (Robbins, 1952; Gittins and Jones, 1979; Berry and Fristedt, 1985). MABs are concerned with selecting a sequence of actions from a discrete action space in order to maximise the cumulative reward, or equivalently to minimise the cumulative regret [2]. This differs from BO, which considers only the immediate regret, as well as the fact that BO operates on a continuous space and MABs on a discrete set of options.

MABs can be thought of simulating a gambler's decision-making process when faced with a fruit machine with multiple levers. The gambler has to decide between either learning about the different levers' reward distributions or exploiting his current knowledge of the value of the possible actions could be taken, with the overall aim being maximising the total monetary reward at the end. This scenario is applicable to a wide range of real-world problems, and has received a lot of attention from the research community as a result. Successful industrial applications include

[2] We define "regret" to be the difference in utility between a selected action compared to the best possible action that could have been selected in hindsight.

online advertisements (Radlinski et al., 2008), crowd sourcing (Liu and Liu, 2015) and dynamic pricing problems (Sauré and Zeevi, 2013; Badanidiyuru et al., 2013).

In the medical domain, MABs are a good fit for driving clinical trials, as the aims of the algorithm, namely the exploration-exploitation tradeoff while maximising the total number of successes along the way, align particularly well with those of such medical trials (Chow and Chang, 2008; Gittins et al., 2011)

A MAB aims to maximise the cumulative reward over the course of T iterations, in which we select one of K possible actions out of a pre-defined action space $\mathcal{A} = \{a_1, \ldots, a_K\}$ per iteration. When action a_j is taken in iteration t, we observe a reward $r_{j,t}$ which is drawn from the action's associated (unknown) reward distribution $\mathcal{R}_j \in \{\mathcal{R}_1, \ldots, \mathcal{R}_K\}$, parameterised by $\theta_j \in \{\theta_1, \ldots, \theta_K\}$. Observing the reward allows us to update our beliefs about the action's reward distribution, which in turn informs our decisions in subsequent iterations.

Most MAB algorithms assume that the rewards for the different actions are sampled independently and each action's reward distribution has fixed parameters (i.i.d.). This applies to the ϵ-first, ϵ-greedy (Watkins, 1989) and UCB (Auer et al., 2002a) algorithms. Assuming i.i.d. rewards may not always be applicable to typical real-world tasks, and a further class of MAB algorithms was developed that relax the independence assumption by assuming instead that the rewards are adversarial. One such algorithm is EXP3 (Auer et al., 2002b), which assumes that the action-reward pairs have been fixed beforehand by an adversary. This can be seen as solving a worst-case scenario, with the odds of success stacked against us. In the sections below we introduce the MAB algorithms we mentioned relating to i.i.d. and adversarial rewards. Further generalisations exist, for example the reward distributions themselves may evolve over time, which is referred to as the restless bandit problem, (Whittle, 1988). This could apply in applications for managing financial portfolios or online advertisements. In other cases, we may receive additional information or hints at each iteration, also called a context, which may help us select the optimal action. Algorithms solving this class of problems are known as contextual bandits (Abe et al., 2003).

2.4.1 ϵ-strategies

The simplest approach to solve the gambler's dilemma is to use a greedy approach (Algorithm 2). We first collect one reward from each arm by performing each action in $\mathcal{A}$ once, which takes K iterations. Thereafter we select the action with the highest sampled reward for the remainder of the budget. It is evident that this is a highly exploitative algorithm, since the reward distributions are explored only in the first K iterations. This initial exploration is insufficient, since we believe that the rewards are not fixed, but rather they stem from unknown distributions. One sample is not sufficient to give us meaningful information about the reward distribution, which renders the greedy strategy suboptimal (Sutton et al., 1998; Scott, 2010). The ϵ-strategies, where $0 < \epsilon < 1$, we discuss below improve upon the greedy approach by introducing elements that allow for more exploration.

Algorithm 2 Greedy MAB

1: **for** $t = 1, \ldots, K$ **do**
2: Perform action a_t and note the reward r_t.
3: **end for**
4: Identify action $a_{\text{best}} = a_j$ where $j = \arg\max_j \{r_1, \ldots, r_K\}$.
5: **for** $t = K + 1, \ldots, T$ **do**
6: Perform action a_{best}.
7: **end for**

ϵ-first

The ϵ-first strategy (Algorithm 3) extends the exploration phase of the greedy strategy to ϵT steps, and performs random selections in this period. Once the exploration budget is exhausted, the remaining iterations perform the action with the highest mean reward μ_i. One problem with this approach is that its performance is likely to be poor in the exploration phase (Slivkins, 2019), as the first samples are taken without any consideration of the rewards each action has received. Also, it is likely to perform poorly if there is any drift in the reward distributions across iterations, as learning only occurs at the start of the algorithm.

Algorithm 3 ϵ-first

for $t = 1, \ldots, \epsilon T$ **do**
  Select an action uniformly at random from $\mathcal{A}$.
end for
Compute the mean rewards $\{\mu_1, \ldots, \mu_K\}$ for each action.
Identify action $a_{\text{best}} = a_j$ where $j = \arg\max_j \{\mu_1, \ldots, \mu_K\}$.
for $t = \epsilon T, \ldots, T$ **do**
  Perform action a_{best}.
end for

ϵ-greedy

Rather than front-loading the exploration phase, the ϵ-greedy strategy (Algorithm 4) spreads out this behaviour more evenly across the iterations (Watkins, 1989). In each round a uniformly-sampled random variable u decides between exploration and exploitation. If $u < \epsilon$, then the MAB will perform an exploratory step by selecting an action uniformly at random from the available actions $\mathcal{A}$. Otherwise the action that has the best average reward so far will be selected.

Algorithm 4 ϵ-greedy

for $t = 1, \ldots, T$ **do**
  Sample u uniformly in the range $[0, 1]$.
  if $u < \epsilon$ **then**
    Select an action uniformly at random from $\mathcal{A}$.
  else
    Perform action with the highest average reward so far.
  end if
end for

The ϵ-greedy strategy has the benefit of guaranteeing that we try every action as $t \to \infty$, but this can simultaneously also be a disadvantage. Once the optimal action is known, we should exclusively perform that action until we exhaust our budget, as this would maximise the cumulative reward. The ϵ-greedy strategy will instead continue sampling the non-optimal action space with probability ϵ. This observation led to the development of ϵ-decreasing strategies, that define a schedule to reduce ϵ as t increases (Auer et al., 2002a; Vermorel and Mohri, 2005). On the other hand, if the reward distributions change over time, then the ϵ-greedy strategy

is able to capture this drift to a certain degree, as there is a finite chance that it will take an action that differs from what is believed to be the optimal action at the time.

2.4.2 Upper confidence bound

Another class of MAB algorithms called upper confidence bounds (UCB) was developed by Lai and Robbins (1985). The authors showed that asymptotically, this strategy selects the best action (i.e. the action with the highest expected reward) exponentially more often than the others. The UCB algorithm in Lai and Robbins (1985) is difficult to compute, as it relies on the entire sequence of rewards generated from every action up until now, so Auer et al. (2002a) introduced a number of UCB-based strategies that were more scalable. The first one, called UCB1, is widely-used and often simply referred to as UCB.

This algorithm's approach can be summarised as "optimism in the face of uncertainty". Despite our limited knowledge, at each iteration we create optimistic estimates of the probable payoff of the different actions and then pick the most promising one. The UCB1 strategy in Auer et al. (2002a) forms an upper confidence $u_{j,t}$ on the true expected reward for each of the actions, such that

$$\mathbb{E}[r_j] \leq \bar{r}_j(t) + u_j(t), \tag{2.49}$$

where $\bar{r}_j(t)$ is the sample mean of the rewards of action a_j up to iteration t. The upper bound for each action is defined as

$$u_j(t) = \sqrt{\frac{2 \ln t}{N_j(t)}}, \tag{2.50}$$

where $N_j(t)$ is the number of times action a_j has been performed so far. From the formulation of the upper bound, we can see that as we progress through time, the upper bound inflates the score of an action if it hasn't been chosen for some time (increasing t while $N_j(t)$ remains constant). It also reduces the contribution of the upper bound to the utility if an action it has been explored often (large $N_j(t)$), so the mean reward estimate $\bar{r}_j(t)$ dominates the score of the action.

This means that if we decide to perform a low-reward action, we will be less likely to repeat that action, the sample mean $\bar{r}_j(t)$ will be small and its upper bound $u_j(t)$ will also decrease. Guessing correctly will make that action more attractive by increasing $\bar{r}_j(t)$. UCB additionally prevents good actions from being penalised too harshly in case they performed badly early on, as the "optimism" term $u_j(t)$ will ensure that we get further chances to update the sample means of less-explored actions.

2.4.3 EXP3

So far the MAB methods have assumed i.i.d. rewards. We now discuss EXP3 (Auer et al., 2002b), which stands for "Exponential-weight algorithm for Exploration and Exploitation", and was developed to deal with the case where the rewards are adversarial. We mentioned that UCB can be seen as "optimism in the face of uncertainty". EXP3 is quite the opposite: it assumes the rewards have been defined adversarially, which is a more pessimistic world view. Real-world rewards rarely are truly adversarial, but instead often lie somewhere in the middle of adversarial and i.i.d. So a method developed to beat the worst-case adversarial setting will probably do well in an easier setting, where the rewards are not adversarially selected.

The adversarial bandit problem differs from the i.i.d. case in that we assume the problem is defined by the pair $(\mathcal{A}, \mathbf{r})$, where $\mathcal{A}$ is the action space with K actions, and $\mathbf{r}$ is a pre-defined sequence of action-reward pairs for each time step: $\mathbf{r} = \{\mathbf{r}(1), \ldots, \mathbf{r}(T)\}$, where $\mathbf{r}(t) = \{r_1(t), \ldots, r_K(t)\}$ are the rewards for each action at iteration t. Note that even though $\mathbf{r}$ defines the rewards for every action at every iteration, the MAB only gets to observe the reward for the action it performs at each iteration. The algorithm is presented in Algorithm 5.

The EXP3 algorithm defines an exploration parameter, $\gamma \in [0, 1]$, which controls the degree to which past experience influences future choices. Choosing $\gamma = 0$ corresponds to selecting actions based solely on observed rewards and $\gamma = 1$ ignores observed rewards and samples uniformly from the available actions instead, similar to the ϵ parameter in Section 2.4.1. Choosing a value within this range

Algorithm 5 EXP3

1: **Input:** $\gamma \in [0, 1]$.
2: Initialise the weights $w_1(1) = w_2(1) = \ldots = w_K(1) = 1$.
3: **for** $t = 1, \ldots, T$ **do**
4: Define the probabilities for each $i \in \{1, \ldots, K\}$ as

$$p_i(t) = (1 - \gamma)\frac{w_i(t)}{\sum_{j=1}^{K} w_j(t)} + \frac{\gamma}{K}.$$

5: Sample the action a_i from the discrete probability distribution

$$\mathbf{p}(t) = \{p_1(t), \ldots, p_K(t)\}.$$

6: Perform action a_i and observe the reward $r_i(t)$.
7: Define the estimated reward $\hat{r}_i(t) = r_i(t)/p_i(t)$.
8: Update the selected action's weight $w_i(t) = w_i(t) \exp[\gamma \hat{r}_i(t)/K]$.
9: **end for**

makes the algorithm more robust to observing a succession of low rewards from a high-value action.

Having selected the value for γ, we initialise the weights for each action to be equal (line 2), since each action is equally promising in the beginning. The algorithm maintains one weight parameter for each action, which summarises the past observations and drives the sampling behaviour. At each round, these weights are used to compute the probabilities for each action to be selected in that round (line 4).

After observing the reward of the selected action, the value of the reward is scaled by the probability that was assigned to that action (line 7). This means that actions that had a high probability of being selected, e.g. because they have historically yielded high rewards, change our beliefs about their value less than actions, whose rewards run counter to our belief (i.e. we expected a low reward but received a high reward). The weights are then updated according to an exponential weighting, giving the algorithm its name (line 8).

EXP3.M

EXP3 was extended by Uchiya et al. (2010) to allow for multiple actions to be chosen at each iteration and is called EXP3.M. The algorithm is outlined in Algorithm 6.

The main ideas remain the same in both EXP3 and EXP3.M. The weight parameters encode our belief of how good we believe the different actions to be. The γ tradeoff parameter allows us to control how much effect the observations have on the selections. Having observed the reward of performing an action, we scale the reward by the probability of choosing that action and update the weight parameter through an exponential re-weighting.

There are two main differences in EXP3.M compared to the original EXP3 algorithm, stemming from the multiple plays setting. First, the weight parameters w_i cannot exceed $\left(\frac{1}{k} - \frac{\gamma}{K}\right) / (1-\gamma) \sum_{j=1}^{K} w_j$, where k is the batch size and K is the total number of possible actions. There is an additional step in the algorithm, that thresholds the weight parameters in case they do exceed this limit, as seen in lines 4-10. The second difference lies in the fact that the algorithm selects a batch of distinct actions each round. Using the `DepRound` (Gandhi et al., 2006) algorithm (line 12) ensures that each action is chosen efficiently and according to its assigned probability, but any reasonable alternative approach can be substituted instead if so desired.

2.5 Summary

In this chapter we introduced the main concepts that are relevant to this thesis. We discussed Bayesian probability theory and model selection and introduced the Gaussian process, which is a Bayesian modelling approach well-suited to the problem domain discussed in this thesis. We discussed posterior inference and kernel choices, as well as model selection in the GP context. Our discussion on Bayesian optimisation focussed on its two main building blocks, the surrogate model, in our case a GP, and the acquisition function, of which we discussed a number of popular variants. We also highlighted research directions and open questions in the BO space that we believe can widen its relevance and extend its reach to more application

domains. Finally, we introduced the multi-armed bandit, a class of algorithms that also considers the exploration-exploitation tradeoff, but applies it to discrete action spaces. We discussed popular approaches for i.i.d. as well as adversarial rewards.

Algorithm 6 EXP3.M

Input: $\gamma \in [0,1]$, number of choices per iteration k.
Initialise the weights $w_1(1) = w_2(1) = \ldots = w_K(1) = 1$.
for $t = 1, \ldots, T$ **do**
  if $\arg\max_{j \in \{1,\ldots,K\}} w_j(t) \geq \left(\frac{1}{k} - \frac{\gamma}{K}\right) \sum_{i=1}^{K} w_i(t)/(1-\gamma)$ **then**
    Find the value of $\alpha(t)$ such that

$$\frac{\alpha_t}{\sum_{w_i(t) \geq \alpha_t} \alpha_t + \sum_{w_i(t) < \alpha_t} w_i(t)} = \left(\frac{1}{k} - \frac{\gamma}{K}\right)/(1-\gamma).$$

    Set $S_0(t) = \{i : w_i(t) \geq \alpha_t\}$ and $w_i'(t) = \alpha_t$ for $i \in S_0(t)$.
  else
    Set $S_0(t) = \emptyset$.
  end if
  Set

$$w_i'(t) = w_i(t) \text{ for } i \in \{1, 2, \ldots, K\} - S_0(t).$$

  Define the probabilities for each action as

$$p_i(t) = k\left((1-\gamma)\frac{w_i'(t)}{\sum_{j=1}^{K} w_j'(t)} + \frac{\gamma}{K}\right) \text{ for } i = 1, 2, \ldots, K.$$

  Sample the set of k actions $S(t)$ from the discrete probability distribution

$$S(t) = \texttt{DepRound}\left(k, (p_1(t), \ldots, p_K(t))\right).$$

  Observe rewards $r_i(t)$ for $i \in S(t)$.
  Define the estimated rewards for $i \in \{1, \ldots, K\}$

$$\hat{r}_i(t) = \begin{cases} r_i(t)/p_i(t) & \text{if } i \in S(t) \\ 0 & \text{otherwise} \end{cases}$$

  Update the actions' weights for $i \in \{1, \ldots, K\}$

$$w_i(t+1) = \begin{cases} w_i(t) \exp\left(k\gamma \hat{r}_i(t)/K\right) & \text{if } i \notin S_0(t) \\ w_i(t) & \text{otherwise} \end{cases}$$

end for

3

Noisy-input Gaussian processes

Contents

Applications of BO usually consider the optimisation of a black-box function $f(\mathbf{x})$, where $\mathbf{x}$ is a quantity we have full control over, e.g. hyperparameters of a model. This is reflected in the fact that both the acquisition function $\alpha(\mathbf{x})$ as well as the surrogate model treat $\mathbf{x}$ as exact (i.e. its value is known perfectly).

Cases where this assumption does not hold include situations where the inputs, $\mathbf{x}$, are spatial coordinates, and the task involves acquiring measurements on a spatial grid. Relevant applications include measuring concentrations of air pollutants (CO_2,

NO_2 etc.) and terrestrial or marine wildlife studies. When measurements are acquired using drones or sensors attached to road vehicles, then triangulation (e.g. with mobile phone towers) or GPS (Kumar and Moore, 2002) are obvious choices for acquiring the position of the measuring device. The positional readings $\mathbf{x}$, though, will actually be noisy estimates of the (unknown) true position $\hat{\mathbf{x}}$.

Following the Bayesian modelling philosophy, we want to include as much knowledge as possible about the problem into the algorithm in order to develop a high-quality, data-efficient model. In this case we consider the fact that the values of the inputs $\mathbf{x}$ are uncertain or noisy. We suggest that a natural approach for BO would be to use a surrogate model that models input uncertainty. There are a number of adaptations of GPs that allow it to incorporate input uncertainty, but there exists no head-to-head analysis comparing the different methods as uncertain-input regression models, as far as we are aware. Research on uncertain-input GP models has instead focused on autoregressive modelling, where GP predictions for previous time are used as inputs of to predict a trajectory of future values (Girard et al., 2003; McHutchon and Rasmussen, 2011).

It is helpful to differentiate between two types of uncertainty, aleatoric and epistemic uncertainty. Aleatoric uncertainty, also known as statistical uncertainty, is caused by unknown processes that differ each time an experiment is conducted, and hence cannot be reduced by repeated measurements. Epistemic uncertainty, or systematic uncertainty, is the type of uncertainty we are investigating in this chapter. Repeated measurements of a property, e.g. positioning via GPS, can provide a more accurate understanding, i.e. we can achieve lower variance in our predictions than the variance in the underlying noise process.

In this chapter we conduct a comparison of a number of existing uncertain-input GP models that have been proposed thus far, and also propose a new uncertain-input GP method based on the unscented transform (UT), and analyse their performance as *regression methods* for data with uncertain inputs. We aim to provide a better understanding of the state-of-the-art approaches, so that a BO practitioner can make a more informed choice regarding which uncertain-input GP

surrogate model to choose for their application. Our investigation showed that our proposed method utilising the UT showed the best predictive performance, with the standard GP coming in at a close second.

3.1 Problem definition

When using Gaussian processes for regression, we generally consider the situation where we are given a data set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$, where $\mathbf{x}_i \in \mathbb{R}^d$ are d-dimensional inputs and $y_i \in \mathbb{R}$ are realisations of a latent function $f : \mathbb{R}^d \to \mathbb{R}$, corrupted by additive noise (Rasmussen and Williams, 2006):

$$y_i = f(\mathbf{x}_i) + \epsilon_y. \tag{3.1}$$

If the noise is assumed Gaussian, i.e. $\epsilon_y \sim \mathcal{N}(0, \sigma_y)$, then that leads to a closed-form solution for the posterior (see Section 2.2).

In the applications we mentioned above, assuming noise existing solely on the outputs does not fully capture the underlying truth. The general uncertain-input uncertain-output situation with varying input noise is:

$$y_i = f(\mathbf{x}_i + \boldsymbol{\epsilon}_i) + \epsilon_y, \tag{3.2}$$

where the use of a separate noise term, $\boldsymbol{\epsilon}_i$, for each input location emphasises that each input location may be corrupted by different amounts of noise.

A simple approach to this problem could be the following. Given a data set $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ and assuming the noise distribution can be parameterised by θ_i, i.e. $\boldsymbol{\epsilon}_i \sim g(\theta_i)$ for some distribution g, a simple approach could be to add these parameters as additional features such that $\tilde{\mathbf{x}} = [\mathbf{x}; \theta_i]$ (Almosallam, 2017). In other words, we rely on our GP model to learn the effects of the noise parameters θ_i. In order for this approach to provide useful results, we need a lot of data, as well as varying input noise in the training data: a global noise model would extend every input vector with the same constant values: $\theta_i = \theta_x \forall i$, so $\tilde{\mathbf{x}} = [\mathbf{x}; \theta_x]$, rendering the noise parameters uninformative. This idea was explored in Almosallam (2017) and was shown to not work well for these reasons.

Approaching the uncertain-input problem from a more Bayesian standpoint, and given we know the input noise distribution $\mathbf{x} \sim p(\mathbf{x} \mid \hat{\mathbf{x}}, \theta_x)$, where $\hat{\mathbf{x}}$ is the true (unobserved) value of the input $\mathbf{x}$ and θ_x are the global input noise distribution's parameters, we should marginalise out the input uncertainty from our GP posterior:

$$p(f|\{\hat{\mathbf{x}}_i, y_i\}_{i=1}^N) = \int p(f|\{\mathbf{x}_i, y_i\}_{i=1}^N) \prod_{i=1}^N p(\mathbf{x}_i|\hat{\mathbf{x}}_i) \mathrm{d}\mathbf{x}_i, \tag{3.3}$$

where we have dropped the conditioning on θ_x and the kernel hyperparameters for clarity. This is similar to how we combine the observation noise $p(y|f)$ with the GP posterior $p(f|\mathcal{D}, \theta)$ to explicitly represent the uncertainty in our data. The difference here is the fact that the integral in Equation (3.3) is intractable, since the posterior requires inverting the covariance matrix, e.g. to compute the posterior mean (Rasmussen and Williams, 2006):

$$\boldsymbol{\mu}(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_y^2 I\right]^{-1} \mathbf{y}. \tag{3.4}$$

The models we will consider approximate the intractable integral in Equation (3.3) in different ways. We will focus on the case of a global noise process on the inputs, so all noise terms $\boldsymbol{\epsilon}_i$ follow the same distribution. And in particular we will consider the case where the noise process is a zero-mean Gaussian with known covariance:

$$\boldsymbol{\epsilon}_i \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_{\boldsymbol{x}}). \tag{3.5}$$

This is a practical assumption, since this models the situation where we know the accuracy of the positioning technology being used (triangulation, GPS, etc.). We will compare a standard GP with three existing popular uncertain-input methods that each extend GPs in different ways, and a novel method based on the UT.

The standard GP will be the baseline method and we will use the popular RBF as well as the rational quadratic (RQ) kernels:

$$r(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T \Sigma_k^{-1} (\mathbf{x}_i - \mathbf{x}_j), \tag{3.6}$$

$$k_{\mathrm{RBF}}(r) = \sigma^2 \exp(-\frac{1}{2} r^2), \tag{3.7}$$

$$k_{\mathrm{RQ}}(r) = \sigma^2 \left(1 + \frac{r^2}{2\alpha}\right)^{-\alpha}, \tag{3.8}$$

where Σ_k is the $d \times d$ diagonal matrix with the lengthscales for each input dimension, σ^2 is the kernel variance and α is the power parameter in the RQ kernel.

Although the RBF kernel is a popular choice for GP regression, we also include the rational quadratic kernel for comparison, because it represents a mixture of RBF kernels with different lengthscales (Rasmussen and Williams, 2006), which may provide the GP with more flexibility to adapt to the uncertain-input setting. We will now introduce the uncertain-input models.

3.2 Analytic model

One way to approximate the integral in Equation (3.3) is to integrate the Gaussian input noise distribution into a RBF kernel (Dallaire et al., 2011):

$$k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_j) = \iint k_{\mathrm{RBF}}\left(\mathbf{x}_i, \mathbf{x}_j\right) p\left(\mathbf{x}_i\right) p\left(\mathbf{x}_j\right) \mathrm{d}\mathbf{x}_i \mathrm{d}\mathbf{x}_j. \tag{3.9}$$

Using a RBF kernel results in an analytic result by making use of properties of products of Gaussians, which several authors have used. One particularly clear derivation of this model is that of Dallaire et al. (2011), whose procedure we use to guide the derivation below. We will refer to this model as the *Analytic* model.

We begin by deriving the uncertain-input kernel evaluated for two input locations $\mathbf{x}_i$ and $\mathbf{x}_j$. Start with the RBF kernel

$$k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma_k^2 \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mathbf{x}_j)^T \Sigma_k^{-1} (\mathbf{x}_i - \mathbf{x}_j)\right). \tag{3.10}$$

This is an unnormalised Gaussian distribution, so we can rewrite it as

$$k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_j) = c\, \mathcal{N}\left(\mathbf{x}_i; \mathbf{x}_j, \Sigma_k\right), \tag{3.11}$$

where

$$c = \sigma_k^2 (2\pi)^{D/2} |\Sigma_k|^{1/2}. \tag{3.12}$$

Writing the distributions that represent the noise over our inputs as

$$p(\mathbf{x}_i) = \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right), \tag{3.13}$$

where $\hat{\mathbf{x}}_i$ is the (unknown) true location and $\mathbf{x}_i$ is the observed input location, allows us to write the marginalisation in Equation (3.9) as

$$k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_j) = \iint c\, \mathcal{N}\left(\mathbf{x}_i; \mathbf{x}_j, \Sigma_k\right) \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right) \mathcal{N}\left(\mathbf{x}_j; \hat{\mathbf{x}}_j, \Sigma_j\right) \mathrm{d}\mathbf{x}_i \mathrm{d}\mathbf{x}_j. \tag{3.14}$$

Equation (3.14) consists of convolutions of Gaussian distributions, once w.r.t. each input $\mathbf{x}_i$ and $\mathbf{x}_j$, which we can perform sequentially:

$$\begin{aligned} k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_j) &= c \int \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right) \left[\int \mathcal{N}\left(\mathbf{x}_i; \mathbf{x}_j, \Sigma_k\right) \mathcal{N}\left(\mathbf{x}_j; \hat{\mathbf{x}}_j, \Sigma_j\right) \mathrm{d}\mathbf{x}_j \right] \mathrm{d}\mathbf{x}_i && (3.15) \\ &= c \int \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right) \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_j, \Sigma_k + \Sigma_j\right) \mathrm{d}\mathbf{x}_i && (3.16) \\ &= c\, \mathcal{N}\left(\hat{\mathbf{x}}_i; \hat{\mathbf{x}}_j, \Sigma_k + \Sigma_i + \Sigma_j\right). && (3.17) \end{aligned}$$

Writing out the kernel gives us

$$\begin{aligned} k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_j) &= \sigma_k^2 (2\pi)^{D/2} |\Sigma_k|^{1/2} \mathcal{N}\left(\hat{\mathbf{x}}_i; \hat{\mathbf{x}}_j, \Sigma_k + \Sigma_i + \Sigma_j\right) && (3.18) \\ &= \frac{|\Sigma_k|^{1/2}}{|\Sigma_k + \Sigma_i + \Sigma_j|^{1/2}} \sigma_k^2 \exp\left(-\frac{1}{2} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j)^T (\Sigma_k + \Sigma_i + \Sigma_j)^{-1} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j) \right) && (3.19) \\ &= \frac{\sigma_k^2 \exp\left(-\frac{1}{2} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j)^T (\Sigma_k + \Sigma_i + \Sigma_j)^{-1} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j) \right)}{|I + \Sigma_k^{-1} (\Sigma_i + \Sigma_j)|^{1/2}}, && (3.20) \end{aligned}$$

where the original RBF kernel parameters are (σ_k, Σ_k). The variance at a location $\mathbf{x}_i$ can be computed as:

$$\begin{aligned} k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_i) &= \int k_{\mathrm{RBF}}(\mathbf{x}_i, \mathbf{x}_i) \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right) \mathrm{d}\mathbf{x}_i && (3.21) \\ &= \sigma_k^2 \int \mathcal{N}\left(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_i\right) \mathrm{d}\mathbf{x}_i && (3.22) \\ &= \sigma_k^2. && (3.23) \end{aligned}$$

We can combine the results in Equations (3.20) and (3.23) by using the Kronecker delta δ_{ij} ($\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ otherwise):

$$k_{\mathrm{NI}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{\sigma_k^2 \exp\left(-\frac{1}{2} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j)^T (\Sigma_k + \Sigma_i + \Sigma_j)^{-1} (\hat{\mathbf{x}}_i - \hat{\mathbf{x}}_j) \right)}{|I + \Sigma_k^{-1} (\Sigma_i + \Sigma_j)(1 - \delta_{ij})|^{1/2}}. \tag{3.24}$$

This ensures that the diagonal of the covariance matrix is only scaled by the variance at the input locations, without affecting the covariance in the off-diagonal

elements of the kernel matrix. When $i \neq j$ we get Equation (3.20), and when $i = j$ we recover Equation (3.23).

This analytic kernel is a modified RBF kernel, where the lengthscale is adapted by the variance of the noise on the inputs. High input noise will have a large noise covariance, which will reduce the value of k_{NI}, and the GP's predictions will rely more on the prior mean. Higher input noise variance will also have the effect of reducing the output variance, again reducing the similarity between points, and hence the effect a training point has on the posterior. As a sanity-check, we should expect any kernel that models input noise to reduce to the standard noise-less kernel when we set the covariances Σ_i and Σ_j to zero. Substituting $\Sigma_i = \Sigma_j = \mathbf{0}$, the denominator reduces to one and the sum of covariances $\Sigma_k + \Sigma_i + \Sigma_j$ reduces to just the lengthscale parameter Σ_k, resulting in the standard RBF kernel.

This modelling approach for input noise restricts us to using the RBF kernel, which may not always be appropriate if we are modelling real-world phenomena, where f is unlikely to be infinitely-differentiable. Though we are focussing on a global noise process in our work, we note that this approach could also allow us to incorporate heteroscedastic input noise (spatially-varying noise), if desired. This could be relevant in cases where measurements are collected from different measurement techniques, e.g. creating a combined data set of measurements with triangulated position data as well as measurements with finer-grained GPS positions. In this case, the values of Σ_i and Σ_j would be observed along with the input location, and likely vary across the data.

Similarly, if we are modelling data with a single, global noise process, i.e. $\Sigma_i = \Sigma_j = \Sigma_x$, then this simplifies the kernel's expression. If the noise process is known, then we can provide the correct noise variance values to the calculation, but alternatively we can also treat the noise parameter as a hyperparameter and learn it together with the other hyperparameters of the kernel.

It is important to note the differences between this model and the original task in Equation (3.3). We are in effect defining a kernel between Gaussian distributions

that are centred around the samples, which is a simplification of the true problem of integrating a global noise process on the input locations.

3.3 RandFunc model

We now introduce an uncertain-input model developed in Girard and Murray-Smith (2003). Due to its derivation relying on random function theory, we will refer to it as the *RandFunc* model. The authors begin with the same assumptions as in Equation (3.2):

$$y_i = f(\mathbf{x}_i) + \epsilon_y, \tag{3.25}$$

$$\mathbf{x}_i = \hat{\mathbf{x}}_i + \boldsymbol{\epsilon}_x, \tag{3.26}$$

but make the added assumption that the noise distribution is isotropic: $\boldsymbol{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, v_x\mathbf{I})$. They state that the expected output value at a random location $\mathbf{x}$ is given by

$$\mathbb{E}[y \mid \hat{\mathbf{x}}, v_x] = \int f(\mathbf{x})p(\mathbf{x})\mathrm{d}\mathbf{x}, \tag{3.27}$$

where $\hat{\mathbf{x}}$ is the true value of the noise-corrupted location $\mathbf{x}$. The authors approach this problem by approximating $f(\mathbf{x})$ using a second-order Taylor expansion:

$$f(\mathbf{x}) \approx f(\hat{\mathbf{x}}) + (\mathbf{x} - \hat{\mathbf{x}})^T f'(\hat{\mathbf{x}}) + \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T f''(\hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}}), \tag{3.28}$$

where $f'(\hat{\mathbf{x}}) = \frac{\partial f(\hat{\mathbf{x}})}{\partial \mathbf{x}}$ and similarly for $f''(\hat{\mathbf{x}})$. The authors then substitute Equation (3.28) into Equation (3.27), which gives us

$$\mathbb{E}[y \mid \hat{\mathbf{x}}, v_x] \approx \int \left(f(\hat{\mathbf{x}}) + (\mathbf{x} - \hat{\mathbf{x}})^T f'(\hat{\mathbf{x}}) + \frac{1}{2}(\mathbf{x} - \hat{\mathbf{x}})^T f''(\hat{\mathbf{x}})(\mathbf{x} - \hat{\mathbf{x}}) \right) p(\mathbf{x})\mathrm{d}\mathbf{x} \tag{3.29}$$

$$= f(\hat{\mathbf{x}}) + \frac{v_x}{2}\mathrm{Tr}[f''(\hat{\mathbf{x}})] = g(\hat{\mathbf{x}}, v_x). \tag{3.30}$$

If we use a GP with kernel function $k(.)$ to model the function $f(\hat{\mathbf{x}})$, then the quantity $g(\hat{\mathbf{x}}, v_x)$ can be seen as the sum of two correlated random functions. The authors suggest using results from random function theory (Pugachev, 1967) to

find an expression for the covariance function for $g(\hat{\mathbf{x}}, v_x)$. This allows us to write the covariance as:

$$\begin{aligned}\operatorname{cov}[g(\hat{\mathbf{x}}_i, v_x), g(\hat{\mathbf{x}}_j, v_x)] = k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j) + \frac{v_x^2}{4}\operatorname{Tr}\left[\frac{\partial^2}{\partial\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T}\left[\frac{\partial^2 k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)}{\partial\hat{\mathbf{x}}_j\hat{\mathbf{x}}_j^T}\right]\right] \\ + v_x\operatorname{Tr}\left[\frac{\partial^2 k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)}{\partial\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T}\right]. \qquad (3.31)\end{aligned}$$

Note that the kernel is assumed to be four times differentiable. In the second term, $\left[\frac{\partial^2}{\partial\hat{\mathbf{x}}_i\hat{\mathbf{x}}_i^T}\left[\frac{\partial^2 k(\hat{\mathbf{x}}_i,\hat{\mathbf{x}}_j)}{\partial\hat{\mathbf{x}}_j\hat{\mathbf{x}}_j^T}\right]\right]$ is the result of a $d \times d$ matrix differentiated w.r.t. another $d \times d$ matrix, which results in a matrix of size $d^2 \times d^2$, where d is the dimensionality of $\mathbf{x}$. Since we only need the trace of this quantity, we only need to compute the d^2 diagonal terms of the matrix. The third term with the second-order derivative similarly only requires d computations instead of d^2.

Letting $v_x \to 0$, we can attest to the fact that the model does indeed revert back to a standard GP model, since the kernel in Equation (3.31) reduces to the simple zero-input-noise kernel $k(\hat{\mathbf{x}}_i, \hat{\mathbf{x}}_j)$ in this case.

3.4 NIGP

The Noisy-Input Gaussian Process (NIGP) was developed in McHutchon and Rasmussen (2011) and also takes a Taylor expansion approach to approximating the effect of uncertain inputs. They began by considering the Taylor expansion about the observed state $\mathbf{x}$,

$$f(\mathbf{x} + \epsilon_x) = f(\mathbf{x}) + \boldsymbol{\epsilon}_x^T\boldsymbol{\partial}_f + \cdots, \qquad (3.32)$$

where we follow the authors' nomenclature $\boldsymbol{\partial}_f = \frac{\partial f(\mathbf{x})}{\partial\mathbf{x}}$. Rather than trying to exactly model the Taylor series expansion's terms (as the derivative of a GP is another GP), the authors advocated considering only up to first order terms. This was motivated by the difficulty in defining a tractable distribution over all of the Taylor expansion terms – one would have to find a distribution that represents derivatives of ever increasing orders.

Instead, the result of truncating the Taylor expansion after the first order term is a linear model in the noise:

$$y \approx f(\mathbf{x}) + \boldsymbol{\epsilon}_x^T \boldsymbol{\partial}_f + \epsilon_y. \tag{3.33}$$

This changes the likelihood of the GP to include a term due to the noise on the inputs:

$$p(y \mid f) = \mathcal{N}(f, \sigma_y^2 + \boldsymbol{\partial}_f^T \Sigma_x \boldsymbol{\partial}_f), \tag{3.34}$$

which allows us to write the posterior mean and variance equations for the NIGP as:

$$\mu(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_y^2 I + \text{diag}\{\boldsymbol{\Delta}_f \Sigma_x \boldsymbol{\Delta}_f^T\}\right]^{-1} \mathbf{y}, \tag{3.35}$$

$$\mathbf{V}(\mathbf{x}_*) = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X}) \left[k(\mathbf{X}, \mathbf{X}) + \sigma_y^2 I + \text{diag}\{\boldsymbol{\Delta}_f \Sigma_x \boldsymbol{\Delta}_f^T\}\right]^{-1} k(\mathbf{X}, \mathbf{x}_*), \tag{3.36}$$

where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ are the training points, diag{.} creates a diagonal matrix with diagonal elements equal to the diagonal of its argument, and $\boldsymbol{\Delta}_f$ is a $N \times d$ matrix of first derivatives of f evaluated at each of the training points. McHutchon and Rasmussen (2011) use the derivative of the posterior mean for $\boldsymbol{\Delta}_f$.

Looking at the posterior mean and variance expressions gives us an intuition of how the NIGP models input uncertainty. At locations where the gradient of f is zero, the model predictions are equal to a standard GP, but as the gradient increases, the contribution of the uncertain-input term increases. This behaviour makes sense if you consider that input noise would have little effect on the value of the output if the function were mostly flat in the region being considered – changes in $\mathbf{x}$ would not change our belief about y by a lot. If the function was changing rapidly, corresponding to larger values for $\boldsymbol{\partial}_f$, then errors in $\mathbf{x}$ would lead to greater inaccuracies in the predictions, which is reflected in an inflated posterior variance in such regions. Similarly, the posterior mean reduces the contribution of the data and moves closer to the prior mean in high-gradient regions, reflecting our reduced trust in the data.

Also note that the model satisfies our requirement that it revert to a noiseless GP for $\Sigma_x = \mathbf{0}$.

One important consideration is the fact that the NIGP inference contains a recursive relationship between the posterior and its derivative. In order to evaluate the posterior, we need the derivative of its mean, which in turn requires the value of the posterior in the first place. In our implementation, we initialise the model without the input noise term and find its gradient. Then we update the posterior followed by the gradient of the mean a set number of times until we are satisfied that the value will have converged. We chose to repeat the loop 20 times after evaluating the convergence for different initialisations. This was a trade-off between execution time and the fact that the model seemed to stabilise most of the time, i.e. the posterior changed minimally across iterations. For a more detailed discussion and some examples of different behaviours we observed see Section 3.7.

3.5 Unscented transform GP (UTGP)

In this section we develop a novel extension of GPs to modelling uncertain inputs by using the unscented transform (UT) (Julier and Uhlmann, 1997a, 2004). The UT was developed as an alternative method for adapting the popular Kalman filter (KF) (Kalman, 1960) to model nonlinearities, called the unscented Kalman filter (UKF). Before the UT, the extended Kalman filter (EKF) (Jazwinski, 1970) was the most popular adaptation of the KF for such tasks. The shortcoming of the EKF is that it uses first-order information to linearise the state and observation models, leading to predictions that can be inaccurate and overconfident in regions of highly nonlinear behaviour. The benefit of using the UT is that no such linearisation is required. First-order linearisations are also used in the RandFunc and NIGP methods discussed earlier, so developing a UT-based approach for GPs is a natural choice. Below, we will first provide an overview of the UT and then show how we use it to extend GPs to perform inference with uncertain inputs.

3.5.1 Unscented transform

Let $\mathbf{a}$ be a vector-valued random variable with mean $\hat{\mathbf{a}}$ and covariance Σ_a. Define a second random variable, $\mathbf{b}$, that is related to $\mathbf{a}$ via a nonlinear function $\mathbf{b} = \mathbf{g}(\mathbf{a})$

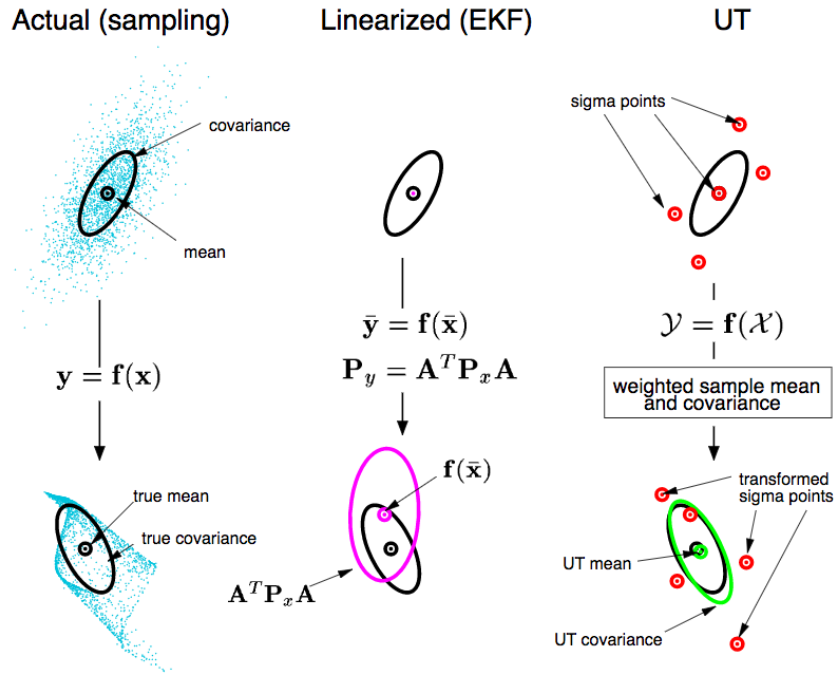


Figure 3.1: Unscented transform in action: EKF vs. UKF (Wan and Van Der Merwe, 2000, Figure 1). The left example shows a Monte Carlo estimate of the transformation through $f(\mathbf{x})$, resulting in highly non-Gaussian surface. The first and second moment of this surface are shown as the black ellipse ("true mean" and "true covariance") and are the target quantities to be estimated. The EKF and UKF are shown in the center and right respectively, as well as the resulting estimates for the first two moments. We can see that the mean and covariance of the UKF are closer to the true values than the EKF.

and we wish to find its mean $\hat{\mathbf{b}}$ and covariance Σ_b. In this case an initial approach to finding the distribution $p(\mathbf{b})$ could be to sample a large number of points from $p(\mathbf{a})$, transform them by computing $g(\mathbf{a})$ and then computing the moments of the transformed samples. While such approaches often are guaranteed to converge to the correct values, they often require a large number of samples to give a low-variance estimate (Bishop, 2006). The UT is a procedure that defines a set of rules for *deterministically* selecting a set of points, called "sigma points", that allow us to calculate the moments of $p(\mathbf{b})$ (Julier and Uhlmann, 1997a).

The UT is defined as follows: Given a nonlinear function $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^{d'}$ and a d-dimensional random variable $\mathbf{a} \sim p(\mathbf{a})$, where the first and second moments of $p(\mathbf{a})$ are $\hat{\mathbf{a}}$ and Σ_a respectively, we can define $(2d+1)$ sigma points $\{\mathbf{a}^{(0)}, \mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(2d)}\}$ and weight parameters $\{W^{(0)}, W^{(1)}, \ldots, W^{(2d)}\}$ where $W^{(i)} \in \mathbb{R}$ and $\sum_i W^{(i)} = 1$.

We can compute the mean and covariance of the distribution $\mathbf{b} \sim p(\mathbf{b})$, where $\mathbf{b} = \mathbf{g}(\mathbf{a})$, $\mathbf{b} \in \mathbb{R}^{d'}$, by applying the nonlinearity $\mathbf{g}$ to the sigma points and computing their sample mean and covariance:

$$\mathbf{b}^{(i)} = \mathbf{g}(\mathbf{a}^{(i)}), \tag{3.37}$$

$$\hat{\mathbf{b}} = \sum_{i=0}^{2d} W^{(i)} \mathbf{b}^{(i)}, \tag{3.38}$$

$$\Sigma_b = \sum_{i=0}^{2d} W^{(i)} (\mathbf{b}^{(i)} - \hat{\mathbf{b}})(\mathbf{b}^{(i)} - \hat{\mathbf{b}})^T. \tag{3.39}$$

The sigma points are selected according to the following scheme:

$$\mathbf{a}^{(0)} = \hat{\mathbf{a}}, \tag{3.40}$$

$$\mathbf{a}^{(i)} = \hat{\mathbf{a}} + \left(\sqrt{(d+k)\Sigma_a}\right)_i \qquad \forall i = 1 \ldots d, \tag{3.41}$$

$$\mathbf{a}^{(i)} = \hat{\mathbf{a}} - \left(\sqrt{(d+k)\Sigma_a}\right)_{i-d} \qquad \forall i = (d+1) \ldots 2d, \tag{3.42}$$

where k is a free parameter (Julier and Uhlmann (1997a) suggest using the heuristic $k + d = 3$), and $\left(\sqrt{\cdot}\right)_i$ denotes the i-th column of the matrix square root of the argument. We use the Cholesky decomposition (Stachniss, 2006; Press et al., 2007), but any matrix square root can be used in its place. For the weights, Julier and Uhlmann (1997a) suggest

$$W^{(0)} = \frac{k}{d+k}, \tag{3.43}$$

$$W^{(i)} = \frac{k}{2(d+k)}, \tag{3.44}$$

where $i \in \{1, \ldots, 2d\}$. This version of the UT is called the simplex formulation. Other adaptations of the UT exist that can capture more information, e.g. the first three moments (Julier, 1998), or the first four nonzero moments of a Gaussian (Julier and Uhlmann, 1997b). See Figure 3.1, which is taken from Wan and Van Der Merwe (2000), for an illustration of Kalman filters utilising linearisation (EKF) and the UT (UKF) for modelling nonlinearities compared to a sampled ground truth. The first-order linearisation is unable to capture the nonlinearities accurately, so the transformed distribution's mean and covariance differ significantly from the ground truth. The UKF on the other hand is able to model the effect

of the nonlinear transform better, and its calculated mean and covariance are both closer to the ground truth.

3.5.2 Scaled unscented transform

The simplex UT exhibits an unwanted behaviour as the dimension of $\mathbf{a}$ increases. We can see in Equations (3.41) and (3.42) that the distance between sigma points increases due to the additive term scaling with $\sqrt{d}$. So with increasing d, the UT increasingly models global variations at the expense of local features. Note also that the weights are not restricted to being positive, and indeed with $k < 0$ they are likely to have negative values, which can lead to numerical problems (Julier, 2002).

To address this shortcoming, Julier (2002) proposed a generalisation of the UT, the *scaled* UT, which allows us to control the spread of the sigma points while still capturing the desired moments of the transformed distribution $p(\mathbf{b})$.

Starting with the definitions of the sigma points and weights of the simplex UT, the scaled UT can be written as (Julier, 2002):

$$\mathbf{a}_s^{(i)} = \mathbf{a}^{(0)} + \alpha(\mathbf{a}^{(i)} - \mathbf{a}^{(0)}), \tag{3.45}$$

$$\mathbf{b}_s^{(i)} = \mathbf{g}(\mathbf{a}_s^{(i)}), \tag{3.46}$$

$$W_s^{(0)} = W^{(0)}/\alpha^2 + (1 - 1/\alpha^2), \tag{3.47}$$

$$W_s^{(i)} = W^{(i)}/\alpha^2 \qquad i = 1, 2, \ldots, 2d, \tag{3.48}$$

$$\hat{\mathbf{b}} = \sum_{i=0}^{2d} W_s^{(i)} \mathbf{b}_s^{(i)}, \tag{3.49}$$

$$\begin{aligned}\Sigma_b = &\sum_{i=0}^{2d} W_s^{(i)} (\mathbf{b}_s^{(i)} - \hat{\mathbf{b}})(\mathbf{b}_s^{(i)} - \hat{\mathbf{b}})^T \\ &+ (W_s^{(0)} + 1 - \alpha^2 + \beta)(\mathbf{b}_s^{(0)} - \hat{\mathbf{b}})(\mathbf{b}_s^{(0)} - \hat{\mathbf{b}})^T,\end{aligned} \tag{3.50}$$

where $0 < \alpha \leq 1$ is the scaling parameter and β is a parameter that allows higher-order effects to be incorporated, and usually $\beta = 2$. Additionally, Stachniss (2006) suggest the following scheme for choosing the k parameter:

$$k = \alpha^2(d + \kappa) - d, \tag{3.51}$$

where the new parameter $\kappa \geq 0$ gives added control over the spread of the sigma points. We use the recommended values for the UT parameters $\alpha = 0.001$, $\beta = 2$ and $\kappa = 0$ (Fisher, 2009).

The scaled UT is similar in computational complexity as the simplex UT, and it addresses the numerical and modeling weaknesses of the simplex UT directly. This makes it a good choice to use for extending GPs to modelling input noise.

3.5.3 The unscented transform GP (UTGP)

The RandFunc (Section 3.3) and NIGP (Section 3.4) models both are derived from first-order linearisation via Taylor expansion, which is also the method utilised in the EKF. The motivation for investigating the UT for GP regression for uncertain inputs comes from the fact that the UT was developed to avoid such first-order approximations and the UKF performs well as an alternative to the EKF .

Our modeling task is the same as in the uncertain-input GP models above. We re-state the uncertain-input regression objective from above to facilitate the discussion below:

$$p(f|\hat{\mathbf{X}}, \mathbf{y}) = \int p(f|\mathbf{X}, \mathbf{y}) \prod_{i=1}^{N} p(\mathbf{x}_i|\hat{\mathbf{x}}_i) \mathrm{d}\mathbf{x}_i, \tag{3.52}$$

where we have collected the observed values as $\mathbf{y} = \{y_i\}_{i=1}^N$, the observed locations as $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^N$, and the unknown (true or noiseless) locations as $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_i\}_{i=1}^N$ for brevity.

Drawing the links with the UT, our input distribution is the noise distribution over input locations $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \hat{\mathbf{x}}, \Sigma_x)$ and our nonlinearity is the GP. Here the first difference emerges, since the nonlinear transformation ($\mathbf{g}(.)$ in the section above) is a deterministic function, and here the nonlinearity is the GP posterior. We will focus on the case where we wish to model input uncertainty where the input noise process is known. This assumption relates to the setting where the data is acquired using a single measurement process (GPS, triangulation etc.), whose precision is known a priori. This assumption will also simplify our model development in

the following section. Learning the input noise directly from the data, e.g. as an additional hyperparameter, could constitute an interesting future research direction.

In the past, the UT has been used in two settings related to GPs: in Nogueira et al. (2016) the authors applied the UT to the acquisition function in Bayesian Optimisation and in Steinberg and Bonilla (2014) the UT is used for GP inference with non-Gaussian likelihoods. An analysis of the UT applied directly to GPs for regression has not been undertaken in the literature as far as we are aware.

We begin by assuming that the noise process in Equation (3.52) is Gaussian with known parameters, i.e. $p(\mathbf{x}_i) = \mathcal{N}(\mathbf{x}_i; \hat{\mathbf{x}}_i, \Sigma_x)$ for each input location $\mathbf{x}_i$ in the training set. On a high level, the UT approaches this integral by replacing the probability distributions $\prod_{i=1}^{N} p(\mathbf{x}_i|\hat{\mathbf{x}}_i)$ with delta functions at the sigma points' locations, turning the continuous integral into a (weighted) sum of evaluations of $p(f|\{\mathbf{x}_i, y_i\}_{i=1}^{N})$. In our case this turns our posterior, after marginalising out the input noise, into a weighted sum of GPs.

In our approach we propose that our sigma points can be seen as sigma *data sets*, where we define $\mathbf{X}^{(0)} = \mathbf{X}$ according to Equation (3.40) and we apply the sigma transformations in Equations (3.41) and (3.42) to each instance in $\mathbf{X}$ to create $2d$ new data sets $\left\{\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(2d)}\right\}$. We then train a separate GP on each data set $\mathbf{X}^{(i)}$ to create our $2d+1$ sigma GPs $\left\{\mathcal{GP}^{(0)}, \mathcal{GP}^{(1)}, \ldots, \mathcal{GP}^{(2d)}\right\}$. Note that we still use $\dim(\mathbf{x}) = d$ when computing the locations of the sigma points and the weights, and the process of finding the weights $\left\{W^{(0)}, \ldots, W^{(2d)}\right\}$ remains unchanged. Now we can find the posterior mean at a test location $\mathbf{x}_*$, referring to the posterior means of each sigma $\mathcal{GP}^{(i)}$ as $\mu^{(i)}(\mathbf{x}_*)$, by applying the UT formula in Equation (3.38):

$$\boldsymbol{\mu}_{\mathrm{UT}}(\mathbf{x}_*) = \sum_{i=0}^{2d} W_m^{(i)} \boldsymbol{\mu}^{(i)}(\mathbf{x}_*). \tag{3.53}$$

To compute the UTGP posterior variance we need to combine the posterior variance from each sigma GP as well as variance provided by the UT. We chose to combine both of these variances by moment-matching the weighted sum of posterior sigma GP variances and adding it to the UT variance:

$$\boldsymbol{V}_{\mathrm{UTGP}}(\mathbf{x}_*) = \boldsymbol{V}_{mm}(\mathbf{x}_*) + \boldsymbol{V}_{\mathrm{UT}}(\mathbf{x}_*). \tag{3.54}$$

This is a simple approach, that arrives at a simple-to-compute model, but these variances could conceivably be combined in a different way by leveraging ideas from other bodies of research such as signal fusion.

For the moment-matched variance, $\boldsymbol{V}_{mm}$, we make use of a Gaussian identity (as seen in e.g. Osborne (2010)), that states that for a probability distribution that is a mixture of weighted Gaussians

$$p(\mathbf{x} \mid \Theta) = \sum_i \rho_i \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_i, \mathbf{A}_i), \tag{3.55}$$

where the conditioning variables are collected into Θ, the mean and covariance of this mixture respectively are

$$\boldsymbol{m}(\mathbf{x} \mid \Theta) = \int \mathbf{x}' p(\mathbf{x}' \mid \Theta) \mathrm{d}\mathbf{x}' = \sum_i \rho_i \boldsymbol{\mu}_i(\mathbf{x}), \tag{3.56}$$

$$\boldsymbol{C}(\mathbf{x} \mid \Theta) = \int \mathbf{x}'\mathbf{x}'^T p(\mathbf{x}' \mid \Theta) \mathrm{d}\mathbf{x}' - \boldsymbol{m}(\mathbf{x} \mid \Theta)\boldsymbol{m}(\mathbf{x} \mid \Theta)^T \tag{3.57}$$

$$= \sum_i \rho_i (\mathbf{A}_i + \boldsymbol{\mu}_i \boldsymbol{\mu}_i^T) - \boldsymbol{m}(\mathbf{x} \mid \Theta)\boldsymbol{m}(\mathbf{x} \mid \Theta)^T. \tag{3.58}$$

This is analogous to our situation, since at any location $\mathbf{x}_*$ that we are evaluating the posterior at, we have a weighted mixture of the posterior sigma GPs evaluated at this location. Using Equation (3.58) allows us to write our moment-matched posterior variance at a location $\mathbf{x}_*$ as

$$\boldsymbol{V}_{mm}(\mathbf{x}_*) = \sum_{i=0}^{2d} W^{(i)} (\boldsymbol{V}^{(i)}(\mathbf{x}_*) + \boldsymbol{\mu}^{(i)}(\mathbf{x}_*)\boldsymbol{\mu}^{(i)}(\mathbf{x}_*)^T) - \boldsymbol{\mu}_{\mathrm{UT}}\boldsymbol{\mu}_{\mathrm{UT}}^T(\mathbf{x}_*)^2, \tag{3.59}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{V}^{(i)}$ are the posterior mean and variance of $\mathcal{GP}^{(i)}$ respectively. We note that we used the scaled UT in our work, so the UT's contribution to the variance, $\boldsymbol{V}_{\mathrm{UT}}(\mathbf{x}_*)$, is computed as in Equation (3.50).

We approximated the mixture of Gaussians with a single Gaussian distribution as a simple solution in order to facilitate initial empirical tests of the approach. From initial testing, this approach tends to predict higher uncertainties than the other methods in our comparison. Other methods for combining the mixture of Gaussians may be reasonable, so investigating the effectiveness of alternative methods could constitute future research directions.

Training the UTGP

Two approaches are often used in practice to train GPs: optimisation and integration. When optimising the model hyperparameters θ, we maximise the marginal log likelihood of the model (Rasmussen and Williams, 2006):

$$\mathcal{L}(\theta) = -\frac{1}{2}\mathbf{y}^T\mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| - \frac{N}{2}\log 2\pi, \tag{3.60}$$

where $\mathbf{K} = k(\mathbf{X}, \mathbf{X}|\theta)$ is the GP covariance kernel with hyperparameters θ evaluated at all pairs of input locations.

Integration requires marginalising out the hyperparameters, which can be achieved by using slice sampling (Murray and Adams, 2010), which is a Markov Chain Monte Carlo sampling technique that can be used to generate samples from the unnormalised distribution in Equation (3.60). Slice sampling also has the benefit of not needing any helper distributions like Metropolis Hastings or Gibbs sampling. It also doesn't require full conditionals of the joint distribution, which is a requirement for Gibbs sampling.

Each of our sigma GPs provides one such log marginal likelihood $\mathcal{L}^{(i)}$ for its own perturbed data set $\mathbf{X}^{(i)}$. We need to combine the log marginal likelihoods so that the weight parameters of the UT are taken into account, giving us the total marginal likelihood $\mathcal{L}_{\text{UT}}$. This turns out to be a straightforward application of the transform, as the log likelihood is a deterministic, non-linear transformation. For both the optimisation and integration approaches, the objective needs to be deterministic, so we define the UTGP marginal likelihood as

$$\mathcal{L}_{\text{UT}} = \sum_{i=0}^{2d} W_m^{(i)} \mathcal{L}^{(i)}. \tag{3.61}$$

This also gives us easy access to the derivative of the transformed log likelihood, which we use in gradient-based optimisers:

$$\frac{\partial \mathcal{L}_{\text{UT}}}{\partial \theta} = \sum_{i=0}^{2d} W_m^{(i)} \frac{\partial \mathcal{L}^{(i)}}{\partial \theta}. \tag{3.62}$$

This shows that the training overhead is increased by a factor of $(2d + 1)$, as we are evaluating the log likelihood for each of the sigma GPs, each necessitating

an inversion of the covariance matrix $\mathbf{K}$. Our simple UTGP further simplifies learning the hyperparameters: for a stationary kernel, we train one sigma GP on its own dataset (e.g. $\mathcal{GP}^{(0)}$) and set the hyperparameters of all the other sigma GPs equal to those of $\mathcal{GP}^{(0)}$'s hyperparameters. This becomes clear when we look at how the datasets differ: they are simply copies of each other shifted by $\alpha \left(\sqrt{(d+k)\Sigma_x}\right)_i$ along one dimension of $\mathbf{x}$, so the intra-dataset distances are the same in every sigma data set.

Choosing the number of sigma points

There are two possible ways of applying the UT to GP regression. The first option is outlined above, where we apply the same sigma point transformation to every instance in the data set, and create $2d+1$ sigma data sets and GPs ("simple" UTGP).

The other option would be to define $2d$ sigma points for each point in the training set, in order to create $2Nd + 1$ sigma data sets for a training set with N instances ("full" UTGP). Considering the computational complexity of the full UTGP, it is clear that creating $2Nd + 1$ GPs and combining their outputs becomes computationally expensive even for moderately-sized training sets of a few hundred instances. If we were to perform GP inference via slice sampling with s samples, then this increases the number of GPs whose posteriors we are combining to $s(2Nd + 1)$, making the full UTGP model unattractive from a computational point of view.

To visualise the differences of the simple and full UTGP models, we will train both models on data from the following one-dimensional test function defined over the range $-1 < \mathbf{x} < 1$:

$$\begin{aligned} t &= \frac{5}{2}(x+1), \\ f(t) &= t\sin(3t)\exp(-t). \end{aligned} \tag{3.63}$$

Note that the transformation of x to t is simply a scaling of the input domain to achieve the desired shape when evaluated in the range $[-1, 1]$. We refer to this function as `decaying-sin` and it is plotted in Figure 3.2.

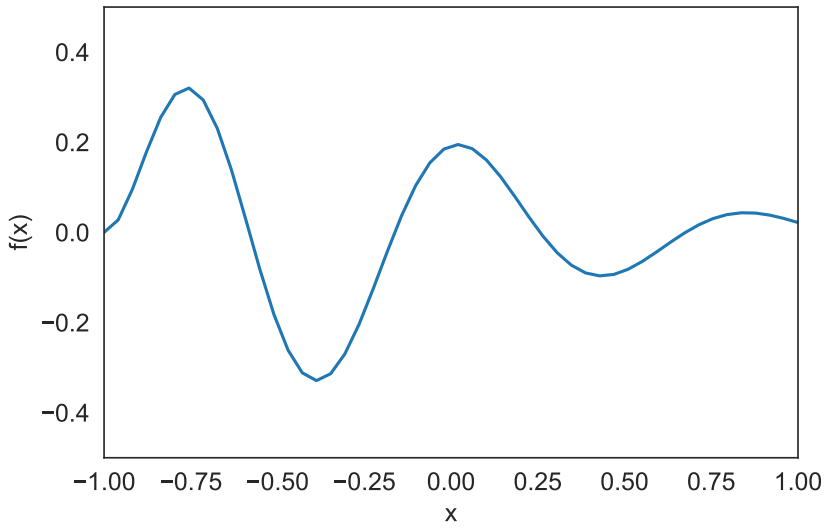
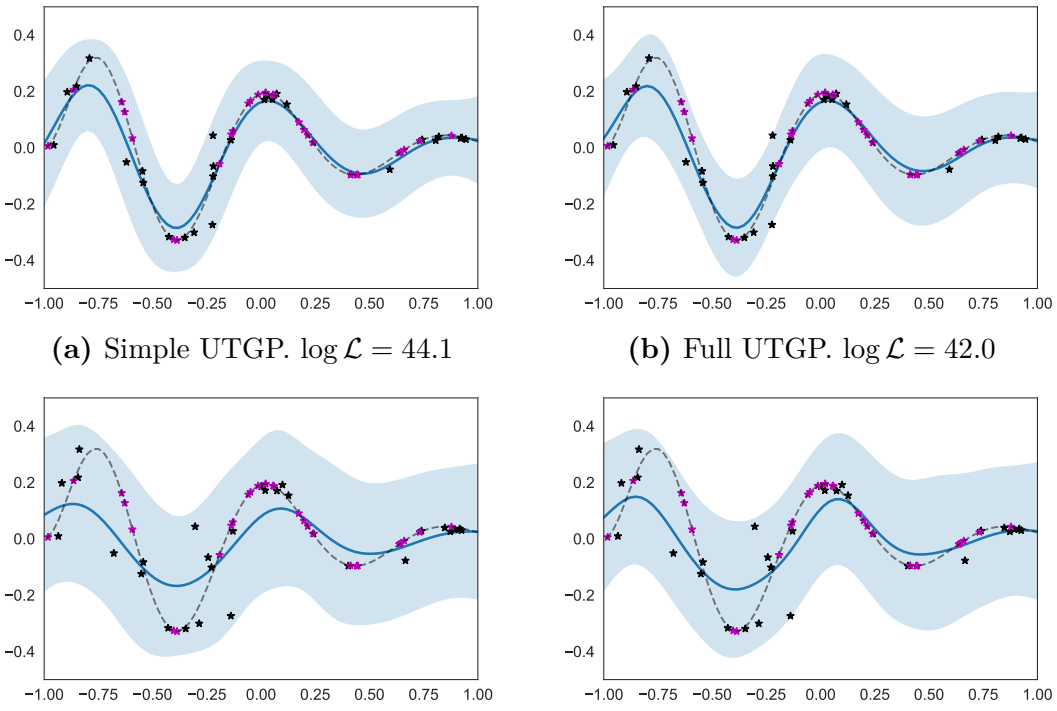


Figure 3.2: Visualisation of the `decaying-sin` test function defined in Equation (3.63).

Figure 3.3 shows a comparison of the posteriors of the simple UTGP and the full UTGP. We sampled 30 training locations in the range $[-1, 1]$ and optimised the model hyperparameters using multi-started gradient descent. The posteriors for $\epsilon_x \sim \mathcal{N}(0, 0.05^2)$ are shown in Figures 3.3a and 3.3b, and $\epsilon_x \sim \mathcal{N}(0, 0.1^2)$ in Figures 3.3c and 3.3d. We also sampled 30 test points (magenta points) and evaluated the sum of the predictive log likelihood of the simple and full UTGPs on these points.

The predictive log likelihoods have similar values for both the simple and full UTGP models, with the simple UTGP combining the results of 3 GPs and the full UTGP utilising 61 GPs. The predictive log likelihood of the full UTGP was lower than the simple UTGP in the low-noise scenario, but higher in the high-noise scenario, which may suggest that the more complete treatment of the input noise in the full UTGP may make it more robust to higher amounts of noise. We observed this behaviour for a number of different random initialisations, and suggest that using the simple UTGP should be sufficient to gain an idea of the efficacy of the UT for GP regression with uncertain inputs at a fraction of the computational cost.

Further exploration of the full UTGP is computationally prohibitively expensive, but points towards an interesting research direction focusing on developing an efficient scheme for computing and combining the posteriors of multiple GPs. In



(a) Simple UTGP. $\log \mathcal{L} = 44.1$ **(b)** Full UTGP. $\log \mathcal{L} = 42.0$

(c) Simple UTGP. $\log \mathcal{L} = 26.4$ **(d)** Full UTGP. $\log \mathcal{L} = 28.5$

Figure 3.3: Comparing the posteriors of the simple and full UTGP models. Training points are shown as black stars, test points are in red. The posterior mean (solid blue) and two standard deviations (shaded blue area) are shown. The ground truth `decaying-sin` function is shown as the dashed grey curve. The case $\sigma_x = 0.05$ is shown in (a) and (b), and $\sigma_x = 0.1$ is shown in (c) and (d). The predictive log likelihood of each posterior on the test points is reported in each subfigure's caption.

light of this computational limit, we will compare only the simple UTGP against the other uncertain-input GP models described thus far, and will refer to the simple UTGP as just the UTGP henceforth.

3.6 Experimental evaluation

In this section we will empirically compare the uncertain-input methods discussed above. A number of different metrics can be used to analyse regression performance of models, e.g. mean squared error, mean absolute error or predictive log likelihood of the posterior on the test samples. The predictive log likelihood, which is the probability to which the posterior explains the test data, takes into account both the accuracy of the posterior mean, as well as the model's uncertainty, so we

will use this metric in our evaluation below to give a more complete picture of a model's performance. Since the test set is the same for every iteration, we do not need to scale the predictive likelihood by the number of data points to facilitate a like-for-like comparison.

3.6.1 Experiment setup

In all of our experiments, we trained our models on data where the inputs have been corrupted by a known noise distribution. We started by choosing a function f and randomly sampling locations $\mathbf{X}$ uniformly in a bounded region $[-1, 1]^d$. The input domain of f was scaled to this region. We then split the sampled locations into $\mathbf{X}_{\text{train}}$ and $\mathbf{X}_{\text{test}}$[1]. For the test data, we evaluated f at each test input: $\mathbf{f}_{\text{test}} = f(\mathbf{X}_{\text{test}})$. For our training data, we evaluated f at perturbed versions of $\mathbf{X}_{\text{train}}$, by adding independent noise to each dimension, to simulate an observation process with input noise: $\mathbf{y}_{\text{train}} = f(\mathbf{X}_{\text{train}} + \boldsymbol{\epsilon}_x) + \epsilon_y$. The additive input noise was drawn from a Gaussian distribution $\boldsymbol{\epsilon}_x \sim \mathcal{N}(\mathbf{0}, \sigma_x^2 \boldsymbol{I})$ with $\sigma_x \in \{0.05, 0.1\}$. We added a small amount of noise to the function values from a Gaussian $\epsilon_y \sim \mathcal{N}(0, \sigma_y^2)$ with $\sigma_y = 0.001$ to improve numerical stability.

During initial testing, we noticed that choosing $\sigma_x > 0.1$ lead to the posterior very often reverting to a flat mean. After testing different values we chose $\sigma_x \in \{0.05, 0.1\}$ to simulate low and medium amounts of noise respectively while reducing the chance of the models falling back to the prior during training.

We fixed the input noise parameter in the NIGP, Analytic, RandFunc and UTGP models to the true value. The UTGP model as shown above does not allow us to learn the noise parameter, so this decision facilitates a like-for-like comparison of the models. Fixing the noise parameter also means that all models (with exception of the RQ kernel) are optimising the same number of hyperparameters, so the uncertain-input models are not disadvantaged from an optimisation point of view.

[1]Since our target application domain is Bayesian optimisation, we are primarily interested in the performance for *interpolation*, rather than extrapolation, so it is fine to select training and test examples from the same hypercube.

Hyperparameter inference was performed using multi-started gradient descent on the marginal log likelihood with 10 restarts.

3.6.2 Comparing model posteriors in 1D

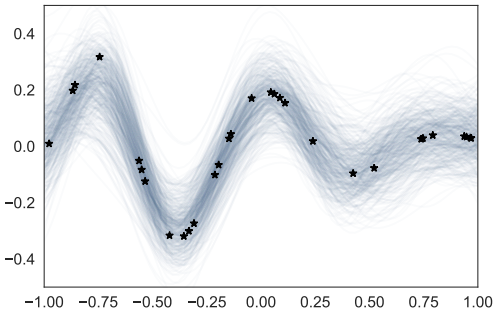
We repeated the 1-D posterior comparison from the previous section that used the `decaying-sin` test function in Equation (3.63) and compared the posteriors of the uncertain-input models and the standard GP. We used 30 training points and evaluated each model on 30 held-out test locations.

In this comparison we added a further illustrative sampling-based model to visualise the "ideal" posterior that could result from solving Equation (3.3). For this baseline, we trained 50 GPs with a RBF kernel on $\{\mathbf{X}_{\text{train}} + \boldsymbol{\epsilon}_x, f(\mathbf{X}_{\text{train}} + \boldsymbol{\epsilon}_x)\}$ for 50 different realisations of $\boldsymbol{\epsilon}_x \sim \mathcal{N}(0, \sigma_x^2 \boldsymbol{I})$. We then sampled 100 functions from the posterior of each trained GP, which together form the sampling baseline for our comparison. Figures 3.4a and 3.5a show a subset of these function samples for $\sigma_x = 0.05$ and $\sigma_x = 0.1$ respectively, and the sampled functions have been summarised into the "ideal" posterior in Figures 3.4b and 3.5b.

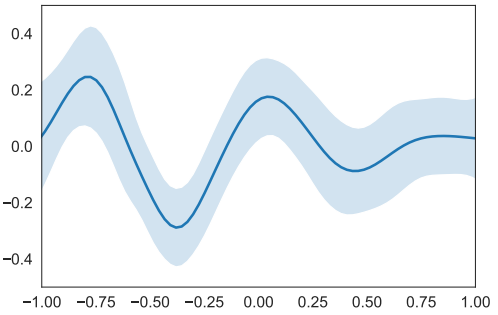
Figure 3.4 show the posterior mean and the 2σ confidence interval for all models for $\sigma_x = 0.05$. Figure 3.5 shows the resulting posteriors for $\sigma_x = 0.1$. Each model caption also reports the predictive log likelihood of each model on the 30 held-out test data points.

In the $\sigma_x = 0.05$ case, we can see that visually the uncertain-input models and the RBF and RQ kernel GPs are quite similar. The predictive log likelihoods are very similar to each other, and the UTGP predictive likelihood is lower than the other models, which can be attributed by the slightly inflated posterior variance compared to the other models. Somewhat surprisingly, the GP models with the RBF and RQ kernels are doing as well as the uncertain-input GP models that are designed specifically to tackle this setting.

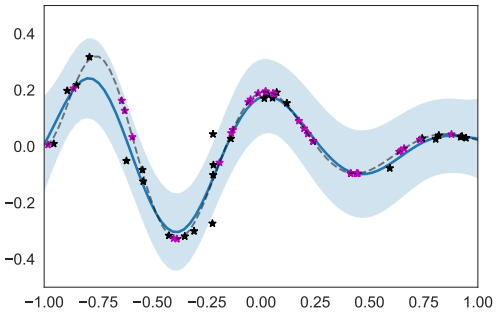
When we increase the noise to $\sigma_x = 0.1$, the RandFunc and UTGP models deviate from the other models. Visually, the posterior of the RandFunc model is more varied, compared to the more smoothly-varying posteriors of the other



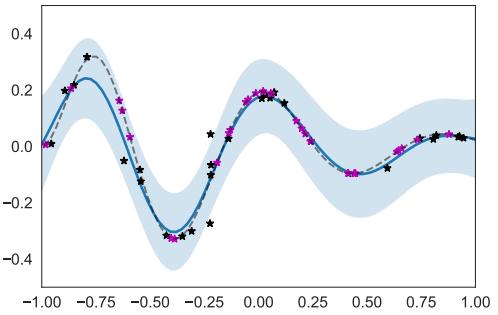
(a) Some example function samples



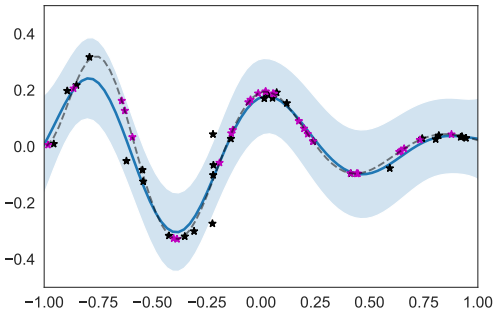
(b) Mean and $\pm 2\sigma$ of the function samples



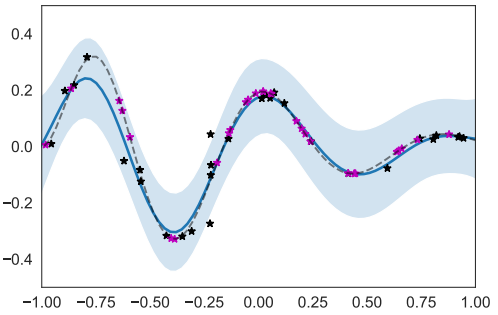
(c) RBF kernel. $\log \mathcal{L} = 48.7$



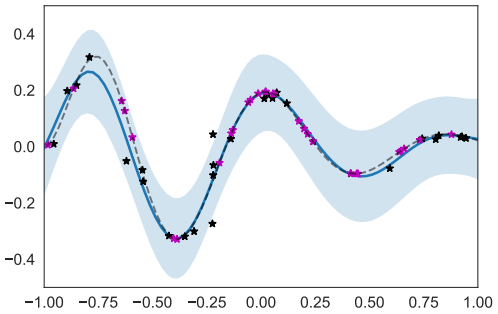
(d) RQ kernel. $\log \mathcal{L} = 48.7$



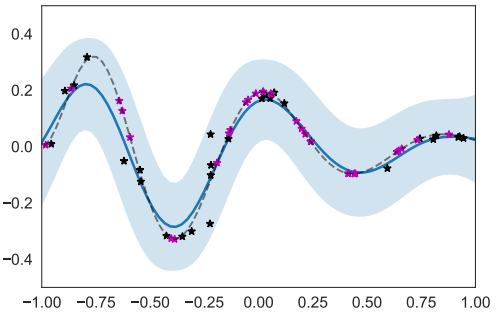
(e) Analytic. $\log \mathcal{L} = 48.7$



(f) NIGP. $\log \mathcal{L} = 48.8$

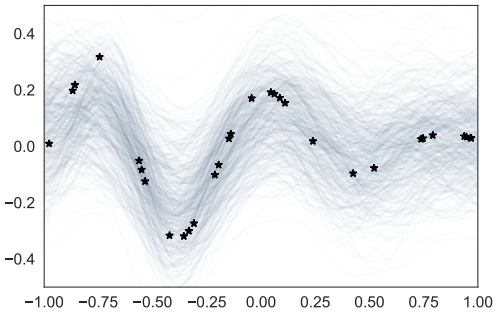


(g) RandFunc. $\log \mathcal{L} = 48.2$

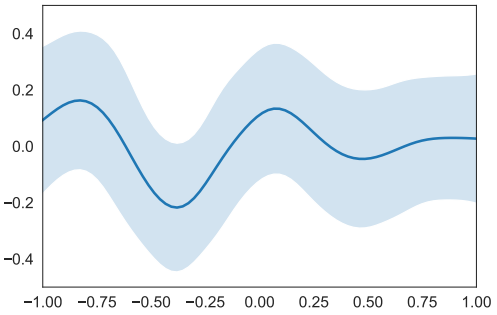


(h) UTGP. $\log \mathcal{L} = 44.1$

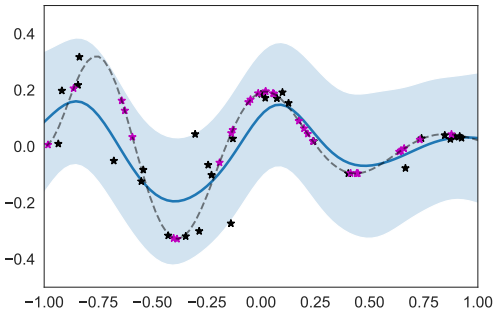
Figure 3.4: Posteriors of the trained models for the 1D sample function for $\sigma_x = 0.05$. The shading represents the 95% confidence interval.



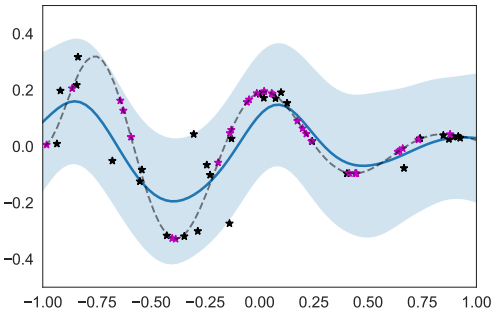
(a) Some example function samples



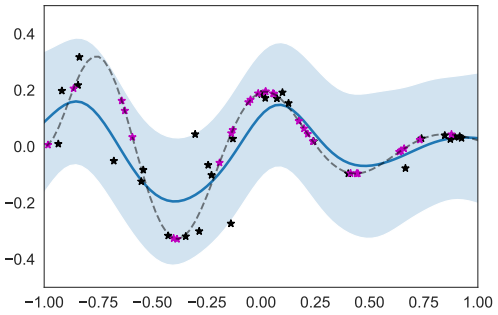
(b) Mean and $\pm 2\sigma$ of the function samples



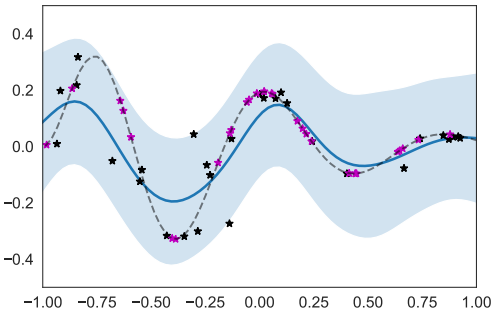
(c) RBF kernel. $\log \mathcal{L} = 30.0$



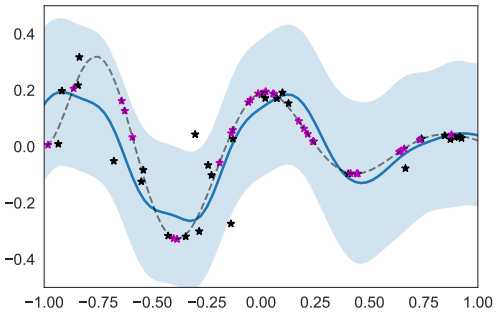
(d) RQ kernel. $\log \mathcal{L} = 30.0$



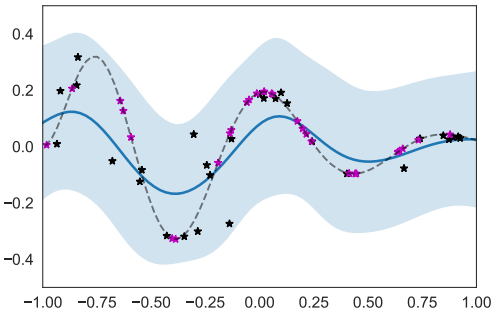
(e) Analytic. $\log \mathcal{L} = 30.0$



(f) NIGP. $\log \mathcal{L} = 30.0$



(g) RandFunc. $\log \mathcal{L} = 28.4$



(h) UTGP. $\log \mathcal{L} = 26.4$

Figure 3.5: Posteriors of the trained models for the 1D sample function for $\sigma_x = 0.1$. The shading represents the 95% confidence interval.

models. The predictive log likelihoods show no difference in predictive performance between the RBF and RQ kernels as well as the Analytic and NIGP models. The RandFunc and UTGP models' predictive log likelihoods, 28.4 and 26.4 respectively, are both lower than the other models' 30.0. Despite these differences, the posteriors all look sensible, so the 1D comparison does not highlight any obvious issues that would discount a model at this stage.

It is surprising how well the standard GP (RBF and RQ kernels) compares against models that are especially designed to solve this setting of uncertain-inputs. The standard GP's RBF kernel is quite similar to the Analytic model's kernel in Equation 3.24, so it is possible that training the models yields similar results. The Analytic model's kernel can indeed be roughly approximated as a RBF with an inflated lengthscale.

3.6.3 Higher-dimensional experiments

We now move to testing the models' regression performance with varying amounts of data on a number of different tasks. We chose the following test functions for the evaluation:

- The `decaying-sin` test function from Equation (3.63).
- Functions (2D, 3D and 4D) drawn from a GP with an RBF kernel. This is an in-model test, since the function is generated by a kernel of the same class as the one used in the models. We selected a RBF kernel with known hyperparameters and sampled a function defined on a large random grid. Since we need to evaluate the sampled function at new locations, we trained a GP with an RBF kernel on these samples and used its posterior mean as the value of the function at a queried location.
- Popular mathematical test functions. These functions are often used to test optimisation algorithms, due to their complex surfaces, which makes these tasks well-suited to our evaluation, since we are interested in how well the different GP methods can model these challenging surfaces. We

chose the following tasks: `branin-2d`, `ackley-2d`, `ackley-4d`, `ackley-8d`, `michalewicz-2d`, `michalewicz-4d` and `michalewicz-8d`, which are defined in Table 3.1. Details on these (and many other) popular test functions can be found at https://www.sfu.ca/~ssurjano/optimization.html.

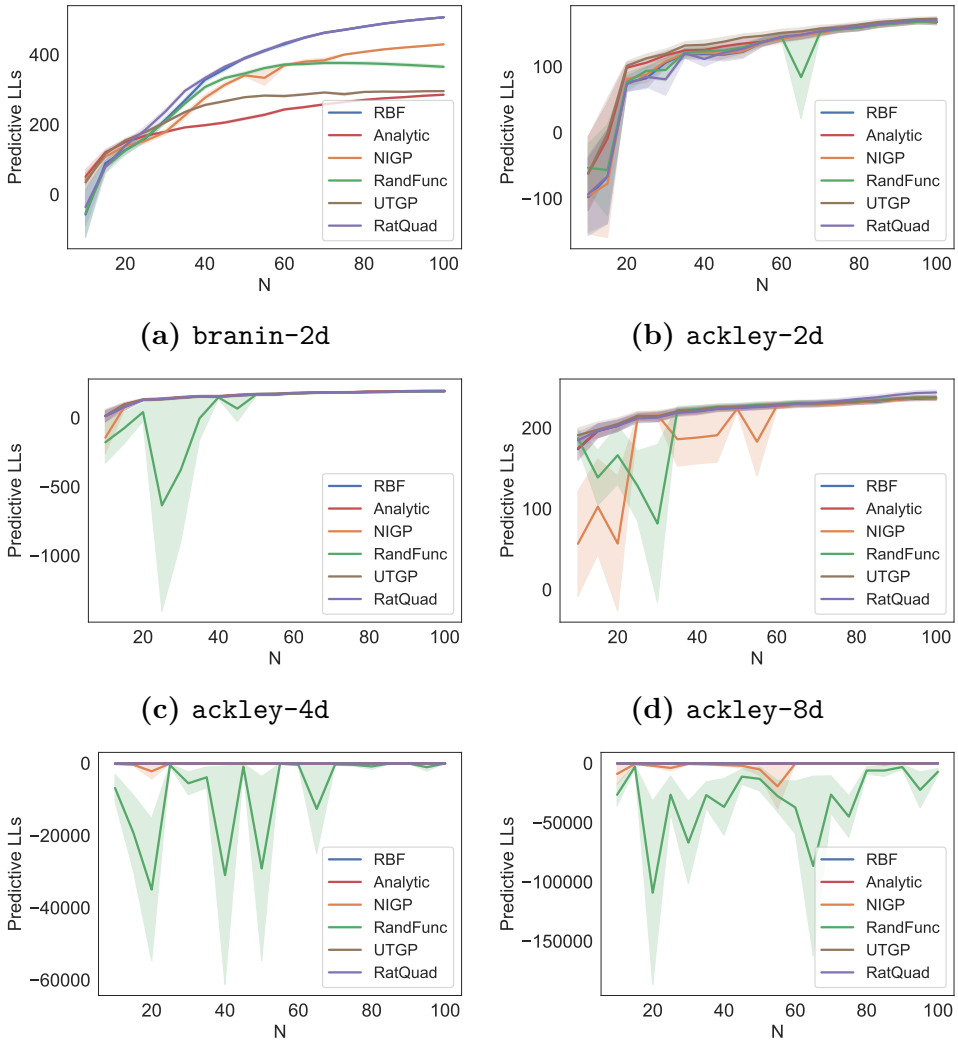
Table 3.1: Mathematical expressions of the synthetic test problems.

Function	Mathematical expression	Input range
`branin`	$\left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10(1 - \frac{1}{8\pi})\cos(x_1) + 10$	$x_1 \in [-5, 10], x_2 \in [0, 15]$
`egg`	$-(x_2 + 47)\sin(\sqrt{\lvert x_2 + \frac{x_1}{2} + 47\rvert}) - x_1 \sin(\sqrt{\lvert x_1 - (x_2 + 47)\rvert})$	$[-512, 512]^2$
`ack`	$-20\exp\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^d x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^d \cos(2\pi x_i)\right) + 20 + \exp(1)$	$[-32.768, 32.768]^d$
`mic`	$-\sum_{i=1}^d \sin(x_i)\sin^{20}\left(\frac{ix_i^2}{\pi}\right)$	$[0, \pi]^d$

All test functions' input spaces were scaled to $[-1, 1]^d$. For each experiment we first created a test set with 50 observations $\{\mathbf{X}_{\text{test}}, \mathbf{f}_{\text{test}}\}$ of the test function. We then evaluated the predictive log likelihoods of the different models when trained on 10, 20, ..., 100 training points. Since our motivation stems from the domain of Bayesian optimisation, we focus on cases with small to medium amounts of data.

Selecting the train and test data from the same space is reasonable for our purposes, as we are interested in evaluating the models' efficacy as regression models for use in BO, where the surrogate model is used to *interpolate* between known values, rather than extrapolation (more relevant in forecasting applications).

The predictive log likelihoods for the different regression models are shown in Figures 3.6 and 3.7 for $\sigma_x = 0.05$ and $\sigma_x = 0.1$ respectively. We immediately see that the NIGP and RandFunc methods display large swings in their predictive log likelihoods, whereas the Analytic, UTGP and standard GP (RBF and RQ) methods perform more reliably – there are no significant swings in their predictive performance across the experiments. Since the training data, test data and hyperparameter optimisation approaches are identical across all methods, this points to some pathologies unique to the NIGP and RandFunc approaches, which we will discuss below.



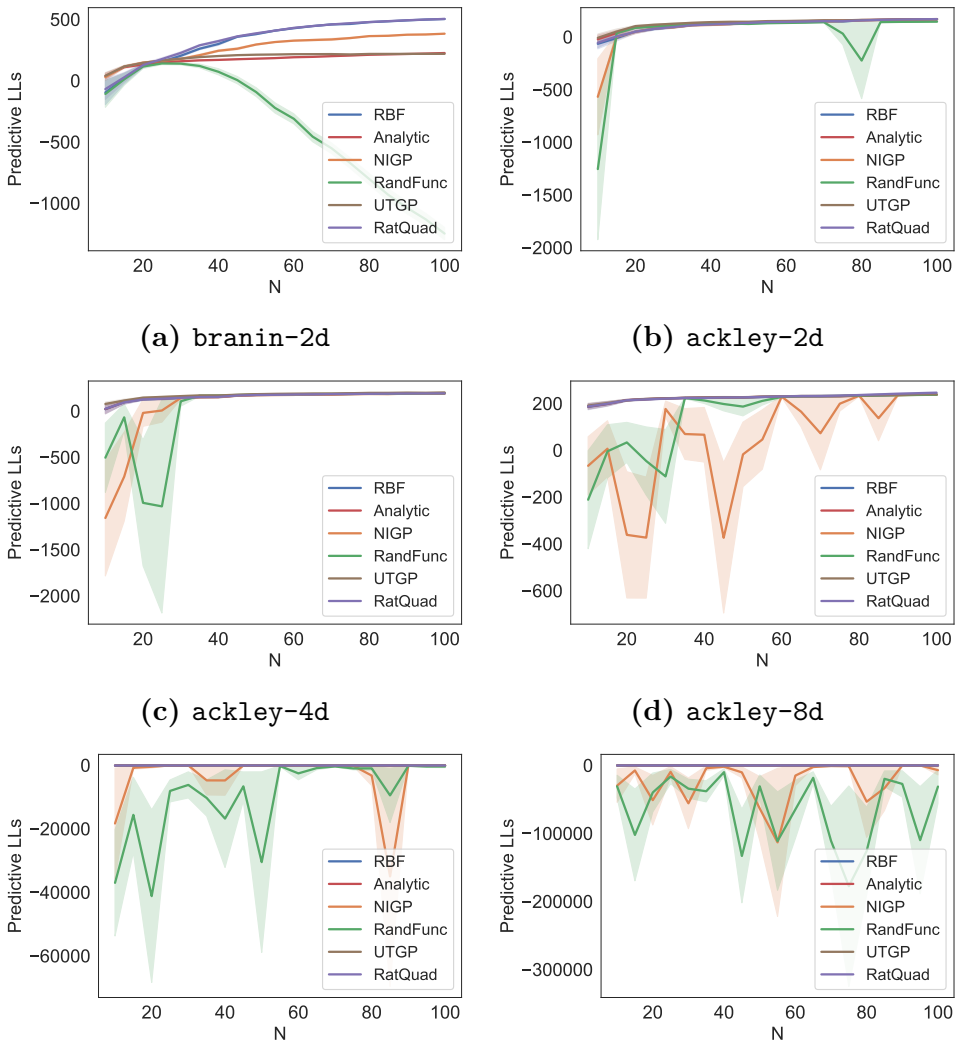
(a) `branin-2d` (b) `ackley-2d`

(c) `ackley-4d` (d) `ackley-8d`

(e) `michalewicz-4d` (f) `michalewicz-8d`

Figure 3.6: Mean and standard error of the test set predictive log likelihood on test functions with $\sigma_x = 0.05$.

Our empirical evaluation is summarised in Tables 3.2 and 3.3. In Table 3.2 we show the mean and standard error of the predictive log likelihoods of the models for $N_{\text{train}} = 50$ and $N_{\text{train}} = 100$. Table 3.3 shows the squared difference between each model's posterior mean and the test points. We chose to also compute the squared error of the mean, as this can provide an additional understanding for a model's performance: If a model has a low predictive likelihood but also a low squared error, this indicates that there may be issues in the variance computation, since the posterior mean is still close to the test data.



(a) `branin-2d` (b) `ackley-2d`

(c) `ackley-4d` (d) `ackley-8d`

(e) `michalewicz-4d` (f) `michalewicz-8d`

Figure 3.7: Mean and standard error of the test set predictive log likelihood on test functions with $\sigma_x = 0.1$.

In terms of predictive log likelihood, the UTGP was the best model more often than any other model (14 instances), followed by the standard GP with a RQ kernel. The RandFunc and NIGP methods performed very poorly in some cases, which we will analyse in the model discussion below.

This is a surprising result: existing uncertain-input models perform *worse* than a standard GP when modelling noisy-input data. Comparing the performance of the RandFunc and NIGP models with the UTGP, we see a similar pattern emerge as the one we see when comparing the EKF with the UKF. The linearisation of the

uncertainty, which is the common thread between the RandFunc, NIGP and EKF, could be the cause of the weaker performance compared to the UT-based alternatives.

The standard GP and the UTGP both do well, often achieving similar performance to each other. Our analysis of the simple and full UTGP formulations provides an additional avenue of research that could improve the modelling performance of the UTGP even further (as we saw in the medium-noise example in Figure 3.3). Until a scalable formulation of the UTGP is developed, this leads us to conclude that, for most purposes, the standard GP is sufficient for modelling data with noisy inputs.

3.7 Model discussion

Standard GP

The standard GP with both RBF and RQ kernels performed well in our empirical evaluation. The RQ kernel's predictive likelihoods were often better than the RBF kernel, but even then the difference is often quite small. This performance difference could be explained by the fact that the RQ kernel models a weighted sum of RBF kernels, and so it is more able to model variations that arise from the noise on the inputs.

Analytic model

The Analytic model performed well, but rarely reached the scores of the standard GP. This is surprising, as our earlier discussion highlighted its similarity to a RBF kernel with an inflated lengthscale. The noise parameter was fixed for the Analytic model, so the optimisation of the hyperparameters was of equal complexity as the standard GP with RBF kernel. On the other hand, fixing the input noise parameter may have restricted its ability to fit the data as well as it could have with a free input noise hyperparameter.

RandFunc

The performance of the RandFunc model was unreliable in our evaluation. This can be explained by the fact that it was one of the more numerically unstable models

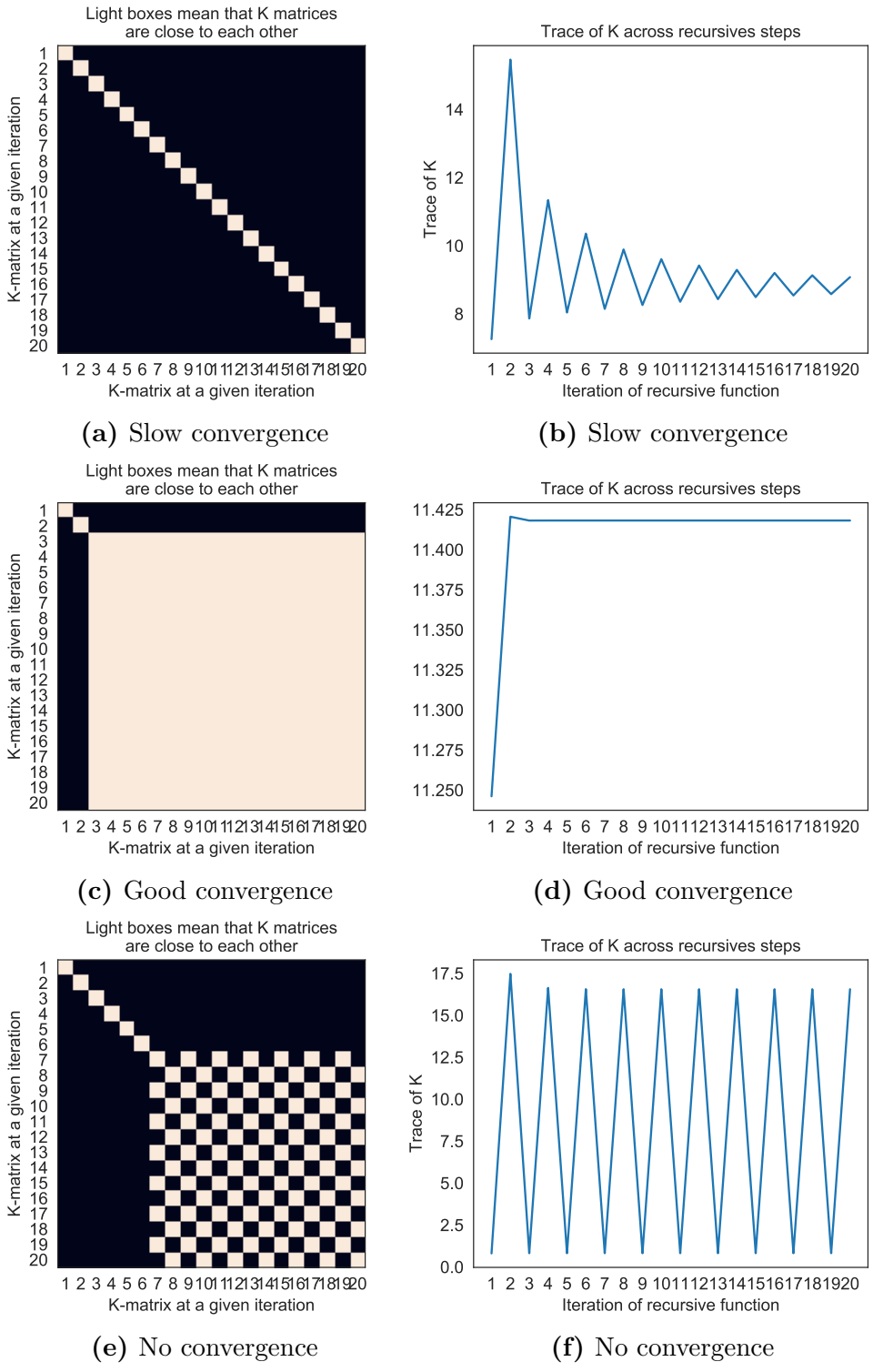
in our comparison. The training objective of the RandFunc model requires the fifth-order derivative of the kernel: twice w.r.t. the first input, twice w.r.t. the second input (as seen in Equation 3.31), and then once w.r.t. the hyperparameter in question to compute the gradient of the log marginal likelihood (Equation (2.41)). For the RBF kernel, this results in a term containing l^{-9}, where l is the lengthscale. We observed lengthscales of the order of magnitude between 0.01 and 100, so this can lead to terms in the kernel of the order 10^{-18} or 10^{18}, which can cause numerical instability.

NIGP

The NIGP performed similarly to the other methods in some cases, and performed very poorly in others. It is also the most computationally intensive model in our comparison, due to the recursive computation involved in integrating the input noise into the posterior via the term $\mathbf{\Delta}_f \Sigma_x \mathbf{\Delta}_f^T$, that occurs in both the mean and variance equations (see Equations (3.35) and (3.36)).

Once the extra term is computed and the posterior mean and variances have been updated, this changes our derivative of the posterior mean and hence $\mathbf{\Delta}_f$, so we need to repeat this step. This is the part of the model inference that takes most of the computation time. Empirically, we often see this recursive relationship settling quite fast, say in 4-5 steps, but sometimes it takes many more iterations or doesn't converge to a single solution at all. It is not clear how often this update should be repeated and if at all convergence can be guaranteed.

In Figure 3.8 we show results after having trained a NIGP with RBF kernel on the `decaying-sin` function and tracked how the total covariance matrix $\left(K + \sigma I + \text{diag}\{\mathbf{\Delta}_f \Sigma_x \mathbf{\Delta}_f^T\}\right)$ evolved across recursive steps, computing the derivative of the posterior mean and then updating the posterior mean and variance for 10 steps. The left column of plots shows the difference between the covariance matrices evaluated after the recursive steps. We defined two matrices to be "equal" if the absolute difference between every element in the first matrix and its corresponding element in the second matrix was within a tolerance of 10^{-6}. Light squares indicate



(a) Slow convergence **(b)** Slow convergence

(c) Good convergence **(d)** Good convergence

(e) No convergence **(f)** No convergence

Figure 3.8: NIGP recursive step analysis. The left column shows whether K-matrices of the model are similar to each other across multiple recursive steps. The right column shows how the trace of the K-matrix evolves over the recursive steps. The optimal model behaviour is shown in the central row.

that the selected matrices are equal according to this definition. This means that we expect to start with light boxes along the diagonal only, which should then grow into a large light block, like in Figure 3.8c, which means that the covariance matrix has stabilised and is not changing any more across recursive steps.

The right column of plots shows the trace of the noise-related term. As only the trace of the NIGP's extra term affects our covariance matrix, we are primarily interested in how those values develop across iterations. Here the optimal desired behaviour is shown in Figure 3.8d, where the trace is stabilising quickly.

There are two failure modes that we experienced in our analyses. The first failure mode is shown in Figures 3.8a and 3.8b, corresponding to slow convergence. We can see that 20 recursive steps are not enough, as the trace is still oscillating and decaying slowly. Increasing the number of recursive steps allows the posterior to stabilise, but this comes at the expense of increased computational effort.

The second failure mode is shown in Figures 3.8e and 3.8f, where the matrix does not settle at all, and instead oscillates between two distinct (and equal) states. No convergence is observed in this case, even after running an increased number of recursive steps. The values of the matrices oscillate between a matrix with small diagonal values and one with an inflated diagonal. The fact that it is oscillating between two equal matrices is made clear in Figure 3.8e, as the checkered shape means that the matrices at steps 8, 10, 12, ... are the same, as are the matrices at steps 7, 9, 11, ...

So in conclusion, the oscillatory behaviour of the recursive computation in the NIGP often decays quite quickly in practice, but it is not clear what properties of hyperparameters or the training data lead to the non-decaying behaviour. The NIGP incurs a higher computational cost than the other models in our comparison and when used in practice we will not be able to tell if the model will be well-behaved.

UTGP

Our empirical evaluation considered only the simple UTGP, and we see that it performs competitively. It also inherits the numerical stability of the standard

GP, as a result of the form of its training objective (Equation (3.61)). It was the best method more often than any other method, and close to the standard GP in many cases, and we believe further improvements to the model could provide even better results than the ones we see here. The simple UTGP is very simple computationally, as it only scales with the dimensionality of the input space. GPs are often restricted to relatively low dimensional data, due to the fact that in high dimensions the distances between points are larger and hence learning based on distance metrics becomes challenging. This means that the UTGP can be viable in practice and our brief comparison of the full and simple UTGPs provides an incentive to find a scalable formulation of the full UTGP.

3.8 Conclusion

In this chapter we contrasted a total of five models for modelling smooth functions where the data is corrupted on both the inputs and the outputs: three existing uncertain-input GP approaches, a novel approach based on the unscented transform, and a standard GP. As far as we are aware, the experiments conducted here provide the first side-by-side comparison of popular uncertain-input GP methods for regression. Our novel method, the UTGP, was analysed in its "simple" formulation, due to scaling challenges of the "full" formulation. The simple UTGP was the best-performing uncertain-input approach, which is an encouraging result and motivates researching methods to make the full UTGP scalable.

We note that the models here were all provided with the true underlying noise variance. The three existing uncertain-input GP models (Analytic, RandFunc, NIGP) allow this variance to be treated as a hyperparameter that can be learnt from the data if we wish to. Since the UTGP does not allow the noise to be learnt in its current formulation, we defined experiments with fixed input noise variance to facilitate a comparison for all models.

Finally, we conclude that specialised uncertain-input methods perform as well as, and often worse than, a standard GP. The UTGP was the best-performing uncertain-input model, but the results suggest that when we turn to using Bayesian

optimisation in an uncertain-input environment, taking into account computational complexity and model accuracy, using a standard GP surrogate is likely to be sufficient to model the input-uncertainty in the data.

Task	Noise	N_train	RBF	Analytic	NIGP	RandFunc	UTGP	RatQuad
ackley-2d	0.05	50	122.5 (72.8)	135.0 (50.8)	123.2 (63.7)	129.3 (53.3)	**144.1 (39.1)**	127.1 (52.1)
		100	167.3 (28.9)	167.3 (28.7)	168.9 (26.8)	168.4 (24.9)	**172.5 (22.8)**	170.4 (26.7)
	0.1	50	132.4 (58.9)	136.9 (58.3)	130.5 (59.8)	123.5 (47.4)	**144.0 (41.4)**	133.9 (53.5)
		100	168.6 (28.2)	**169.7 (27.8)**	168.1 (29.1)	145.2 (26.9)	167.8 (13.8)	169.6 (29.0)
ackley-4d	0.05	50	167.2 (64.2)	168.3 (58.2)	165.8 (63.8)	168.5 (60.0)	**172.5 (52.9)**	166.2 (62.6)
		100	188.6 (44.9)	190.3 (40.4)	190.7 (40.6)	192.5 (38.4)	**196.1 (36.6)**	191.1 (40.7)
	0.1	50	176.8 (36.1)	177.5 (36.6)	173.5 (37.7)	179.6 (30.0)	**183.6 (32.6)**	176.2 (38.8)
		100	192.0 (54.9)	193.3 (54.1)	191.1 (58.9)	193.8 (44.0)	**201.8 (29.7)**	193.0 (54.1)
ackley-8d	0.05	50	225.6 (28.4)	225.8 (28.4)	224.4 (30.5)	**226.7 (27.3)**	225.9 (28.3)	225.5 (28.4)
		100	236.6 (19.5)	236.6 (20.1)	236.1 (21.8)	237.9 (18.2)	237.8 (18.8)	**244.4 (23.4)**
	0.1	50	225.5 (23.9)	226.1 (23.6)	52.7 (880.9)	186.8 (288.0)	**226.4 (22.6)**	225.7 (23.8)
		100	237.2 (18.0)	237.9 (18.1)	236.5 (18.9)	237.4 (14.2)	239.4 (15.6)	**245.5 (20.8)**
decaying-sin	0.05	50	**543.1 (23.3)**	309.6 (11.4)	519.2 (14.2)	-4783.9 (1675.5)	225.2 (8.0)	543.0 (23.3)
		100	**577.9 (10.0)**	349.4 (6.9)	558.7 (8.1)	-7800.9 (1496.4)	225.7 (7.7)	577.8 (10.0)
	0.1	50	550.3 (23.4)	235.3 (12.2)	497.2 (16.1)	-2337.2 (1503.7)	131.7 (7.2)	**550.5 (23.3)**
		100	**579.8 (8.4)**	268.9 (18.4)	546.0 (8.4)	-2878.4 (2229.1)	131.6 (6.8)	579.7 (8.4)
draw-from-gp-2d	0.05	50	84.7 (35.9)	23.2 (19.1)	39.9 (237.9)	-187.6 (1634.2)	24.6 (18.4)	**84.8 (38.1)**
		100	**280.5 (59.0)**	90.3 (12.2)	204.1 (30.3)	-1808.6 (1098.2)	61.9 (31.9)	279.7 (61.3)
	0.1	50	**81.9 (27.4)**	-10.9 (18.2)	56.4 (39.4)	-40.4 (27.5)	-28.1 (16.1)	77.1 (41.7)
		100	**273.3 (57.1)**	39.4 (13.0)	169.6 (17.5)	-212.0 (90.5)	-13.7 (13.4)	273.0 (59.2)
draw-from-gp-3d	0.05	50	**124.6 (118.2)**	94.9 (80.8)	-1927.9 (10189.0)	123.4 (102.7)	106.9 (67.8)	124.1 (117.2)
		100	200.8 (176.1)	145.5 (104.5)	-6267.4 (43701.4)	166.5 (146.8)	139.2 (92.0)	**208.7 (172.5)**
	0.1	50	16.4 (896.5)	36.8 (400.4)	-561.9 (4172.2)	-21.3 (536.2)	**67.8 (168.2)**	17.5 (886.4)
		100	212.6 (163.9)	125.8 (95.4)	-1055.7 (6871.3)	-1086.2 (5304.2)	120.1 (80.6)	**217.5 (160.5)**
michalewicz-2d	0.05	50	5.0 (37.6)	10.4 (25.3)	2.1 (37.9)	2.0 (21.5)	**15.3 (22.7)**	7.1 (37.5)
		100	53.8 (24.5)	54.1 (13.5)	53.9 (24.3)	27.4 (16.6)	52.7 (10.3)	**55.2 (28.9)**
	0.1	50	6.8 (26.4)	4.5 (21.6)	6.4 (27.0)	-15.3 (15.0)	2.1 (13.6)	**7.3 (26.0)**
		100	51.3 (19.9)	39.9 (11.6)	50.2 (21.5)	-14.6 (11.1)	18.5 (11.2)	**54.9 (20.4)**
michalewicz-4d	0.05	50	-67.5 (11.9)	-67.3 (11.6)	-67.5 (11.6)	-29139.0 (185335.6)	-67.3 (11.6)	**-67.2 (12.1)**
		100	-63.5 (11.3)	-63.8 (11.3)	-63.5 (11.4)	-65.3 (11.5)	**-63.2 (10.8)**	-63.3 (11.2)
	0.1	50	-73.0 (22.9)	-70.6 (17.6)	-73.1 (22.6)	-30504.7 (206940.8)	**-70.2 (16.1)**	-73.2 (22.7)
		100	-68.1 (14.4)	-67.1 (14.1)	-67.2 (12.0)	-406.9 (2425.3)	**-65.6 (12.4)**	-67.5 (14.6)
michalewicz-8d	0.05	50	-101.8 (10.1)	-101.6 (9.9)	-6668.4 (43598.7)	-13144.4 (69110.1)	-101.9 (10.1)	**-101.6 (9.8)**
		100	-99.8 (9.1)	**-99.8 (9.0)**	-100.1 (9.0)	-7075.2 (18280.5)	-99.9 (9.2)	-100.1 (8.8)
	0.1	50	-101.9 (12.0)	-101.8 (12.1)	-71535.9 (385762.1)	-30765.8 (121396.4)	-101.7 (11.9)	**-101.7 (11.9)**
		100	-99.5 (9.3)	-99.6 (9.2)	-7980.5 (56828.5)	-31138.5 (177958.7)	**-99.4 (9.4)**	-99.8 (9.3)

Table 3.2: Mean and standard error of test set predictive log likelihoods. Best value is highlighted.

Task	Noise	N_train	RBF	Analytic	NIGP	RandFunc	UTGP	RatQuad
ackley-2d	0.05	50	0.3 (0.1)	**0.3 (0.1)**	0.3 (0.2)	0.3 (0.1)	0.3 (0.2)	0.3 (0.2)
		100	0.2 (0.1)	0.2 (0.1)	0.2 (0.1)	0.2 (0.1)	0.2 (0.1)	**0.2 (0.1)**
	0.1	50	0.3 (0.2)	**0.3 (0.2)**	0.3 (0.2)	0.4 (0.2)	0.3 (0.2)	0.3 (0.2)
		100	0.2 (0.1)	0.2 (0.1)	0.2 (0.1)	0.3 (0.2)	0.2 (0.1)	**0.2 (0.1)**
ackley-4d	0.05	50	0.1 (0.1)	0.1 (0.1)	0.1 (0.1)	**0.1 (0.1)**	0.1 (0.1)	0.1 (0.1)
		100	0.1 (0.1)	0.1 (0.1)	0.1 (0.0)	0.1 (0.0)	**0.1 (0.0)**	0.1 (0.0)
	0.1	50	0.1 (0.1)	0.1 (0.1)	0.1 (0.1)	**0.1 (0.1)**	0.1 (0.1)	0.1 (0.1)
		100	0.1 (0.1)	0.1 (0.0)	0.1 (0.0)	0.1 (0.0)	0.1 (0.1)	**0.1 (0.0)**
ackley-8d	0.05	50	0.1 (0.0)	0.1 (0.0)	0.1 (0.0)	**0.1 (0.0)**	0.1 (0.0)	0.1 (0.0)
		100	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	**0.0 (0.0)**
	0.1	50	0.1 (0.0)	0.1 (0.0)	0.1 (0.0)	**0.1 (0.0)**	0.1 (0.0)	0.1 (0.0)
		100	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	**0.0 (0.0)**
decaying-sin	0.05	50	**0.0 (0.0)**	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
		100	**0.0 (0.0)**	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)
	0.1	50	0.0 (0.0)	0.0 (0.0)	0.0 (0.0)	0.8 (0.1)	0.3 (0.0)	**0.0 (0.0)**
		100	**0.0 (0.0)**	0.0 (0.0)	0.0 (0.0)	0.8 (0.1)	0.3 (0.0)	0.0 (0.0)
draw-from-gp-2d	0.05	50	6.4 (6.3)	8.1 (5.7)	9.7 (19.5)	7.3 (10.9)	**6.0 (5.1)**	6.8 (9.3)
		100	2.0 (11.7)	1.7 (2.0)	**0.8 (1.6)**	2.6 (14.7)	2.8 (16.6)	2.7 (16.9)
	0.1	50	**6.7 (5.5)**	11.2 (6.8)	9.5 (17.9)	11.4 (5.8)	8.6 (5.6)	8.0 (9.7)
		100	2.1 (11.8)	3.1 (2.7)	**1.1 (1.9)**	7.7 (2.9)	3.0 (1.6)	2.7 (16.8)
draw-from-gp-3d	0.05	50	0.8 (1.0)	1.6 (3.9)	1.2 (2.1)	0.8 (1.0)	1.0 (2.1)	**0.8 (1.0)**
		100	1.1 (5.9)	0.4 (0.5)	0.6 (1.8)	0.4 (0.5)	2.1 (6.6)	**0.4 (0.5)**
	0.1	50	**0.8 (0.9)**	1.3 (2.2)	1.0 (1.0)	0.8 (0.8)	2.3 (5.6)	0.8 (0.9)
		100	0.6 (2.1)	0.5 (0.5)	0.7 (1.5)	1.1 (5.1)	1.7 (5.9)	**0.4 (0.5)**
michalewicz-2d	0.05	50	5.3 (1.6)	**5.2 (1.5)**	5.7 (2.0)	5.5 (1.6)	5.3 (1.6)	5.3 (1.6)
		100	2.8 (1.2)	**2.8 (1.1)**	2.8 (1.2)	3.4 (1.3)	2.8 (1.1)	2.9 (1.7)
	0.1	50	**5.7 (1.6)**	5.7 (1.7)	5.7 (1.7)	7.1 (2.2)	6.6 (1.7)	5.7 (1.6)
		100	2.8 (0.9)	3.0 (0.9)	3.0 (1.3)	7.4 (1.8)	5.1 (1.2)	**2.8 (0.9)**
michalewicz-4d	0.05	50	21.6 (3.9)	21.5 (3.8)	21.5 (3.8)	21.5 (3.8)	21.5 (3.8)	**21.4 (3.9)**
		100	20.3 (3.5)	20.3 (3.5)	20.2 (3.5)	20.3 (3.4)	20.2 (3.4)	**20.2 (3.5)**
	0.1	50	22.1 (3.9)	**21.9 (3.7)**	22.1 (3.8)	22.3 (4.0)	22.1 (3.7)	22.1 (3.9)
		100	21.3 (3.4)	21.0 (3.4)	21.2 (3.6)	21.3 (3.5)	**20.9 (3.4)**	21.0 (3.5)
michalewicz-8d	0.05	50	43.6 (7.7)	43.7 (7.7)	43.6 (7.6)	**43.5 (7.6)**	43.8 (7.8)	43.5 (7.6)
		100	42.7 (7.1)	**42.6 (7.1)**	42.8 (7.1)	43.0 (7.1)	42.7 (7.2)	42.9 (7.0)
	0.1	50	42.9 (7.2)	**42.8 (7.2)**	42.8 (7.3)	42.9 (7.2)	42.8 (7.1)	42.8 (7.3)
		100	42.1 (7.1)	42.2 (7.1)	42.1 (7.2)	**42.0 (7.0)**	42.1 (7.3)	42.3 (7.1)

Table 3.3: Mean and standard error of the squared difference between the posterior mean and the test set. Best value is highlighted.

4

Asynchronous batch Bayesian optimisation

Contents

4.1 Introduction

Bayesian optimisation is at its heart a sequential algorithm. In situations where the cost incurred is mainly driven by acquiring samples of f, sequential BO is a

good fit. It is often the case, though, that multiple experiments can be run at the same time in parallel at little to no extra cost. For example, in the case of drug discovery many different compounds can be tested in parallel via high throughput screening equipment (Hernández-Lobato et al., 2017).

Another area amenable to parallelisation is machine learning hyperparameter optimisation. Here in particular BO has been remarkably successful (Snoek et al., 2012), for example it was used to improve the performance of DeepMind's highly-publicised AlphaGo agent (Chen et al., 2018). In such cases, the search space is made up of the hyperparameters of the model (e.g. the numbers of units in each layer of a neural network), and we are interested in identifying the best model according to an error (or accuracy) metric. With the advent of cloud computing and other easily-available computing resources, we are often able to evaluate multiple hyperparameter configurations at the same time. This lead to the development of a number of parallel BO methods that evaluate a batch of k configurations in parallel at each iteration of the optimisation (Contal et al., 2013; Shah and Ghahramani, 2015; Wu and Frazier, 2016; Kathuria et al., 2016; Kandasamy et al., 2018).

While (synchronous) batch BO methods are well-motivated and a good fit for parallelisable tasks, *asynchronous* BO has received surprisingly little attention by the research community. In this chapter, we motivate the use of asynchronous BO and develop a new method to address this setting. Our contributions can be summarised as follows:

- We analyse the benefits and weaknesses of synchronous batch BO methods and make the case for *asynchronous* batch BO.
- We develop a new approach to asynchronous parallel BO, *Penalising Locally for Asynchronous Bayesian Optimisation on k workers* (PLAyBOOK), which uses penalisation-based strategies to prevent redundant batch selection. We show that our approach compares favourably against existing asynchronous methods.

- We propose a new penalisation function, which prevents redundant samples from being chosen. We also propose designing the penalisers using local (instead of global) variability features of the surrogate to more effectively explore the search space.

- We demonstrate empirically that asynchronous methods perform at least as well as their synchronous variants. We also show that PLAyBOOK outperforms its synchronous variants *both* in terms of wall-clock time and sample efficiency, particularly for larger batch sizes. This renders PLAyBOOK a competitive parallel BO method.

4.2 Related work

4.2.1 Batch BO

We can probably credit the work in Snoek et al. (2012) for leading to a surge in interest in developing batch BO methods. The authors demonstrated the efficacy of BO to optimise hyperparameters of expensive-to-train machine learning models, which was particularly well-timed to occur alongside a renewed interest in deep neural networks and a need to find their optimal hyperparameters. The application domain of machine learning hyperparameter tuning naturally leads to the desire to parallelise BO to make use of multi-GPU machines and supercomputers.

An early batch BO method called *q-EI* was proposed in Ginsbourger et al. (2008) and Ginsbourger et al. (2010), where the expected improvement acquisition function is extended to evaluate the utility of a collection of points. The authors show that an analytic result only exists for a batch with 2 points. Larger batch sizes are evaluated using sampling approaches, which quickly becomes computationally intensive. Countering the computational complexity of the multi-point q-EI criterion, the authors suggested heuristics-based strategies that are meant to approximate the sampling-based method and which are computationally more feasible. These heuristics were named *Kriging believer* (KB) and *constant liar* (CL).

The idea behind these heuristics is simple: we create a batch of k points by finding the configurations that a sequential BO algorithm would choose if it were run for k steps. This is achieved by optimising the acquisition function $\alpha(\mathbf{x})$ to identify the configuration $\mathbf{x}_i$ to sample at next, and then updating the surrogate GP data with this configuration paired with the result of the chosen heuristic $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{x}_j, h(\mathbf{x}_j)\}$, where $h(\cdot)$ is the KB or CL heuristic function, until we have filled the batch. The KB heuristic is defined as $h(\mathbf{x}_j) = \mu(\mathbf{x}_j)$, i.e. the value of the posterior mean of the surrogate at the configuration $\mathbf{x}_j$, and the CL heuristic is defined as $h(\mathbf{x}_j) = c$, where c is a constant (often the incumbent best value $\min(y_1, \ldots, y_t)$). In practice, KB is a remarkably effective batch BO technique and provides a strong baseline for comparison.

Other approaches include using knowledge gradient (Wu and Frazier, 2016), Determinental point processes (Kathuria et al., 2016), adding random samples to the sequential BO choice (Contal et al., 2013), maximising an information-theoretic criterion (Shah and Ghahramani, 2015), and sampling-based simulation (Azimi et al., 2010; Kandasamy et al., 2018; Hernández-Lobato et al., 2017)

Another recent batch BO method that demonstrated promising empirical results is Local Penalisation (LP) (Gonzalez et al., 2016). After adding a configuration $\mathbf{x}_j$ to the batch, LP penalises the value of the acquisition function $\alpha_j(\mathbf{x}) = \alpha(\mathbf{x})\phi_{\text{LP}}(\mathbf{x}|\mathbf{x}_j)$ in the neighbourhood of $\mathbf{x}_j$, encouraging diversity in the batch selection. The penalisation function, $\phi_{\text{LP}}(\mathbf{x}|\mathbf{x}_j)$, is defined as

$$\phi_{\text{LP}}(\mathbf{x}|\mathbf{x}_j) = \Phi\left(\frac{\hat{L}\,\|\mathbf{x} - \mathbf{x}_j\| - |\mu(\mathbf{x}_j) - M|}{\sigma(\mathbf{x}_j)}\right), \tag{4.1}$$

where Φ is the Gaussian c.d.f., $\hat{L}$ is the estimated Lipschitz constant of the black-box function f, M is the current best estimate of the optimum location, $\mathbf{x}_j$ is a batch point, and where $\mu(\mathbf{x}_j)$ and $\sigma(\mathbf{x}_j)$ are the posterior mean and variance of the surrogate GP respectively evaluated at the batch point.

The benefit of a penalisation-based approach over the heuristics (KB and CL) is that we avoid updating the surrogate model $(k-1)$ times for a batch of size k. Updating a GP surrogate model can quickly become expensive, as it necessitates

computing the inverse of the $N \times N$ covariance matrix (N is the number of instances in the GP's data), so side-stepping this operation makes for a computationally attractive algorithm for larger batch sizes.

The common theme across all batch BO methods is the desire to identify a collection of configurations that each provide a distinct understanding about the function f and its global optimum configuration.

4.2.2 Asynchronous BO

Asynchronous BO has received surprisingly little attention compared to synchronous BO to date, even though it is a natural extension to the synchronous case. In (Ginsbourger et al., 2011) the authors proposed an approach based on q-EI to incorporate knowledge of the busy configurations. Due to its reliance on sampling, it still suffers from poor scaling.

A more recent method (Kandasamy et al., 2018) utilises Thompson Sampling (TS) to select new batch points. The authors state that the method has attractive scaling, since the method samples and minimises realisations from the surrogate's posterior distribution, one for each configuration in the batch. In the case of a GP surrogate model, it is not clear how we should optimise a sample we draw from the posterior. The practical work-around is to draw a sample defined over a large random grid and to pick the configuration of the maximum value (or minimum in the case of minimisation), forgoing optimisation altogether. The disadvantage of using TS in this context is that it relies heavily on the uncertainty in the surrogate model to ensure that the batch points are well-distributed in the search space.

4.3 Asynchronous vs synchronous BO

In synchronous batch BO algorithms, at the s-th step, a batch $\mathcal{B}_s = \{\mathbf{x}_{s,1}, \ldots, \mathbf{x}_{s,k}\}$ with k configurations is created for evaluation. The BO algorithm then waits until all k jobs in the batch have completed before updating the surrogate model and identifying the next batch $\mathcal{B}_{s+1}$. In cases where different configurations take roughly

the same time to evaluate, this will probably lead to good utilisation of the available hardware, so this approach is well-motivated and a good fit.

Consider the case of optimising a deep neural network, where the BO algorithm is trying to identify the number of units in each layer of the network, as well as other hyperparameters like learning rate, momentum and batch size. It is clear that different values for the hyperparameters can lead to significantly different runtimes – training a network with 10 units per layer will take less time than training a network with 1000 units per layer. Alternatively, consider the case of training a random forest (Breiman, 2001), where the BO algorithm is searching for the best number of trees (amongst other hyperparameters). Again, it is clear that random forests with a smaller number of trees will complete training faster than ones with more trees.

When creating a batch of configurations, we are interested in evaluating a diverse set of configurations so as to efficiently gain the most information about f. This means that a batch could contain very different configurations with very different evaluation runtimes, which would lead to inefficient utilisation of our compute resources. Since we wait for all of the configurations in the batch to complete before choosing new jobs, the slowest configuration in the batch will dictate the runtime of the batch, leaving the other workers idle until it has completed its task. In order to improve the utilisation of parallel computing resources, we can run the batch evaluation *asynchronously*: we start with a batch of k tasks, and choose new tasks as soon as c workers ($c < k$) complete their jobs – we do not wait for all jobs in the batch to complete. It is obvious that asynchronous evaluation has a clear advantage over synchronous batch BO in terms of the number of jobs evaluated in a fixed period of time (Kandasamy et al., 2018); see Figure 4.1 for an illustration. It is unclear whether asynchronous evaluation translates into a better optimisation performance over time. Asynchronous BO may also lose out in terms of sample efficiency, as an asynchronous method takes decisions about the next configuration(s) to evaluate with fewer data than its synchronous counterpart at each stage of the optimisation. It is not clear then, whether asynchronous BO is competitive, or preferable, to synchronous BO. We investigate this empirically in this chapter.

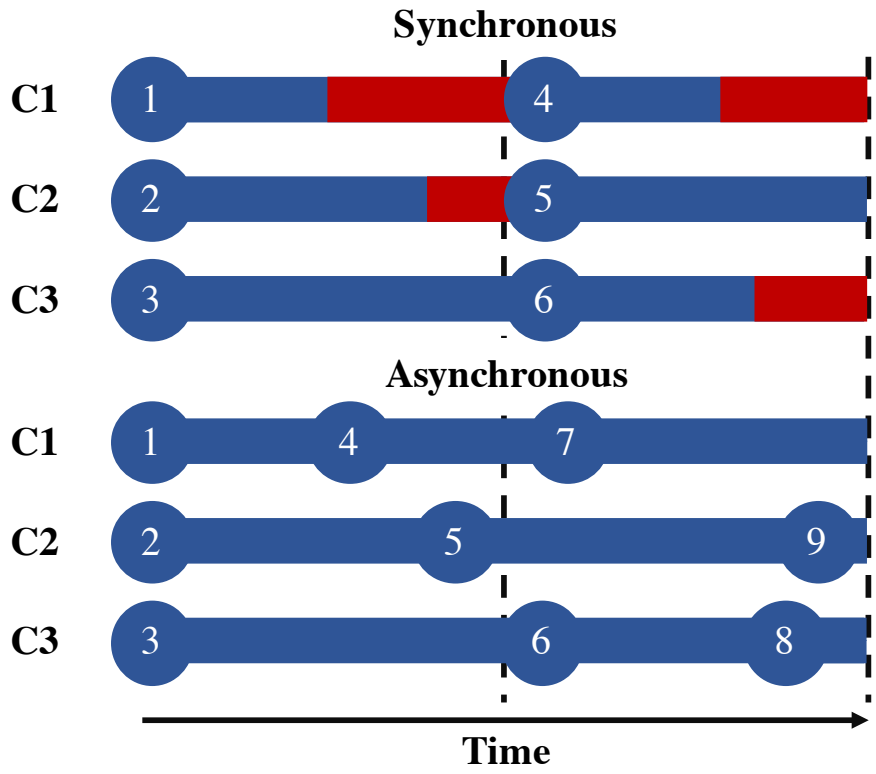


Figure 4.1: Illustration showing the difference between synchronous and asynchronous batch BO in the case of $k = 3$ parallel workers and $c = 1$. A blue bar indicates that a worker is evaluating its assigned task and a red bar indicates a worker waiting for its next job. It is clear that asynchronous batch BO, which makes better use of the computing resources, can complete a greater number of evaluations than its synchronous counterpart within the same duration.

4.3.1 Batch selection

Many batch BO algorithms convert the selection of $\mathcal{B}_s$ into a sequential procedure, adding one point at a time to the batch. At the sth BO step, the choice of batch point $\mathbf{x}_j$ ($j \in \{1, 2, \ldots, k\}$) should then not only take into account our current knowledge of f, but also marginalise over possible function values at the configurations $\{\mathbf{x}_i\}_{i=1}^{j-1}$ that we have chosen so far for the batch:

$$\mathbf{x}_j = \arg\max_{\mathbf{x} \in \mathcal{X}} \int \alpha(\mathbf{x} \mid \mathcal{D}_s, \mathcal{D}_{j-1}) \prod_{i=1}^{j-1} p(y_i \mid \mathbf{x}_i, \mathcal{D}_s, \mathcal{D}_{i-1}) \mathrm{d}y_i, \tag{4.2}$$

where $\mathcal{D}_s$ are the data we have gathered so far and $\mathcal{D}_{j-1} = \{\mathbf{x}_i, y_i\}_{i=1}^{j-1}$ and $\mathcal{D}_0 = \varnothing$ (Gonzalez et al., 2016).

In asynchronous BO, after a desired number of workers $c < k$ complete their tasks, we assign new tasks to them without waiting for the remaining $b = (k-c)$ busy workers to complete their tasks. Thus, the general design for selecting the next query point marginalises over the likely function values both at configurations currently under evaluation by busy workers, as well as the already-selected points in the batch:

$$\begin{aligned}\mathbf{x}_j = \arg\max_{\mathbf{x}\in\mathcal{X}} \iint \alpha(\mathbf{x} \mid \mathcal{D}_{s\_}, \mathcal{D}_b, \mathcal{D}_{j-1}) \times \\ \prod_{i=1}^{j-1} p(y_i \mid \mathbf{x}_i, \mathcal{D}_{s\_}, \mathcal{D}_b, \mathcal{D}_{i-1}) \times \\ \prod_{l=1}^{b} p(y_l \mid \mathbf{x}_l, \mathcal{D}_{s\_}) dy_i dy_l, \end{aligned} \tag{4.3}$$

where $j \in \{1, \ldots, c\}$ and $\mathcal{D}_b = \{\mathbf{x}_l, y_l\}_{l=1}^{b}$ are the configurations and function values of the busy configurations selected at previous BO iterations. $\mathcal{D}_{s\_}$ are the data available at the point of constructing the asynchronous batch. In general, $\mathcal{D}_{s\_}$ contains fewer data than the $\mathcal{D}_s$ that would be used to select the equivalent batch of evaluations in the synchronous setting. Figure 4.1 shows the case of $c = 1$ and thus $b = k - 1 = 2$.

In a given period of time, asynchronous batch BO is able to process a greater number of evaluations than the synchronous approach, so asynchronous BO offers clear gains in resource utilisation. However, Kandasamy et al. (2018) claim that the asynchronous setting may not lead to better performance when measured by the number of evaluations. The authors point out that a new evaluation in a sequentially-selected synchronous batch will be selected with at most $k-1$ evaluations "missing" (that is, with knowledge of their configurations $\mathbf{x}$ but absent the knowledge of their values y), corresponding to the previously-selected points in the current batch (i.e. $j - 1 \leq k - 1$ in Equation (4.2)). Evaluations in the asynchronous case are always chosen with $k - 1$ "missing" values.

However, to our knowledge, there exists little empirical investigation of the performance difference between synchronous and asynchronous batch methods. We conducted this comparison on a large set of benchmark test functions and found that asynchronous batch BO can be as good as synchronous batch BO

for different batch selection methods. Additionally, for the penalisation-based methods we propose, asynchronous operation often outperforms the synchronous setting, particularly as the batch size increases. We will discuss this interesting empirical observation in Section 4.6.2.

4.3.2 An aside about parallel BO algorithms

It is useful to think a little more closely about $\mathcal{B}_s$. What makes a collection of configurations a good batch? A sequential BO algorithm is often straightforward to understand, since it is often derived using an explicit one-step regret measure that is being minimised. This understanding becomes less clear when we start thinking about a batch of points.

Our aims in parallel BO are not different from sequential BO, in that we wish to identify configurations that allow us to learn the most about the configuration of the optimal configuration $\mathbf{x}^*$. We still wish to balance exploration and exploitation behaviours of the algorithm so that we have good coverage of the search space, as well as an accurate understanding of the likely values of f in promising regions. The issue we face, though, is that acquisition functions are one-step criteria. Maximising the acquisition function results in us finding a single optimal configuration.

In an ideal scenario, we would want to have an acquisition function $\alpha_{\text{batch}}(\mathcal{B})$ that takes in a batch $\mathcal{B}$ and returns a score for the utility of that batch. This would allow us to optimise the configurations inside the batch collectively and identify the optimal batch

$$\mathcal{B}_s = \arg\max_{\mathcal{B}} \alpha_{\text{batch}}(\mathcal{B}). \tag{4.4}$$

Such an acquisition function, that is also scalable, does not exist yet as far as we are aware. Instead, parallel batch BO methods make use of the same acquisition functions as used in sequential BO, and introduce heuristics or other intuitions to allow us to identify multiple configurations:

$$\mathbf{x}_j = \arg\max_{\mathbf{x}} \alpha_j(\mathbf{x}|\{\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}\}), \tag{4.5}$$

where $j \in \{1, \ldots, k\}$ and we call the "batch" acquisition function for the j-th point α_j.

In these cases, the first point added to the batch is the same one that would have been selected by a sequential BO algorithm. It is not clear what trade-offs, if any, we are making by approximating the "ideal" procedure (Equation (4.4)) by using an iterative one (Equation (4.5)). Rather than questioning the relevance of batch BO, since those methods perform very well in practice, we instead wish to highlight the open question of how much better a batch acquisition function (as in Equation (4.4)) could perform in practice, once a scalable variant is developed.

On a related note, it is useful to differentiate between non-myopic and parallel BO, as they may seem to be closely related. In both cases we simulate future actions taken by the BO algorithm and use that information to inform the selected configuration(s). The key difference between the two methods is that parallel BO methods aim to create a batch of diverse points, whereas the target of a non-myopic, or multi-step lookahead, BO procedure is to select a configuration that takes into account explicitly the remaining budget and the possible future evaluations the algorithm may encounter.

4.4 Penalisation-based asynchronous BO

As discussed in Section 4.2, the existing asynchronous BO methods suffer drawbacks such as the prohibitively high cost of repeatedly updating GP surrogates when selecting batch points (Ginsbourger et al., 2011) or the risk of redundant sampling at or near a busy configuration in the batch (Kandasamy et al., 2018). In view of these limitations, we propose a penalisation-based asynchronous method which encourages sampling diversity among the points in the batch as well as eliminating the risk of repeated samples in the same batch. Our proposed method remains computationally efficient, and thus scales well to large batch sizes. We list the properties of our "ideal" penalisation method in Section 4.4.1 below.

Inspired by the Local Penalisation approach (LP) in synchronous BO (Gonzalez et al., 2016), see Equation (4.1), we approximate Equation (4.3) for the

case of $c = 1$ as:

$$\mathbf{x}_j = \arg\max_{\mathbf{x}\in\mathcal{X}} \left\{ \alpha(\mathbf{x} \mid \mathcal{D}_{s\_}) \prod_{i=1}^{k-1} \phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{s\_}) \right\}, \tag{4.6}$$

where $\phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{s\_})$ is the penaliser function centred at the busy configurations $\{\mathbf{x}_i\}_{i=1}^{k-1}$. In the following subsections, we design effective penaliser functions by harnessing the Lipschitz properties of the function and its GP posterior. To simplify notation, we denote $\phi(\mathbf{x} \mid \mathbf{x}_i, \mathcal{D}_{s\_})$ as $\phi(\mathbf{x} \mid \mathbf{x}_i)$ and $\alpha(\mathbf{x} \mid \mathcal{D}_{s\_})$ as $\alpha(\mathbf{x})$ in the remainder of the section.

4.4.1 Hard Local Penaliser

Assume the unknown objective function is Lipschitz continuous with constant L. Lipschitz continuity is a relatively weak assumption, as this assumes that the function is smooth, and hence an optimum can be found by reasoning about the function's shape between acquired data. We also assume that the function has a global minimum value $f(\mathbf{x}^*) = M$, and that $\mathbf{x}_j$ is a busy task,

$$|f(\mathbf{x}_j) - M| \leq L\|\mathbf{x}_j - \mathbf{x}^*\|. \tag{4.7}$$

This implies $\mathbf{x}^*$ cannot lie within the spherical region centred on $\mathbf{x}_j$ with radius $r_j = \frac{f(\mathbf{x}_j)-M}{L}$:

$$\mathbb{S}(\mathbf{x}_j, r_j) = \{\mathbf{x} \in \mathcal{X} : \|\mathbf{x} - \mathbf{x}_j\| \leq r_j\}. \tag{4.8}$$

If $\mathbf{x}_j$ is still under evaluation by a worker, there is no need for any further selections inside $\mathbb{S}(\mathbf{x}_j, r_j)$.

Given that $f(\mathbf{x}_j) \sim \mathcal{N}(\mu(\mathbf{x}_j), \sigma^2(\mathbf{x}_j))$ and thus $\mathbb{E}(r_j) = \frac{|\mu(\mathbf{x}_j)-M|}{L}$, applying Hoeffding's inequality for all $\epsilon > 0$ (Jalali et al., 2013) gives

$$P(r_j > \mathbb{E}(r_j) + \epsilon) \leq \exp\left(-\frac{2\epsilon^2 L^2}{\sigma(\mathbf{x}_j)^2}\right). \tag{4.9}$$

Substituting in $\epsilon = \frac{1.5\sigma(\mathbf{x}_j)}{L}$

$$P\left(r_j > \mathbb{E}(r_j) + \frac{1.5\sigma(\mathbf{x}_j)}{L}\right) \leq \exp\left(-\frac{2L^2}{\sigma(\mathbf{x}_j)^2}\left(\frac{1.5\sigma(\mathbf{x}_j)}{L}\right)^2\right) \tag{4.10}$$

$$= \exp(-2 \times 1.5^2) \tag{4.11}$$

$$\approx 0.0111 \tag{4.12}$$

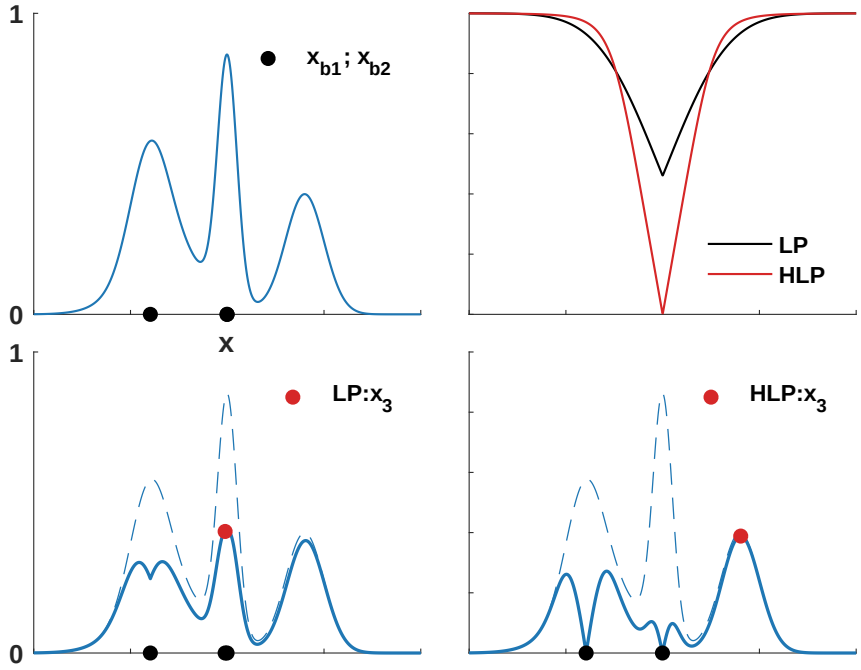


Figure 4.2: Illustration of asynchronous batch selection by naïve LP and HLP. The top left plot shows the acquisition function $\alpha(x)$ and the configurations (i.e. $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ denoted in black dots) under evaluation by busy workers. The top right plot shows the shapes of two penalisers at the busy configuration x_{b1}. Their respective penalisation effects on $\alpha(x)$ at $\mathbf{x}_{b1}$ and $\mathbf{x}_{b2}$ as well as the new batch point x_3 to be assigned to the available worker are shown in the subplots that follow, LP on the left and HLP on the right.

so therefore

$$P\left(r_j \leq \mathbb{E}(r_j) + \frac{1.5\sigma(\mathbf{x}_j)}{L}\right) \geq 0.99, \tag{4.13}$$

which implies there is a high probability (around 99%) that $r_j \leq \frac{|\mu(\mathbf{x}_j)-M|}{L} + 1.5\frac{\sigma(\mathbf{x}_j)}{L}$.

The penalisation function $\phi(\mathbf{x} \mid \mathbf{x}_j)$ should incorporate this belief to guide the selection of the next asynchronous batch point by reducing the value of the acquisition function at configurations $\{\mathbf{x} \in \mathbb{S}(\mathbf{x}_j, r_j)\}$. A valid penaliser should possess the several properties:

- the size of the penalisation region reduces as the expected function value at $\mathbf{x}_j$ gets close to the global minimum (i.e. small $|\mu(\mathbf{x}_j) - M|$) (Gonzalez et al., 2016);
- the penalisation region shrinks as L increases (Gonzalez et al., 2016);

- the extent of penalisation on $\alpha(\mathbf{x})$ increases as $\mathbf{x}$ gets closer to $\mathbf{x}_j$ with $\alpha(\mathbf{x}_j) = 0$ if $\alpha(\mathbf{x}) \geq 0$ for all $\mathbf{x} \in \mathcal{X}$.

The Local Penaliser (LP) in Gonzalez et al. (2016) fulfils the first two properties but not the final one which we believe is crucial. Thus, directly using it for the asynchronous case makes the algorithm vulnerable to redundant sampling as illustrated in Figure 4.2. In view of this limitation, we propose a simple yet effective Hard Local Penaliser (HLP) which satisfies all three conditions

$$\phi(\mathbf{x} \mid \mathbf{x}_j) = \min \left\{ \frac{\|\mathbf{x} - \mathbf{x}_j\|}{\mathbb{E}(r_j) + \gamma \frac{\sigma(\mathbf{x}_j)}{L}}, 1 \right\}, \tag{4.14}$$

where γ is a constant.

The above expression can be made differentiable by the approximation:

$$\hat{\phi}(\mathbf{x} \mid \mathbf{x}_j) = \left[\left(\frac{\|\mathbf{x} - \mathbf{x}_j\|}{\mathbb{E}(r_j) + \gamma \frac{\sigma(\mathbf{x}_j)}{L}} \right)^p + 1 \right]^{1/p}, \tag{4.15}$$

with $\hat{\phi}(\mathbf{x} \mid \mathbf{x}_j) \to \phi(\mathbf{x} \mid \mathbf{x}_j)$ as $p \to -\infty$. Note that p tends to negative infinity, so that the central area of the penaliser follows the desired shape and it tends to 1 as the radius grows.

In addition, the global optimum M is unknown in practice and is usually approximated by the best function value observed $\hat{M} = \min\{f(\mathbf{x}_i)\}_i^n$ (Gonzalez et al., 2016). This approximation tends to lead to underestimation of $\mu(\mathbf{x}_j) - M$ and thus $\mathbb{E}(r_j)$ (by over-estimating $\hat{M} > M$), reducing the extent of the penalisation at $\mathbf{x}_j$ and in the region nearby. HLP mitigates this effect by penalising significantly harder than the penaliser proposed by Gonzalez et al. (2016), and maximally at $\mathbf{x}_j$ ($\alpha(\mathbf{x}_j) = 0$). Thus, our method is less affected by over-estimation of the global minimum $\hat{M} > M$.

4.4.2 Locally-estimated Lipschitz constants

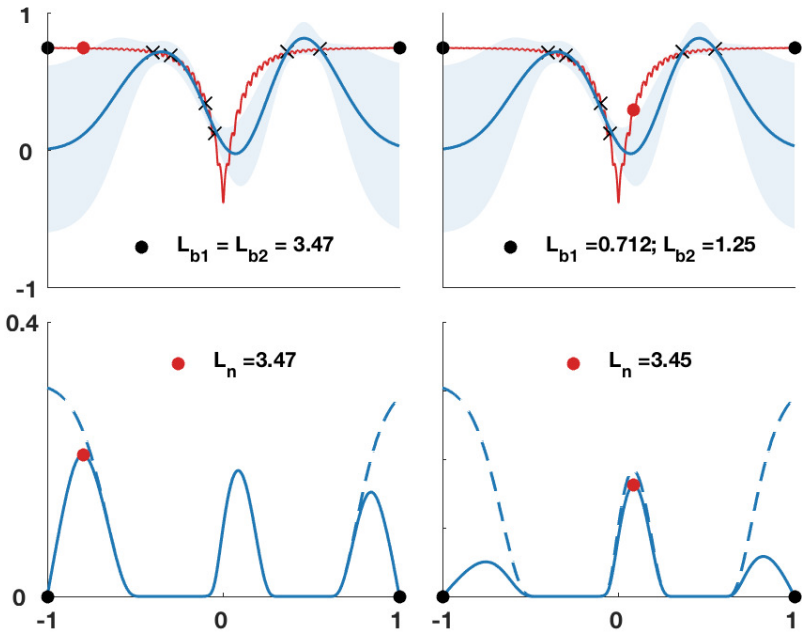
In BO, the global Lipschitz constant L of the objective function is unknown. Assuming the true objective function f is a draw from its GP surrogate model, we can approximate L with $\hat{L} = \max_{\mathbf{x} \in \mathcal{X}} \|\mu_\nabla(\mathbf{x})\|$ where $\mu_\nabla(\mathbf{x})$ is the posterior mean of the derivative GP (Gonzalez et al., 2016). However, using the estimated

global Lipschitz constant $\hat{L}$ to design the shape of the penalisers at all busy configurations in the batch may not be optimal. Consider the case where a point in an unexplored region is still under evaluation. If $\hat{L}$ is large, then the penaliser's radius will be small and we will end up selecting multiple points in the same unexplored region, which is undesirable.

Therefore, we propose to use a separate Lipschitz constant, which is locally estimated, for each busy configuration. Here, "locality" is encoded in our choice of kernel and its hyperparameters, e.g. via the lengthscale parameter in the Matérn class of kernels. The use of local Lipschitz constants will enhance the efficiency of exploration because they allow the penaliser to create larger exclusion zones in areas in which we are very uncertain (the surrogate model is near its prior or has low curvature) and smaller penalisation zones in interesting, high-variability, areas. This insight is also corroborated in Blaas et al. (2019), who used this fact to develop a fast optimisation algorithm for 1D functions.

We demonstrate the different effects of using approximate global and local Lipschitz constants with a qualitative example in Figure 4.3. In Figure 4.3a, the estimated global Lipschitz constant is used for penalisation at both busy configurations $\mathbf{x}_{b1} = -1$ and $\mathbf{x}_{b2} = 1$ (denoted as black dots). The relatively large value of the global Lipschitz constant ($\hat{L}_{b1} = \hat{L}_{b2} = \hat{L} = 3.47$) due to the high curvature of the surrogate in the central region leads to a small penalisation zone around the two busy configurations at the boundary. This causes the algorithm to miss the informative region in the centre and instead revisit the region near $\mathbf{x}_{b1}$ to choose the new point in the asynchronous batch. On the other hand, in Figure 4.3b, the use of a locally estimated Lipschitz constant allows us to penalise a larger zone around points where the surrogate is relatively flat ($\hat{L}_{b1} = 0.712$ for $\mathbf{x}_{b1}$), while still penalising smaller regions where there is higher variability ($\hat{L}_3 = 3.45$ at $\mathbf{x}_3$).

In our experiments we used a Matérn-52 kernel and defined the local region for evaluating the Lipschitz constant for a batch point $\mathbf{x}_j$ to be a hypercube centred on $\mathbf{x}_j$ with the length of the each side equal to twice the lengthscale corresponding to that input dimension, as illustrated in Figure 4.4.



(a) Penalisation with global L **(b)** Penalisation with local L

Figure 4.3: Different penalisation effects on $\alpha(\mathbf{x})$ of using a single global Lipschitz constant compared to local Lipschitz constants. The top plots in both (a) and (b) show the true objective function (red line), six observed points (black crosses), the GP posterior mean (blue line) and variance (blue shade), the two busy configurations (black dots) and the next query point (red dot) selected by using the HLP with global and local Lipschitz constants respectively. The dashed blue line is the original (non-penalised) acquisition function. The plots in (a) show the penalisation effect on busy configurations using the same global Lipschitz constant while those in (b) show the effect of using local Lipschitz constants. It is clear that penalising the busy configurations based on local Lipschitz constants allows the algorithm to capture the informative peak at the central region while selection based on the single global Lipschitz constant leads us to revisit the flat region near the boundary due to insufficient penalisation at $\mathbf{x}_1$.

In our empirical evaluation we differentiate between PLAyBOOK-L, which uses a naïve Local penaliser, PLAyBOOK-H, that uses the HLP penaliser, as well as their variations with locally estimated Lipschitz constants, PLAyBOOK-LL and PLAyBOOK-HL.

4.5 Practical considerations

There are some practical details that are worth discussing before we move to the experimental evaluation.

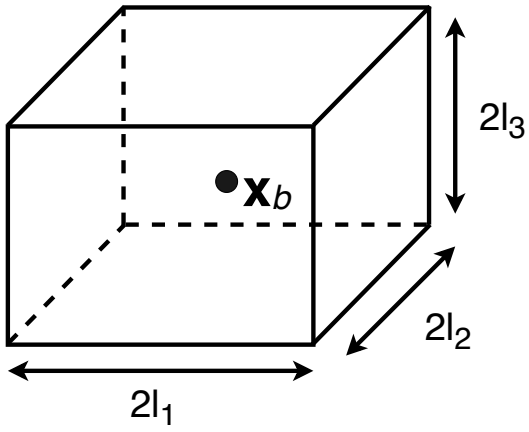


Figure 4.4: We estimate the local Lipschitz constant within a hypercube centred around the batch point $\mathbf{x}_b$, with each side's length being twice the kernel lengthscale corresponding to that dimension. Here shown for $d = 3$.

Multimodality of $\alpha(\mathbf{x})$

One of the advantages of penalisation-based batch methods is that we avoid performing costly re-computations of the inverse of the surrogate GP's covariance matrix, as is the case in the heuristics-based methods in Ginsbourger et al. (2010). Instead we leverage the given acquisition function together with the penaliser function, both of which are cheap to compute. An important practical requirement is that the acquisition function should have multiple peaks spread out in the search space. This is so that successive penalisations can uncover other promising areas, which we define as regions with a high value for $\alpha(\mathbf{x})$.

From practical experience, the popular EI acquisition function does not reliably provide a multi-modal acquisition surface, but rather quickly turns into a single peak (see e.g. Figure 2.5). Applying a penalisation-based batch method to EI is still possible, but its usefulness is then unclear, because after the first few penalisations we would then be choosing configurations that have roughly equal values of $\alpha(\mathbf{x})$. In such cases then the selection would be driven by the randomness in our optimisation approach for $\alpha(\mathbf{x})$, rather than any inherent utility of the selected configurations over others.

We found that using UCB as the acquisition function provided a surface with multiple peaks, as required, but there exist alternative choices in the literature.

ScaledEI is a variant of EI that was proposed in Noè and Husmeier (2018), in which the uncertainty in the estimate of EI is integrated into the criterion, allowing more variability across the search space, and hence leading to a more multi-modal surface. Another extension of EI was proposed in Berk et al. (2019), in which the authors adapt the acquisition function to create a more multi-modal surface for early iterations.

Requiring $\alpha(\mathbf{x}) > 0$

Figure 4.2 shows that the penaliser's value is in the range $\phi(\mathbf{x}) \in [0, 1]$, and it is applied to the acquisition function via multiplying the penaliser with $\alpha(\mathbf{x})$, as in Equation 4.6. This means that we require the acquisition function to be positive across the search space, so that multiplication with $\phi(\mathbf{x})$ attenuates the acquisition value.

If the acquisition function at a selected configuration was negative, then we would have $\alpha(\mathbf{x})\phi(\mathbf{x}) > \alpha(\mathbf{x})$ at that configuration, which would *increase* rather than decrease the value of the acquisition function after penalisation. This would cause subsequent optimisations of the "penalised" acquisition function to select the same configurations repeatedly. This is particularly relevant to us, since we chose to use UCB as our acquisition function, which provides us with no positivity guarantees.

We tried two different approaches to enforce positivity of $\alpha(\mathbf{x})$: offsetting $\alpha(\mathbf{x})$ and using a "softplus" transform. In order to find the correct offset, we optimised $\alpha(\mathbf{x})$ in order to find its *minimum*. We then replaced the acquisition function for the remainder of the algorithm by

$$\alpha'(\mathbf{x}) = \alpha(\mathbf{x}) - \min(\alpha(\mathbf{x})). \tag{4.16}$$

Even though $\alpha(\mathbf{x})$ is a cheap function to evaluate, the search space is often quite large if we are optimising multiple parameters of a model, so we want to reduce computation effort for selecting batch points where possible. In this case, it is not critical to identify the global minimum of $\alpha(\mathbf{x})$. Finding a value that makes sure that, after offsetting $\alpha(\mathbf{x})$, the peaks are positive is sufficient, so instead of running

an exhaustive optimisation, we evaluated the acquisition function on a large random grid and selected the minimum value of that grid to act as $\min(\alpha(\mathbf{x}))$.

The "softplus" transform was suggested in Gonzalez et al. (2016) and suggests transforming the acquisition function using

$$\alpha'(\mathbf{x}) = \log(1 + \exp(\alpha(\mathbf{x}))). \tag{4.17}$$

This transform has a negligible effect for large positive values of $\alpha(\mathbf{x}))$, and squeezes all negative values of $\alpha(\mathbf{x}))$ into $[0, 1]$. Empirically we saw no significant difference in performance between the two, but since offsetting the acquisition function requires an extra optimisation step, we suggest using the "softplus" transform to save on computation effort.

Choosing M when its true value is known

Both LP and HLP require an estimate of the true global minimum of f for sizing the penalisation region correctly. One of the motivations for us to develop the HLP penaliser was LP's over-reliance on the value of M and its insufficient penalisation resulting from estimating M incorrectly.

There arises a natural question: what do we do when the true value of the global minimum is known? This is the case for example when the quantity we are minimising is an error metric of a model, so $\min_{\mathbf{x}}(f(\mathbf{x})) = 0$. In our evaluation below we show results for using LP and HLP with estimates of M, as when we tested all of the PLAyBOOK methods with estimated M as well as the actual value of M, we found that using the latter did not provide a significant difference.

4.6 Experimental evaluation

We begin our empirical investigations by performing a head-to-head comparison of synchronous and asynchronous BO methods, to test the intuitions described in Section 4.3. We specifically look at optimisation performance for asynchronous and synchronous variants of the parallel BO methods measured over time and

number of evaluations, and we show empirically that asynchronous is preferable over synchronous BO on both counts.

We then evaluate our proposed asynchronous methods (PLAyBOOK-L, PLAyBOOK-H, PLAyBOOK-LL and PLAyBOOK-HL) on a number of benchmark test functions as well as a real-world expensive optimisation task. We compare against the state-of-the-art asynchronous BO methods, Thompson sampling (TS) (Kandasamy et al., 2018), as well as the Kriging Believer heuristic method (KB) (Ginsbourger et al., 2010) applied asynchronously.

For all the benchmark functions, we measure the log of the simple regret R, which is the difference between the true minimum value $f(\mathbf{x}^*)$ and the best value found by the BO method:

$$\log(R) = \log \left| f(\mathbf{x}^*) - \min_{i=1,\dots,n} f(\mathbf{x}_i) \right|. \tag{4.18}$$

This is a popular performance metric used in the BO literature, as it captures the ability of the algorithm to identify locations with functional values close to the true optimum. We are not necessarily interested in identifying a specific optimum, as there may exist more than one, or we may only know the value of the true optimum and not its location, so the distance in the input space is not relevant in our empirical evaluation.

4.6.1 Implementation details

To ensure a fair comparison, we implemented all methods in Python using the same packages[1].

In all experiments, we used a zero-mean Gaussian process surrogate model with a Matérn-52 kernel with ARD. We optimised the kernel and likelihood hyperparameters by maximising the log marginal likelihood. For the benchmark test functions, we fixed the noise variance to $\sigma^2 = 10^{-6}$ and started with $3d$ random initial configurations. Each experiment was repeated with 30 different random initialisations and the input domains for all experiments were scaled to $[-1, 1]^d$.

[1] Implementation available at https://github.com/a5a/asynchronous-BO

All methods except TS used UCB as the acquisition function $\alpha(\mathbf{x})$. For our PLAyBOOK-H and PLAyBOOK-HL, we choose $\gamma = 1$ and $p = -5$ in Equation (4.15). Larger values of p did not provide much benefit in terms of the shape of the penaliser. For TS, we use 10,000 sample points for each batch point selection. For the other methods, we evaluate $\alpha(\mathbf{x})$ at 3,000 random configurations and then choose the best location after locally optimising the best 5 samples for a small number of local optimisation steps.

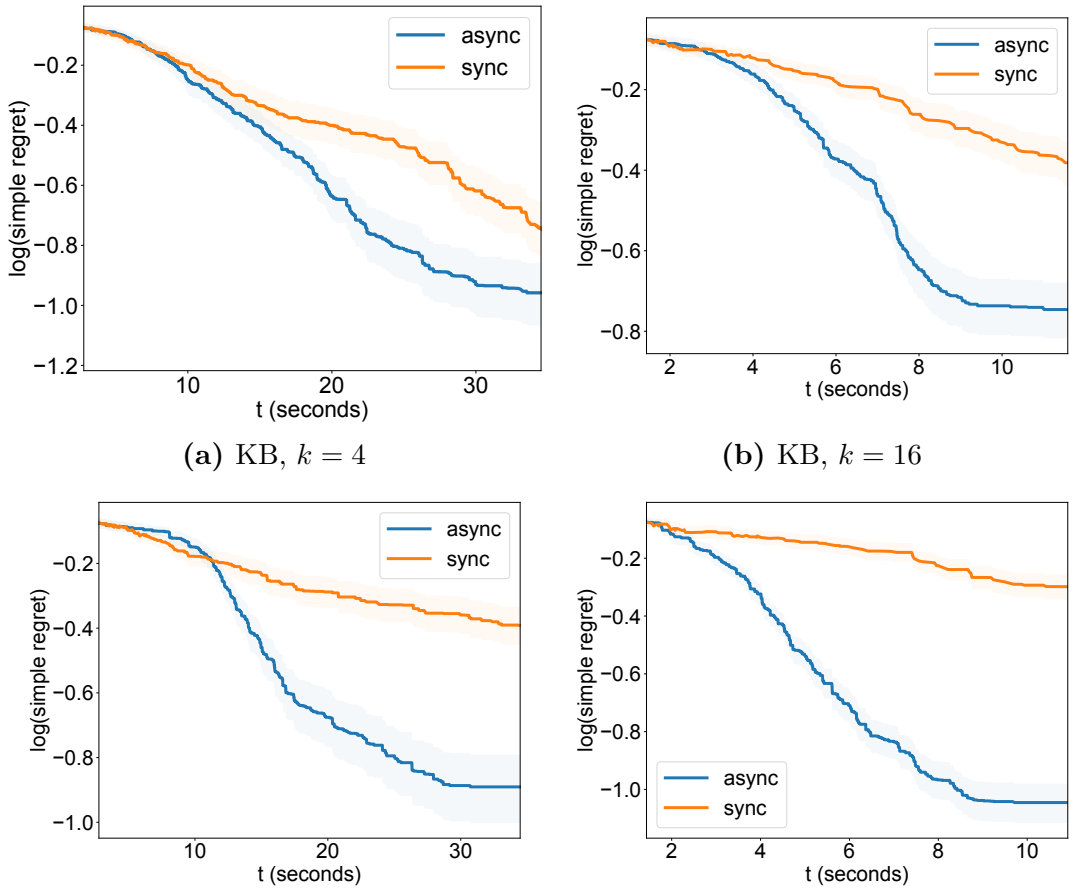
We evaluate the performance of the different batch BO strategies using popular global optimisation test functions[2]:

- The tasks `mat-2` and `mat-6` refer to functions drawn from a Gaussian process with Matérn-52 kernel in $\mathbb{R}^2$ and $\mathbb{R}^6$ respectively.
- The global optimisation tasks that we considered are the Ackley function defined on $\mathbb{R}^5$ and $\mathbb{R}^{10}$ (`ack-5` and `ack-10`), the Michalewicz function defined on $\mathbb{R}^5$ and $\mathbb{R}^{10}$ (`mic-5` and `mic-10`) and the Eggholder function in $\mathbb{R}^2$ (`egg-2`). The mathematical expressions for these test problems are shown in Table 3.1.
- We also selected a robot pushing simulation experiment, `nrobot-4`, which was first explored in a BO context by Wang and Jegelka (2017). Here the task is to learn the correct pushing action to minimise the distance of the robot to a goal. The problem has 4 inputs: the robot's location (r_x, r_y), the angle of the pushing force r_θ and the pushing duration t_r. We used the domain suggested by Wang and Jegelka (2017), namely $r_x \in [-5, 5]$, $r_y \in [-5, 5]$, $r_\theta \in [0, 2\pi]$ and $t_r \in [1, 30]$.

4.6.2 Synchronous vs asynchronous BO

In this section we address the question of choosing between asynchronous and synchronous BO. In order to investigate their relative merits, we compared asynchronous and synchronous BO methods' performance as a function of evaluation time and number of evaluations.

[2]Details for these and other challenging global optimisation test functions can be found at https://www.sfu.ca/~ssurjano/optimization.html



(a) KB, $k = 4$ **(b)** KB, $k = 16$

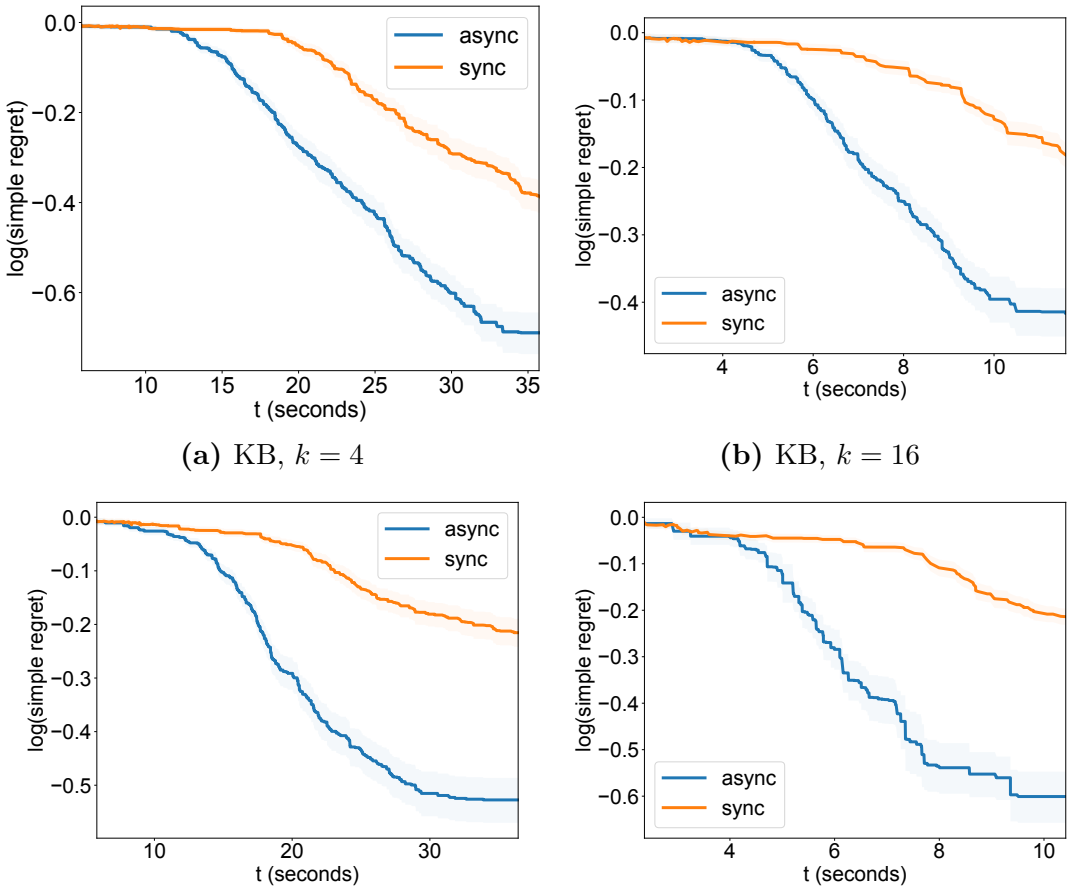
(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.5: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method in terms of *evaluation time* for $k = 4$ and $k = 16$. The mean (solid line) and standard error (shaded region) of the regret for optimising `ack-5` for 30 random initialisations are shown. Notice how the asynchronous methods outperform their synchronous counterparts in terms of evaluation time.

Evaluation time

In order to perform a time-based comparison, we needed to inject a measure of runtime for different tasks, as the test functions can be evaluated instantaneously. We followed the procedure proposed in Kandasamy et al. (2018) to sample an evaluation time for each task so as to simulate the asynchronous setting. We chose to use a half-normal distribution with scale parameter $\sigma = \sqrt{\pi/2}$, which gives us a distribution of runtime values with mean at 1.

Results on the `ack-5` task are shown in Figure 4.5 and `ack-10` in Figure 4.6. Note that each subplot should be considered separately, as we are showing the

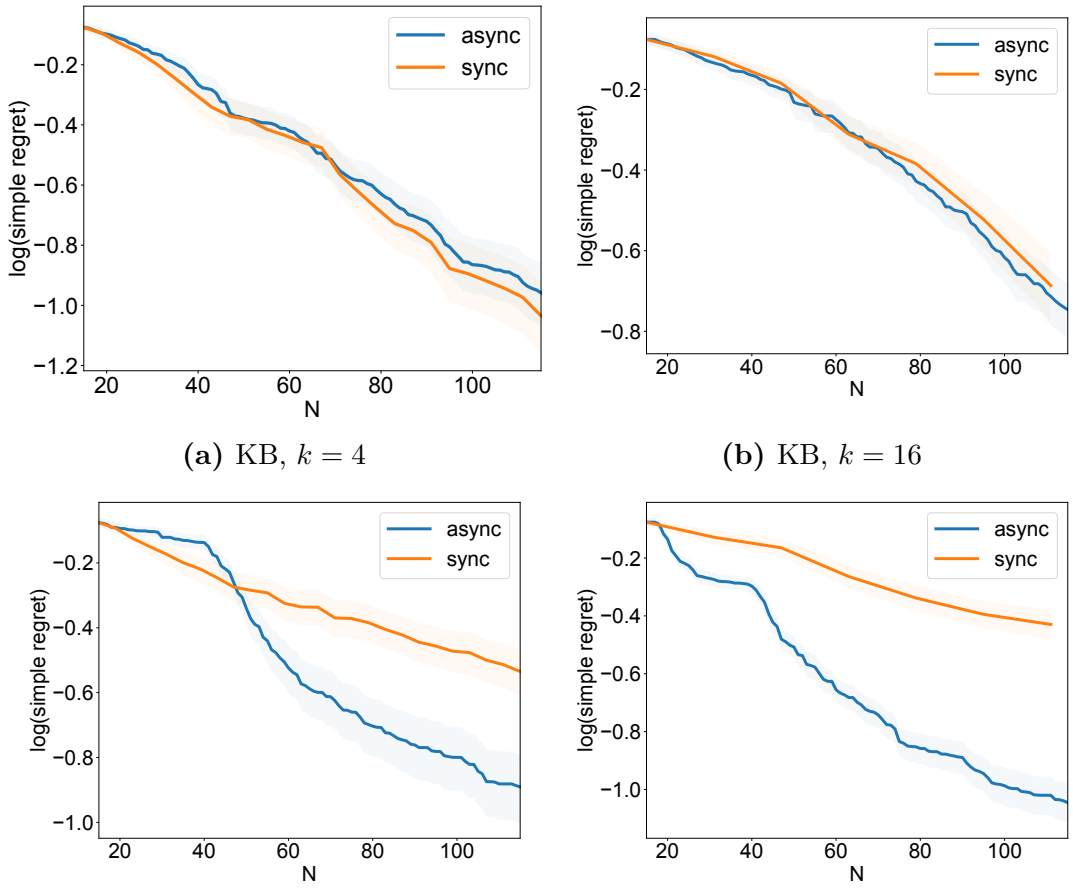


(a) KB, $k = 4$ **(b)** KB, $k = 16$

(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.6: Head-to-head comparison as in Figure 4.5 on `ack-10`.

difference between synchronous and asynchronous evaluation in a specific setting. We know that asynchronous BO has the advantage over synchronous BO in terms of utilisation of resources, as shown qualitatively in Figure 4.1, simply due to the fact that any available worker is not required to wait for all evaluations in the batch to finish before moving on. Therefore, given the same time budget, a greater number of evaluations can be performed in the asynchronous setting than in the synchronous setting, as we see in Figures 4.5 and 4.6. Our experiments show that this also translates to faster optimisation of f in terms of the (wall-clock) time spent evaluating tasks. We observed this behaviour for PLAyBOOK, as well as TS and KB. This shows that asynchronous evaluation should be preferred over synchronous evaluation, when the cost is dominated by (wall-clock) time.



(a) KB, $k = 4$ **(b)** KB, $k = 16$

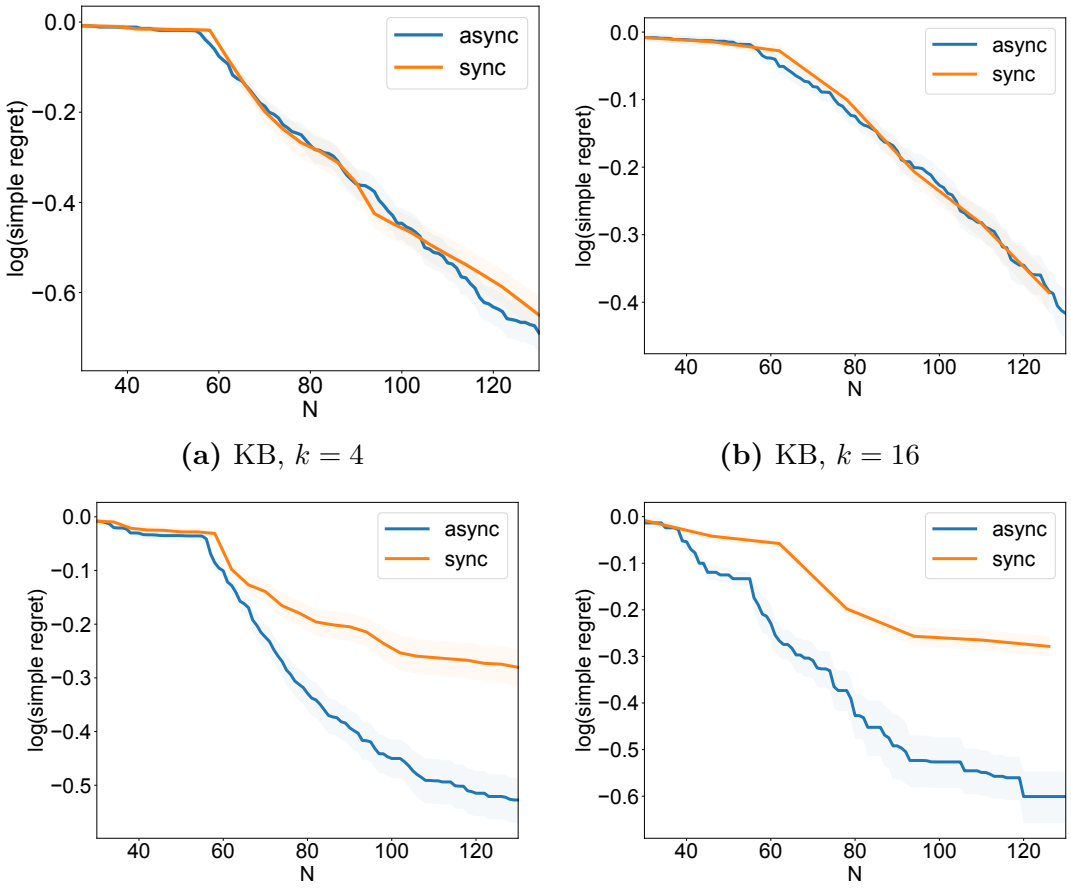
(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.7: A head-to-head comparison of synchronous (orange) vs asynchronous (blue) versions of a parallel BO method in terms of *number of samples*. The mean (solid line) and standard error (shaded region) of the regret for optimising `ack-5` for 30 random initialisations are shown. PLAyBOOK-HL outperforms its synchronous variant in terms of sample efficiency, especially as k increases, whereas asynchronous and synchronous KB are quite similar.

Number of evaluations

A more interesting question to investigate is whether asynchronous BO methods are really less data efficient than synchronous BO methods as discussed in Section 4.3. Figures 4.7 and 4.8 show the same experiments as above, this time evaluating the BO methods in terms of their sample efficiency.

An unexpected yet interesting behaviour we note is that as k increases, the PLAyBOOK methods tend to clearly outperform their respective synchronous counterparts even in terms of sample efficiency, as seen in Figures 4.5c, 4.5d and



(a) KB, $k = 4$ **(b)** KB, $k = 16$

(c) PLAyBOOK-HL, $k = 4$ **(d)** PLAyBOOK-HL, $k = 16$

Figure 4.8: Head-to-head comparison as in Figure 4.7 on `ack-10`.

Figures 4.6c, 4.6d. This observation runs counter to the guidance provided in Kandasamy et al. (2018) and such behaviour is less evident for the other two batch methods, TS and KB.

We think this observation may be explained by the difference in nature between the PLAyBOOK and TS/KB: in the case of TS we rely on stochasticity in sampling, and in KB we are re-computing the posterior variance and $\alpha(\mathbf{x})$ each time a batch point is selected. The penalisation-based methods, on the other hand, simply down-weight the acquisition function, and in the synchronous case these penalisers coincide with the high-value regions of the acquisition function. This means that unless the acquisition function has a large number of spaced-out peaks, we will quickly be left without high-utility configurations to choose new batch points from.

This explains the superior performance of asynchronous PLAyBOOK methods over their synchronous variants because they benefit from the fact that the busy configurations being penalised do not necessarily coincide with the peaks in $\alpha(\mathbf{x})$, as the surrogate used to compute $\alpha(\mathbf{x})$ is more informed than the one used to decide the configurations of the busy configurations previously. This means that points with high utilities are more likely to to be preserved.

Taking into account the fact that the asynchronous PLAyBOOK methods tend to perform at least equally well, if not significantly better, than their synchronous variants on both time and efficiency, and that the asynchronous PLAyBOOK methods gain more advantage over synchronous ones as the batch size increases, we believe that this points to the fact that penalisation-based methods are inherently better suited as asynchronous methods. Hence, for users that are running parallel BO and have selected LP, we recommend they consider running PLAyBOOK instead due to its attractive benefits.

4.6.3 Asynchronous parallel BO

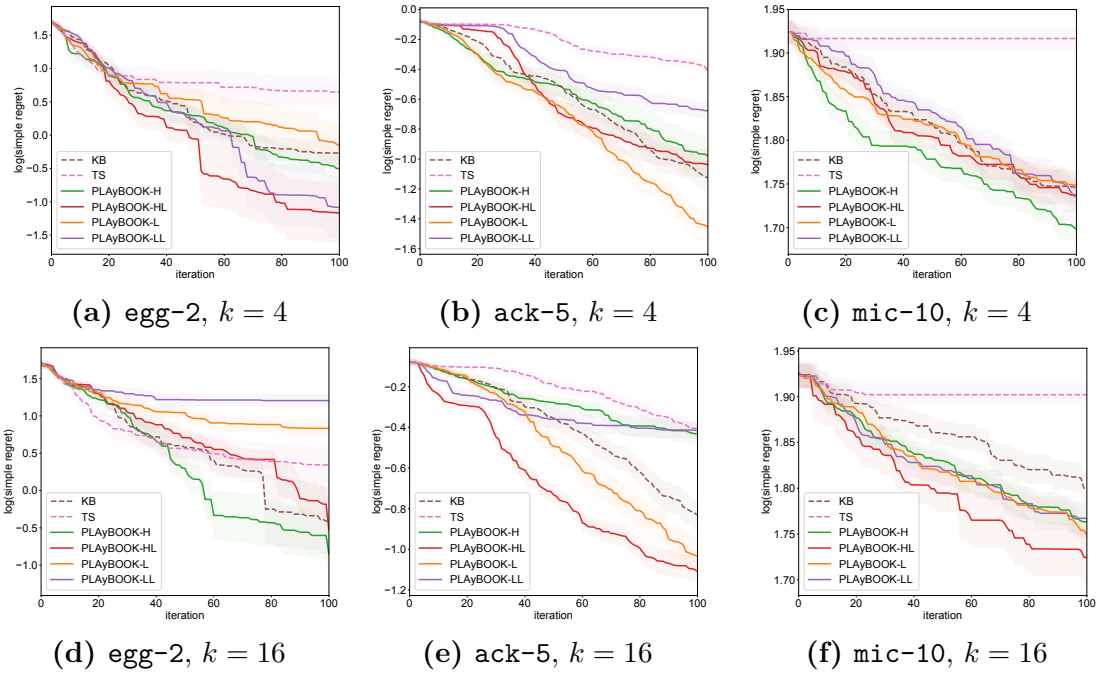
Now that we have strengthened the appeal of asynchronous BO, we turn to evaluating PLAyBOOK against existing asynchronous BO methods.

Synthetic experiments

We ran PLAyBOOK and competing asynchronous BO methods on the global optimisation test functions described in Section 4.6.1. Some results are are shown in Figure 4.9, while a comprehensive summary of the evaluation is presented in Tables 4.2-4.4.

On the global optimisation test functions we noted that in most cases PLAyBOOK outperforms the alternative asynchronous methods TS and KB. The TS algorithm performed poorly on these tasks, which we believe is caused by the fact that TS relies heavily on the surrogate's uncertainty to explore new regions.

PLAyBOOK methods show strong performance, achieving better optimisation performance than both TS and KB baselines. Our experiments are summarised in



(a) `egg-2`, $k = 4$ **(b)** `ack-5`, $k = 4$ **(c)** `mic-10`, $k = 4$

(d) `egg-2`, $k = 16$ **(e)** `ack-5`, $k = 16$ **(f)** `mic-10`, $k = 16$

Figure 4.9: The mean (solid line) and standard error (shaded region) of the regret for different asynchronous BO methods on the global optimisation test functions for 30 random initialisations is shown. We can see that our proposed PLAyBOOK methods perform competitively, especially when we start choosing larger batch sizes k.

Tables 4.2, 4.3 and 4.4, showing the mean and standard error of the optimisation methods when terminated after 50, 75 and 100 BO iterations respectively. The best method for each task is highlighted in each row in the tables. We can see that PLAyBOOK outperforms TS and KB in most cases.

For future investigations, it would be interesting to explore why the ordering of the best variant within the PLAyBOOK framework changes across tasks. Our initial hypothesis is that this may be related to how the shape of the function being optimised may be more or less amenable to optimisation with local vs global Lipschitz constants. For smaller numbers of BO iterations PLAyBOOK-HL shows strong performance across most tasks, and as the number of iterations increases, the difference between the PLAyBOOK methods reduces, while still outperforming TS and KB in most cases.

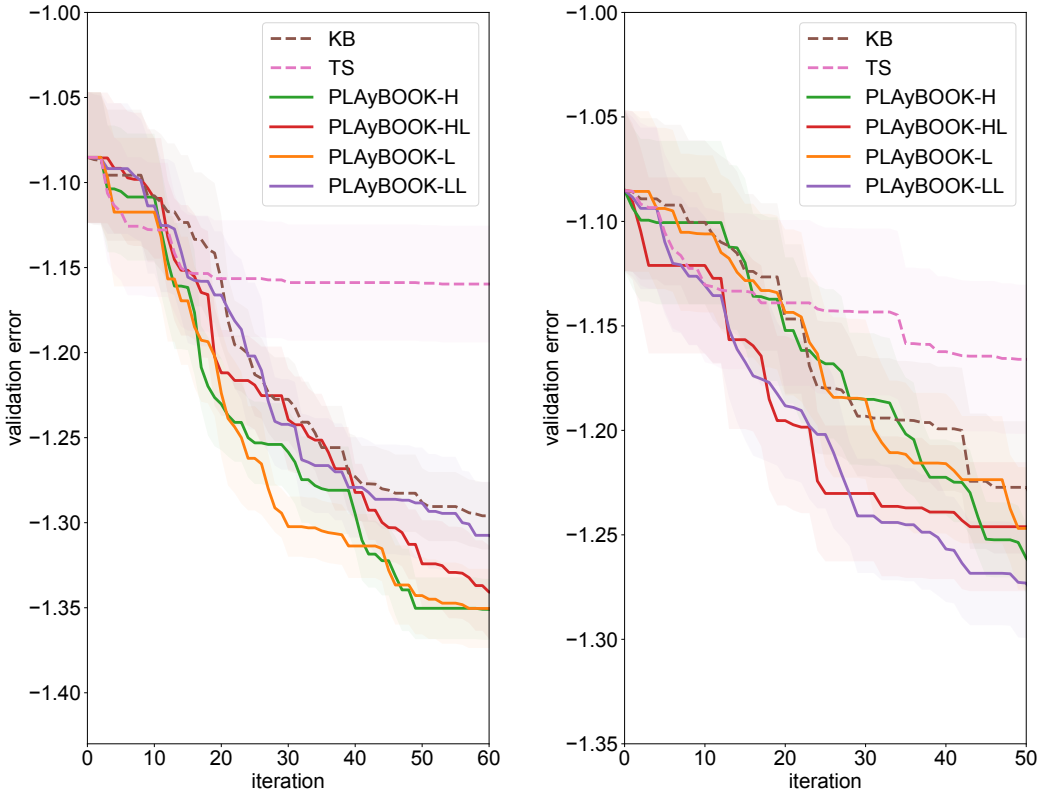
Real-world optimisation

We experimented on a real-world application of tuning the hyperparameters of a 6-layer Convolutional Neural Network (CNN)[3] for an image classification task on CIFAR10 dataset (Krizhevsky, 2009). The 9 hyperparameters that we optimise with BO are the learning rate and momentum for the stochastic gradient descent training algorithm, the batch size used and the number of filters in each of the six convolutional layers. We trained the CNN on half of the training set for 20 epochs and each function evaluation returns the validation error of the model. We tested the use of $k = 2$ and $k = 4$ parallel workers to run this real-world experiment. The results are shown in Figures 4.10a and 4.10b.

We can see that for both $k = 2$ and $k = 4$ parallel settings, all PLAyBOOK methods outperform the other asynchronous methods, TS and KB. In the case of $k = 2$ (2 parallel processors), only one busy configuration is penalised in each batch so there is little gain from using a locally estimated Lipschitz constant. However, as the batch size increases to $k = 4$, we see that methods using estimated Lipschitz constants (PLAyBOOK-LL and PLAyBOOK-HL) show faster decrease in validation error than PLAyBOOK-L and PLAyBOOK-H with PLAyBOOK-LL demonstrating the best performance.

As a final check, we took the final configurations recommended by each asynchronous BO method in the $k = 2$ and $k = 4$ settings and retrained the CNN model on the full training set of 50K images for 80 epochs. The accuracy on the test set of 10K images achieved with the best model chosen by each BO method is shown in Table 4.1. In both settings, our PLAyBOOK methods achieve superior performance with PLAyBOOK-H providing the best test accuracy when $k = 2$ and PLAyBOOK-LL doing the best when $k = 4$.

[3]We followed the implementation shown in https://blog.plon.io/tutorials/cifar-10-classification-using-keras-tutorial/



(a) $k = 2$ **(b)** $k = 4$

Figure 4.10: Asynchronous optimisation of 9 hyperparameters of a 6-layer CNN for image classification on the CIFAR10 dataset. The network is trained on half of the training set and evaluated on the second half. The objective being minimised is the classification accuracy on the validation set. PLAyBOOK outperforms both KB and TS in this expensive optimisation task.

Table 4.1: Test accuracy (%) on CIFAR-10 after training the best model chosen by various asynchronous BO methods for 80 epochs

k	TS	KB	PLAyBOOK			
			L	H	LL	HL
2	81.0	83.9	84.7	**85.2**	84.1	84.9
4	81.2	82.8	82.5	83.8	**84.2**	83.0

4.7 Conclusion

We argue for the use of asynchronous (over synchronous) Bayesian optimisation (BO), and provide supporting empirical evidence. Additionally, we developed a

k	Task	KB	TS	PLAyBOOK-L	PLAyBOOK-LL	PLAyBOOK-H	PLAyBOOK-HL
2	ack-10	-0.3 (0.0)	-0.0 (0.0)	-0.2 (0.0)	**-0.4 (0.0)**	-0.3 (0.0)	-0.3 (0.0)
	ack-5	-0.6 (0.1)	-0.2 (0.0)	-0.9 (0.1)	-0.5 (0.1)	**-0.6 (0.1)**	-0.7 (0.1)
	egg-2	0.3 (0.2)	**0.9 (0.2)**	-0.1 (0.2)	-0.0 (0.2)	-0.4 (0.4)	-0.3 (0.3)
	mat-2	0.8 (0.1)	1.0 (0.0)	0.8 (0.1)	0.8 (0.0)	**0.8 (0.0)**	0.9 (0.0)
	mat-6	1.0 (0.0)	1.1 (0.0)	0.9 (0.1)	0.9 (0.1)	**0.9 (0.0)**	0.9 (0.1)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	**1.8 (0.0)**	1.8 (0.0)
	mic-5	0.7 (0.1)	1.0 (0.0)	0.7 (0.0)	**0.7 (0.0)**	0.7 (0.1)	0.9 (0.0)
	nrobot-4	-0.7 (0.2)	**-0.5 (0.2)**	-0.9 (0.2)	-0.8 (0.2)	-0.9 (0.1)	-0.9 (0.2)
4	ack-10	**-0.3 (0.0)**	-0.0 (0.0)	-0.3 (0.0)	-0.3 (0.0)	-0.2 (0.0)	-0.3 (0.0)
	ack-5	**-0.5 (0.1)**	-0.2 (0.0)	-0.7 (0.1)	-0.4 (0.0)	-0.5 (0.1)	-0.7 (0.1)
	egg-2	**0.2 (0.2)**	0.8 (0.2)	0.5 (0.2)	0.2 (0.2)	0.3 (0.2)	-0.1 (0.2)
	mat-2	0.8 (0.1)	1.0 (0.1)	**0.9 (0.0)**	1.0 (0.1)	0.8 (0.0)	0.9 (0.0)
	mat-6	**1.1 (0.0)**	1.1 (0.0)	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)	0.9 (0.0)
	mic-10	1.8 (0.0)	**1.9 (0.0)**	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)
	mic-5	0.7 (0.0)	1.0 (0.0)	**0.8 (0.0)**	0.9 (0.0)	0.8 (0.0)	0.8 (0.1)
	nrobot-4	-0.7 (0.2)	-0.4 (0.2)	-1.0 (0.2)	-0.9 (0.2)	**-0.8 (0.1)**	-0.6 (0.1)
6	ack-10	**-0.2 (0.0)**	-0.0 (0.0)	-0.3 (0.0)	-0.2 (0.0)	-0.2 (0.0)	-0.3 (0.0)
	ack-5	-0.5 (0.1)	-0.2 (0.0)	-0.5 (0.1)	-0.3 (0.0)	**-0.3 (0.0)**	-0.5 (0.0)
	egg-2	0.4 (0.2)	**0.8 (0.2)**	0.7 (0.1)	0.8 (0.1)	0.1 (0.2)	0.3 (0.2)
	mat-2	0.8 (0.0)	**1.0 (0.0)**	1.0 (0.1)	1.1 (0.0)	0.8 (0.0)	0.9 (0.0)
	mat-6	1.0 (0.0)	1.1 (0.0)	1.0 (0.1)	1.0 (0.1)	**1.0 (0.0)**	1.0 (0.0)
	mic-10	**1.8 (0.0)**	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)
	mic-5	0.8 (0.0)	1.0 (0.0)	**0.8 (0.0)**	0.9 (0.0)	0.9 (0.0)	0.9 (0.0)
	nrobot-4	-0.6 (0.2)	-0.2 (0.2)	-0.7 (0.2)	-0.5 (0.2)	**-0.6 (0.2)**	-0.5 (0.2)
8	ack-10	**-0.2 (0.0)**	-0.0 (0.0)	-0.3 (0.0)	-0.3 (0.0)	-0.2 (0.0)	-0.4 (0.0)
	ack-5	**-0.4 (0.0)**	-0.2 (0.0)	-0.6 (0.1)	-0.3 (0.0)	-0.4 (0.1)	-0.7 (0.1)
	egg-2	**0.3 (0.2)**	0.7 (0.2)	0.6 (0.2)	0.9 (0.1)	0.4 (0.1)	0.2 (0.2)
	mat-2	**0.8 (0.0)**	0.9 (0.0)	0.8 (0.1)	1.1 (0.0)	0.9 (0.0)	0.8 (0.1)
	mat-6	1.0 (0.0)	1.1 (0.0)	**1.0 (0.0)**	1.0 (0.0)	1.0 (0.0)	1.0 (0.0)
	mic-10	**1.8 (0.0)**	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)
	mic-5	0.7 (0.1)	1.0 (0.0)	0.7 (0.1)	0.8 (0.0)	**0.7 (0.1)**	0.7 (0.1)
	nrobot-4	-0.6 (0.1)	-0.2 (0.2)	-0.9 (0.2)	-0.6 (0.2)	**-0.8 (0.2)**	-0.4 (0.1)
16	ack-10	**-0.1 (0.0)**	-0.0 (0.0)	-0.2 (0.0)	-0.2 (0.0)	-0.2 (0.0)	-0.4 (0.0)
	ack-5	**-0.4 (0.1)**	-0.2 (0.0)	-0.5 (0.1)	-0.4 (0.0)	-0.3 (0.0)	-0.7 (0.0)
	egg-2	0.6 (0.2)	**0.6 (0.2)**	1.0 (0.1)	1.2 (0.1)	0.3 (0.2)	0.7 (0.1)
	mat-2	0.8 (0.0)	0.9 (0.0)	**1.1 (0.0)**	1.1 (0.0)	0.9 (0.1)	0.9 (0.1)
	mat-6	**1.1 (0.0)**	1.1 (0.0)	1.0 (0.0)	1.1 (0.0)	1.0 (0.0)	1.0 (0.0)
	mic-10	**1.9 (0.0)**	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)
	mic-5	0.8 (0.0)	1.0 (0.0)	**0.9 (0.0)**	0.8 (0.0)	0.8 (0.0)	0.8 (0.0)
	nrobot-4	-0.4 (0.2)	-0.2 (0.2)	-0.8 (0.2)	-0.1 (0.1)	**-0.4 (0.1)**	-0.4 (0.1)

Table 4.2: Mean and standard error of the log(regret) after 50 steps of asynchronous BO over 30 random initialisations.

k	Task	KB	TS	PLAyBOOK-L	PLAyBOOK-LL	PLAyBOOK-H	PLAyBOOK-HL
2	ack-10	-0.4 (0.0)	-0.0 (0.0)	-0.5 (0.0)	**-0.6 (0.0)**	-0.6 (0.1)	-0.5 (0.0)
	ack-5	-0.9 (0.1)	-0.3 (0.0)	-1.2 (0.1)	-0.8 (0.1)	**-1.0 (0.1)**	-0.9 (0.1)
	egg-2	-0.1 (0.2)	0.9 (0.2)	-0.6 (0.2)	**-1.1 (0.4)**	-0.8 (0.4)	-0.8 (0.3)
	mat-2	0.8 (0.1)	1.0 (0.0)	0.8 (0.1)	0.8 (0.0)	**0.8 (0.0)**	0.9 (0.0)
	mat-6	1.0 (0.0)	1.2 (0.0)	0.7 (0.1)	0.9 (0.1)	**0.8 (0.1)**	0.9 (0.1)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	**1.8 (0.0)**	1.8 (0.0)
	mic-5	0.5 (0.1)	1.0 (0.0)	**0.6 (0.1)**	0.6 (0.0)	0.6 (0.1)	0.7 (0.0)
	nrobot-4	-1.1 (0.2)	-0.8 (0.2)	-1.2 (0.2)	**-1.3 (0.1)**	-1.2 (0.1)	-1.3 (0.1)
4	ack-10	-0.5 (0.0)	-0.0 (0.0)	**-0.5 (0.0)**	-0.4 (0.0)	-0.5 (0.0)	-0.5 (0.0)
	ack-5	-0.8 (0.1)	-0.3 (0.0)	-1.1 (0.1)	-0.6 (0.1)	**-0.7 (0.1)**	-0.9 (0.1)
	egg-2	**-0.2 (0.2)**	0.7 (0.2)	0.2 (0.3)	-0.8 (0.5)	-0.2 (0.2)	-0.9 (0.4)
	mat-2	0.7 (0.1)	1.0 (0.1)	**0.9 (0.0)**	1.0 (0.1)	0.8 (0.0)	0.9 (0.0)
	mat-6	1.0 (0.0)	**1.2 (0.0)**	1.0 (0.0)	1.0 (0.0)	0.9 (0.1)	0.9 (0.1)
	mic-10	1.8 (0.0)	**1.9 (0.0)**	1.8 (0.0)	1.8 (0.0)	1.7 (0.0)	1.8 (0.0)
	mic-5	0.6 (0.1)	1.0 (0.0)	**0.7 (0.0)**	0.8 (0.0)	0.6 (0.0)	0.7 (0.1)
	nrobot-4	-0.9 (0.2)	-1.0 (0.2)	-1.5 (0.2)	-1.2 (0.2)	**-1.1 (0.1)**	-1.0 (0.1)
6	ack-10	-0.4 (0.0)	-0.0 (0.0)	-0.5 (0.0)	-0.3 (0.0)	**-0.3 (0.0)**	-0.4 (0.0)
	ack-5	-0.8 (0.1)	-0.3 (0.0)	-0.9 (0.1)	-0.4 (0.0)	**-0.5 (0.1)**	-0.8 (0.1)
	egg-2	0.2 (0.2)	**0.6 (0.2)**	0.6 (0.2)	0.8 (0.2)	-0.3 (0.3)	-0.2 (0.2)
	mat-2	0.8 (0.0)	**1.0 (0.0)**	1.0 (0.1)	1.1 (0.0)	0.8 (0.0)	0.8 (0.0)
	mat-6	0.9 (0.0)	1.1 (0.0)	0.9 (0.1)	1.1 (0.0)	**1.0 (0.1)**	0.9 (0.0)
	mic-10	**1.8 (0.0)**	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)
	mic-5	0.6 (0.0)	1.0 (0.0)	**0.7 (0.1)**	0.8 (0.0)	0.8 (0.0)	0.7 (0.1)
	nrobot-4	-1.0 (0.2)	**-0.7 (0.2)**	-1.0 (0.2)	-0.8 (0.2)	-1.0 (0.2)	-1.0 (0.2)
8	ack-10	**-0.4 (0.0)**	-0.0 (0.0)	-0.5 (0.0)	-0.4 (0.0)	-0.3 (0.0)	-0.5 (0.0)
	ack-5	-0.8 (0.1)	-0.3 (0.0)	-1.0 (0.1)	-0.4 (0.0)	**-0.5 (0.1)**	-0.9 (0.1)
	egg-2	-0.1 (0.2)	0.6 (0.2)	**0.4 (0.2)**	0.9 (0.2)	0.2 (0.1)	-0.1 (0.2)
	mat-2	**0.8 (0.0)**	0.9 (0.0)	0.8 (0.1)	1.1 (0.0)	0.8 (0.0)	0.8 (0.1)
	mat-6	1.0 (0.0)	**1.2 (0.0)**	1.0 (0.0)	1.1 (0.0)	0.9 (0.0)	1.0 (0.1)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	**1.8 (0.0)**	1.8 (0.0)
	mic-5	**0.6 (0.1)**	1.0 (0.0)	0.6 (0.1)	0.7 (0.0)	0.6 (0.1)	0.5 (0.1)
	nrobot-4	-0.9 (0.2)	-0.7 (0.2)	-1.3 (0.1)	-0.9 (0.2)	**-1.1 (0.2)**	-0.9 (0.2)
16	ack-10	**-0.3 (0.0)**	-0.0 (0.0)	-0.4 (0.0)	-0.3 (0.0)	-0.3 (0.0)	-0.5 (0.1)
	ack-5	**-0.6 (0.1)**	-0.3 (0.0)	-0.8 (0.1)	-0.4 (0.0)	-0.4 (0.1)	-0.9 (0.1)
	egg-2	0.3 (0.2)	**0.4 (0.2)**	0.9 (0.1)	1.2 (0.1)	-0.4 (0.3)	0.4 (0.1)
	mat-2	0.8 (0.0)	0.9 (0.0)	**1.1 (0.0)**	1.1 (0.0)	0.9 (0.1)	0.9 (0.1)
	mat-6	1.0 (0.0)	**1.2 (0.0)**	0.9 (0.0)	1.1 (0.0)	0.9 (0.0)	1.0 (0.1)
	mic-10	**1.8 (0.0)**	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	1.8 (0.0)	1.7 (0.0)
	mic-5	0.7 (0.0)	1.0 (0.0)	**0.8 (0.0)**	0.7 (0.1)	0.7 (0.0)	0.7 (0.0)
	nrobot-4	-0.7 (0.2)	-0.8 (0.2)	-1.0 (0.2)	-0.6 (0.2)	**-0.7 (0.1)**	-0.8 (0.1)

Table 4.3: Mean and standard error of the log(regret) after 75 steps of asynchronous BO over 30 random initialisations.

k	Task	KB	TS	PLAyBOOK-L	PLAyBOOK-LL	PLAyBOOK-H	PLAyBOOK-HL
2	ack-10	-0.7 (0.0)	**-0.0 (0.0)**	-0.7 (0.1)	-0.7 (0.1)	-0.8 (0.1)	-0.6 (0.0)
	ack-5	-1.3 (0.1)	-0.3 (0.0)	-1.5 (0.1)	-0.9 (0.1)	**-1.4 (0.1)**	-1.1 (0.1)
	egg-2	-0.4 (0.3)	0.8 (0.2)	-1.1 (0.3)	**-1.6 (0.4)**	-1.5 (0.4)	-1.3 (0.4)
	mat-2	0.8 (0.1)	1.0 (0.0)	0.8 (0.1)	0.8 (0.0)	**0.8 (0.0)**	0.9 (0.0)
	mat-6	1.0 (0.0)	1.3 (0.0)	0.9 (0.1)	1.0 (0.1)	**0.9 (0.0)**	1.0 (0.0)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.7 (0.0)	1.7 (0.0)	**1.7 (0.0)**	1.8 (0.0)
	mic-5	0.4 (0.1)	0.9 (0.0)	**0.5 (0.1)**	0.5 (0.0)	0.4 (0.1)	0.7 (0.0)
	nrobot-4	**-1.3 (0.2)**	-1.1 (0.2)	-1.5 (0.2)	-1.5 (0.2)	-1.5 (0.1)	-1.6 (0.1)
4	ack-10	-0.7 (0.0)	-0.0 (0.0)	**-0.7 (0.0)**	-0.5 (0.0)	-0.7 (0.0)	-0.6 (0.0)
	ack-5	-1.1 (0.1)	-0.4 (0.0)	-1.5 (0.1)	-0.7 (0.1)	**-1.0 (0.1)**	-1.0 (0.1)
	egg-2	**-0.3 (0.2)**	0.6 (0.2)	-0.2 (0.3)	-1.1 (0.5)	-0.5 (0.2)	-1.2 (0.5)
	mat-2	0.7 (0.1)	1.0 (0.1)	**0.9 (0.0)**	1.0 (0.1)	0.8 (0.0)	0.9 (0.0)
	mat-6	1.0 (0.0)	**1.3 (0.0)**	1.1 (0.0)	1.1 (0.0)	1.0 (0.0)	1.0 (0.0)
	mic-10	1.7 (0.0)	**1.9 (0.0)**	1.7 (0.0)	1.7 (0.0)	1.7 (0.0)	1.7 (0.0)
	mic-5	0.4 (0.1)	1.0 (0.0)	**0.6 (0.0)**	0.6 (0.0)	0.5 (0.0)	0.6 (0.1)
	nrobot-4	-1.1 (0.2)	-1.2 (0.2)	-1.7 (0.2)	-1.5 (0.1)	**-1.3 (0.1)**	-1.2 (0.1)
6	ack-10	-0.6 (0.0)	-0.0 (0.0)	-0.7 (0.1)	-0.4 (0.0)	**-0.4 (0.0)**	-0.4 (0.0)
	ack-5	-1.2 (0.1)	-0.4 (0.0)	**-1.2 (0.1)**	-0.4 (0.0)	-0.7 (0.1)	-0.9 (0.1)
	egg-2	-0.2 (0.2)	**0.6 (0.2)**	0.5 (0.2)	0.7 (0.2)	-0.5 (0.3)	-0.5 (0.2)
	mat-2	0.8 (0.0)	**1.0 (0.0)**	1.0 (0.1)	1.1 (0.0)	0.8 (0.0)	0.8 (0.0)
	mat-6	1.0 (0.0)	1.2 (0.0)	1.0 (0.1)	1.2 (0.0)	**1.0 (0.0)**	1.0 (0.0)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.8 (0.0)	1.8 (0.0)	**1.8 (0.0)**	1.8 (0.0)
	mic-5	0.5 (0.1)	1.0 (0.0)	**0.7 (0.0)**	0.7 (0.0)	0.7 (0.1)	0.6 (0.1)
	nrobot-4	-1.3 (0.1)	**-1.1 (0.2)**	-1.2 (0.2)	-1.0 (0.2)	-1.3 (0.2)	-1.3 (0.2)
8	ack-10	**-0.6 (0.0)**	-0.0 (0.0)	-0.6 (0.0)	-0.4 (0.0)	-0.4 (0.0)	-0.6 (0.0)
	ack-5	-1.2 (0.1)	-0.4 (0.0)	-1.3 (0.1)	-0.5 (0.0)	**-0.7 (0.1)**	-1.1 (0.1)
	egg-2	**-0.2 (0.2)**	0.4 (0.2)	-0.0 (0.4)	0.8 (0.2)	-0.0 (0.1)	-0.3 (0.2)
	mat-2	**0.8 (0.0)**	0.9 (0.0)	0.8 (0.1)	1.1 (0.0)	0.8 (0.0)	0.7 (0.1)
	mat-6	1.0 (0.0)	1.3 (0.0)	**1.1 (0.0)**	1.2 (0.0)	1.0 (0.0)	1.0 (0.0)
	mic-10	1.8 (0.0)	1.9 (0.0)	1.7 (0.0)	1.7 (0.0)	**1.7 (0.0)**	1.7 (0.0)
	mic-5	0.5 (0.1)	0.9 (0.0)	**0.5 (0.1)**	0.7 (0.0)	0.5 (0.1)	0.5 (0.1)
	nrobot-4	-1.2 (0.1)	-1.2 (0.2)	-1.4 (0.2)	-1.2 (0.2)	**-1.3 (0.2)**	-1.1 (0.2)
16	ack-10	-0.4 (0.0)	-0.0 (0.0)	-0.6 (0.0)	-0.4 (0.0)	**-0.4 (0.0)**	-0.6 (0.1)
	ack-5	**-0.8 (0.1)**	-0.4 (0.0)	-1.0 (0.1)	-0.4 (0.0)	-0.4 (0.1)	-1.1 (0.1)
	egg-2	-0.4 (0.5)	**0.3 (0.2)**	0.8 (0.1)	1.2 (0.1)	-0.9 (0.4)	-0.5 (0.4)
	mat-2	0.8 (0.0)	0.9 (0.0)	**1.1 (0.0)**	1.1 (0.0)	0.9 (0.1)	0.8 (0.1)
	mat-6	1.1 (0.0)	**1.3 (0.0)**	1.0 (0.0)	1.2 (0.0)	1.0 (0.0)	1.0 (0.1)
	mic-10	**1.8 (0.0)**	1.9 (0.0)	1.7 (0.0)	1.8 (0.0)	1.8 (0.0)	1.7 (0.0)
	mic-5	0.5 (0.1)	0.9 (0.0)	**0.7 (0.0)**	0.7 (0.1)	0.7 (0.1)	0.6 (0.1)
	nrobot-4	-1.1 (0.2)	-1.1 (0.2)	-1.4 (0.2)	-1.0 (0.2)	**-0.9 (0.1)**	-1.1 (0.2)

Table 4.4: Mean and standard error of the log(regret) after 100 steps of asynchronous BO over 30 random initialisations.

new approach, PLAyBOOK, for asynchronous BO, based on penalisation of the acquisition function using information about tasks that are still under evaluation. Empirical evaluation on synthetic functions and a real-world optimisation task showed that PLAyBOOK improves upon the state of the art. Finally, we demonstrate that, for penalisation-based batch BO, PLAyBOOK's asynchronous BO is more efficient than synchronous BO in both wall-clock time and the number of samples.

5

Bayesian optimisation with multiple continuous and categorical inputs

Contents

So far, we have discussed applications of BO to continuous input spaces, which have historically been the preferred application domain in research (Snoek et al., 2012; Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Ru et al., 2018; Shahriari et al., 2016; Frazier, 2018; Alvi et al., 2019). However, in many situations,

optimisation problems involve a mixture of continuous and categorical variables. For example, with a deep neural network, we may want to adjust the learning rate and the number of units in each layer (continuous), as well as the activation function type in each layer (categorical). Similarly, in a gradient boosting ensemble of decision trees, we may wish to adjust the learning rate and the maximum depth of the trees (both continuous), as well as the boosting algorithm and loss function (both categorical).

Having a mixture of categorical and continuous variables presents unique challenges. If some inputs are categorical variables, as opposed to continuous, then the common assumption that the BO acquisition function is differentiable and continuous over the input space, which allows the acquisition function to be efficiently optimised, is no longer valid. Recent research has dealt with categorical variables in different ways. The simplest approach for BO with GP surrogates is to use a one-hot encoding on the categorical variables so that they can be treated as continuous variables, and perform BO on the transformed space (GPyOpt authors, 2016). Alternatively, the mixed-type inputs can be handled by a surrogate with a hierarchical structure, such as using random forests (Hutter et al., 2011; Bergstra et al., 2011) or multi-armed bandits (MABs) (Gopakumar et al., 2018). These approaches come with their own challenges, which we will discuss below (see Section 5.2). In particular, existing approaches are not well designed for *multiple* categorical variables with *multiple* possible values. Additionally, no GP-based BO methods have explicitly considered the batch setting for continuous-categorical inputs, to the best of our knowledge.

In this chapter, we present a new Bayesian optimisation approach for optimising a black-box function with multiple continuous and categorical inputs, termed Continuous and Categorical Bayesian Optimisation (CoCaBO). Our approach is motivated by the success of MABs (Auer et al., 2002a,b) in identifying the best value(s) from a discrete set of options.

Our main contributions are as follows:

- We propose a novel method which combines the strengths of MABs and BO to optimise black-box functions with *multiple* categorical and continuous inputs. (Section 5.3.1).

- We present a GP kernel to capture complex interactions between the continuous and categorical inputs (Section 5.3.2). Our kernel allows sharing of information across different categories without resorting to one-hot transformations.

- We introduce a novel batch selection method for mixed input types that extends CoCaBO to the parallel setting, and dynamically balances exploration and exploitation and encourages batch diversity (Section 5.3.3).

- We demonstrate the effectiveness of our methods on a variety of synthetic and real-world optimisation tasks with *multiple* categorical and continuous inputs (Section 5.4).

5.1 Problem definition

In this chapter, we consider the problem of optimising a black-box function $f(\mathbf{z})$ where the input $\mathbf{z}$ consists of both continuous and categorical inputs, $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$, where $\mathbf{h} = [h_1, \ldots, h_c]$ are the categorical variables, with each variable $h_i \in \{1, 2, \ldots, N_i\}$ taking one of N_i different values, and $\mathbf{x} \in \mathbb{R}^d$ is a point in a d-dimensional hypercube $\mathcal{X}$. Formally, we aim to find the best configuration to maximise the black-box function

$$\mathbf{z}^* = [\mathbf{h}^*, \mathbf{x}^*] = \arg\max_{\mathbf{z}} f(\mathbf{z}) \tag{5.1}$$

by making a series of evaluations $\mathbf{z}_1$,..., $\mathbf{z}_T$. Later we extend our method to allow parallel evaluation of multiple points, by selecting a batch $\{\mathbf{z}_t^{(i)}\}_{i=1}^b$ at each optimisation step t.

5.2 Related work

5.2.1 One-hot encoding

A common method for dealing with categorical variables is to transform them into a one-hot encoded representation, where a variable with N choices is transformed into a vector of length N with a single non-zero element. This is the approach followed by the popular BO packages like Spearmint (Snoek et al., 2012) and GPyOpt (Gonzalez et al., 2016; GPyOpt authors, 2016).

There are two main drawbacks with this approach. First, the commonly-used RBF (squared exponential, radial basis function) and Matérn kernels in the GP surrogate assume that f is continuous and differentiable in the input space, which is clearly not the case for one-hot encoded variables, as the objective is only defined for a small subspace within this representation.

The second drawback is that the acquisition function is optimised as a continuous function. By using this extended representation, we are turning the optimisation into a significantly harder problem due to the increased dimensionality of the search space. Additionally, the one-hot encoding makes our problem sparse, especially when we have multiple categories, each with multiple choices. This causes distances between inputs to become large, reducing the usefulness of the surrogate at such locations. As a result, the optimisation landscape is characterised by many flat regions, making it difficult to optimise (Rana et al., 2017).

5.2.2 Hierarchical approaches

Random forests (RFs) (Breiman, 2001) can naturally consider continuous and categorical variables, and are used in the SMAC (Sequential Model-based Algorithm Configuration) global optimisation algorithm proposed in Hutter et al. (2011) as the underlying surrogate model for f. However, the predictive distribution of the RF, which is used to select the next evaluation, is less reliable, as it relies on randomness introduced by the bootstrap samples and the randomly chosen subset of variables to be tested at each node to split the data. Moreover, RFs can easily overfit and we

need to carefully choose the number of trees. Another tree-based approach is Tree Parzen Estimator (TPE) (Bergstra et al., 2011) which is an optimisation algorithm based on tree-structured Parzen density estimators (Rosenblatt, 1956; Parzen, 1962). TPE uses nonparametric Parzen kernel density estimators to model the distribution of good and bad configurations w.r.t. a reference value. Due to the nature of kernel density estimators, TPE also supports continuous and discrete spaces.

Another more recent approach is EXP3BO (Gopakumar et al., 2018), which can deal with mixed categorical and continuous input spaces by utilising a MAB. When the categorical variable is selected by the MAB, EXP3BO constructs a GP surrogate specific to the chosen category for modelling the continuous domain, i.e. it shares no information across the different categories. The observed data are divided into smaller subsets, one for each category, and as a result EXP3BO can handle only a small number of categorical choices and requires a large number of samples.

5.3 Continuous and Categorical Bayesian Optimisation (CoCaBO)

5.3.1 CoCaBO acquisition procedure

Our proposed method, Continuous and Categorical Bayesian Optimisation, harnesses both the advantages of multi-armed bandits to select categorical inputs and the strength of GP-based BO in optimising continuous input spaces. The CoCaBO procedure is shown in Algorithm 7. CoCaBO first decides the values of the categorical inputs $\mathbf{h}_t$ by using a MAB (Step 4 in Algorithm 7). Given $\mathbf{h}_t$, it then maximises the acquisition function to select the continuous part $\mathbf{x}_t$ which forms the next point $\mathbf{z}_t = [\mathbf{h}_t, \mathbf{x}_t]$ for evaluation, as illustrated in Figure 5.1.

First selecting the categorical component, $\mathbf{h}_t$, followed by the continuous dimensions, $\mathbf{x}_t$, streamlines the acquisition procedure, but also mirrors the approach followed in practice for hyperparameters optimisation. Categorical parameters often describe higher-order properties of the algorithm, which influence the values of the continuous parameters, e.g. selecting the layer type followed by the number of units.

Algorithm 7 CoCaBO Algorithm

1: **Input:** A black-box function f, data $\mathcal{D}_0$, maximum number of iterations T.
2: **Output:** The best recommendation $\mathbf{z}_T = [\mathbf{x}_T, \mathbf{h}_T]$.
3: **for** $t = 1, \ldots, T$ **do**
4: Select $\mathbf{h}_t = [h_{1,t}, \ldots, h_{c,t}] \leftarrow \text{EXP3}(\{\mathbf{h}_i, f_i\}_i^{t-1})$.
5: Select $\mathbf{x}_t = \arg\max \alpha_t(\mathbf{x}|\mathcal{D}_{t-1}, \mathbf{h}_t)$.
6: Query at $\mathbf{z}_t = [\mathbf{x}_t, \mathbf{h}_t]$ to obtain f_t.
7: $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup (\mathbf{z}_t, f_t)$.
8: **end for**

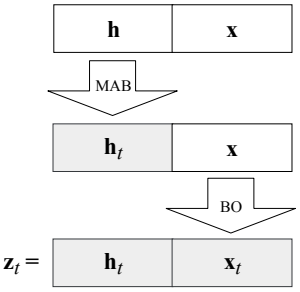


Figure 5.1: Optimisation procedure in CoCaBO

For the MAB, we chose the EXP3 (Auer et al., 2002b) method because it makes comparatively fewer assumptions on reward distributions and can be used under more general conditions than UCB and ϵ-greedy. The latter methods assume i.i.d. rewards, whereas EXP3 assumes an adversarial ordering of the rewards. For our procedure, we define the MAB's reward for each category as the best function value observed so far from that category, which is a monotonically-increasing quantity. Since this best-so-far statistic is not independent across iterations, the reward distribution is not i.i.d., and EXP3's assumption of an adversarial ordering of the rewards can model the true behaviour.

By using the MAB to decide the values for categorical inputs, we only need to optimise the acquisition function over the continuous subspace $\mathcal{X} \in \mathbb{R}^d$. In comparison to one-hot based methods, whose acquisition functions are defined over $\mathbb{R}^{(d+\sum_i^c N_i)}$, our approach enjoys a significant reduction in the difficulty and cost of optimising the acquisition function[1].

In Figure 5.2, we demonstrate the effectiveness of our approach in dealing with categorical variables on a simple synthetic example `func-2C` (described in Section 5.4.1), which comprises two categorical inputs, h_1 ($N_1 = 3$) and h_2 ($N_2 = 5$), and two continuous inputs. The optimal function value lies in the subspace when both

[1]To optimise the acquisition function to within ζ accuracy using a grid search or branch-and-bound optimiser, our approach requires only $\mathcal{O}(\zeta^{-d})$ calls (Kandasamy et al., 2015) and one-hot approaches require $\mathcal{O}(\zeta^{-(d+\sum_i^c N_i)})$ calls. The cost saving grows exponentially with the number of categories c and number of choices for each category N_i.

categorical variables $h_1 = h_2 = 2$, and we show a BO run with 200 iterations. Though we show only the categories selected by the CoCaBO acquisition procedure, we selected the corresponding continuous dimensions at each iteration before evaluating the function on the combined categorical and continuous configuration.

The correct categories are shown in strong red (h_1) and blue (h_2), with the non-optimal categories shown in a lighter red and blue. The categories selected by the CoCaBO procedure at each iteration are shown in Figure 5.2a and are summarised for the whole BO procedure in Figure 5.2b. We can see that the algorithm increasingly focuses on the correct categories $h_1 = 2$ and $h_2 = 2$ as we gather more data. Figures 5.2c and 5.2d show the EXP3 algorithm's scores for each category, and reflect the belief that the correct selections contain higher-value configurations. This results in the correct categories being chosen increasingly often. This demonstrates the suitability of using EXP3 as the MAB algorithm in the context of hyperparameter optimisation.

5.3.2 CoCaBO kernel design

We propose to use a combination of two separate kernels: $k_z(\mathbf{z}, \mathbf{z}')$ will combine a kernel defined over the categorical inputs, $k_h(\mathbf{h}, \mathbf{h}')$ with $k_x(\mathbf{x}, \mathbf{x}')$ for the continuous inputs.

For the categorical kernel, we propose using an indicator-based similarity metric,

$$k_h(\mathbf{h}, \mathbf{h}') = \frac{\sigma^2}{N_c} \sum_{i=1}^{N_c} \mathbb{I}(h_i - h_i'), \tag{5.2}$$

where σ is the kernel variance and $\mathbb{I}(h_i, h_i') = 1$ if $h_i = h_i'$ and is zero otherwise. We use this form for the categorical kernel, as it allows for covariances to exist across different categorical dimensions $(h_1, h_2, \ldots)$, but not between cateogrical selections (e.g. between $h_1 = 1$ and $h_1 = 2$, $\ldots$). This is a key weakness of the one-hot encoded approach. Our kernel can also be derived as a special case of a RBF kernel, which is explored below.

There exist an infinite number of combinations of kernels that result in valid kernels (Duvenaud et al., 2013). One approach is to sum them together. Using a

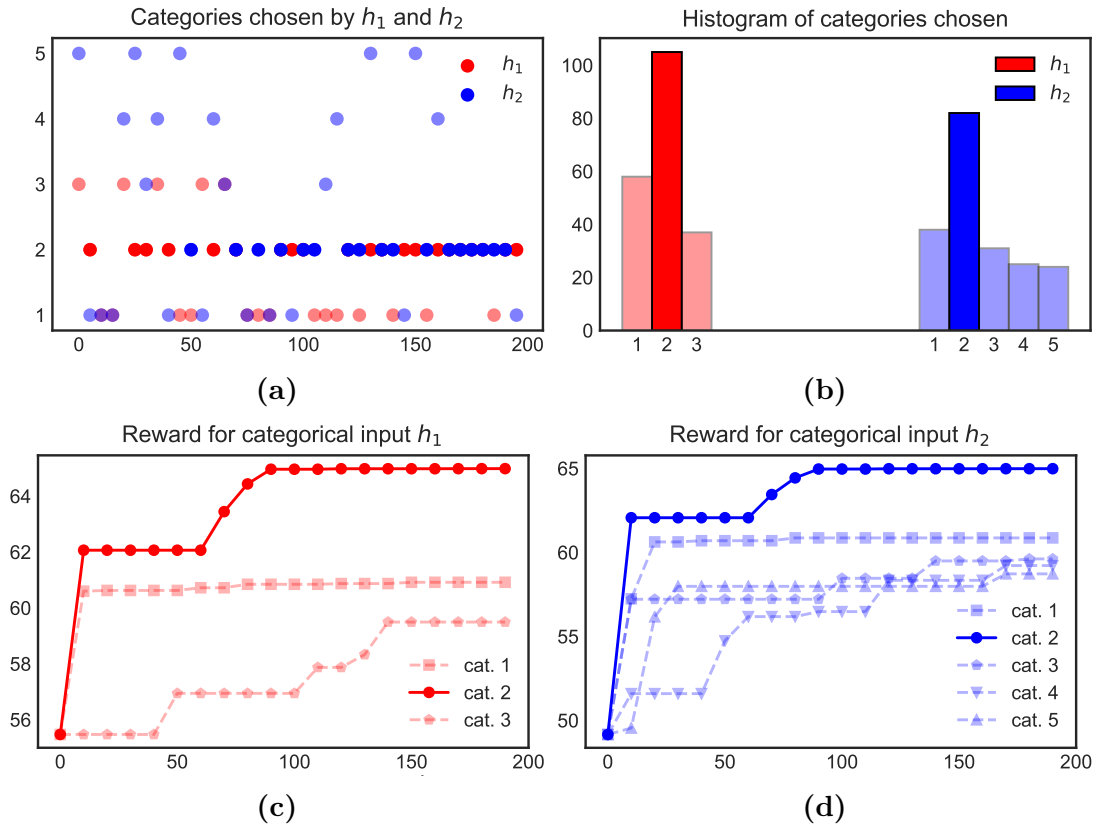


Figure 5.2: CoCaBO correctly optimises the two categorical inputs h_1 (Red) and h_2 (Blue) of the `func-2C` test function over 200 iterations. The *best* category is $h_i = 2$ for both h_1 ($N_1 = 3$) and h_2 ($N_2 = 5$), and is highlighted in all plots with strong red and blue colours. (a) shows the selections made for both categorical inputs by CoCaBO at each iteration, showing the algorithm increasingly focus on the best categories as the algorithm progresses. (b) shows the histogram of categories selected for each variable, also showing that the best category is being selected most frequently. (c) and (d) show the rewards associated with each possible categorical value for h_1 and h_2 respectively across iterations. Again, we see the correct category being identified for both categorical inputs and being assigned the highest rewards.

sum of kernels (which are each defined over different subsets of an input space) has been explored successfully for BO in the context of high-dimensional optimisation in the past (Kandasamy et al., 2015). Simply adding the continuous kernel to the categorical kernel $k_z(\mathbf{z}, \mathbf{z}') = k_x(\mathbf{x}, \mathbf{x}') + k_h(\mathbf{h}, \mathbf{h}')$, though, provides limited expressiveness, as this translates in practice to learning a single common trend over $\mathbf{x}$, and an offset depending on $\mathbf{h}$.

An alternative approach is to use the product $k_z(\mathbf{z}, \mathbf{z}') = k_x(\mathbf{x}, \mathbf{x}') \times k_h(\mathbf{h}, \mathbf{h}')$. This form allows the kernel to encode couplings between the continuous and

categorical domains, allowing a richer set of relationships to be captured. If there are no overlapping categories in the data, which is likely to occur in early iterations of BO, this would cause the product kernel to be zero and prevent the model from learning.

We therefore propose our CoCaBO kernel to automatically exploit the strengths and avoid the weaknesses of the sum and product kernels by combining them using a trade-off parameter $0 \leq \lambda \leq 1$. This parameter can be optimised jointly with the GP hyperparameters:

$$k_z(\mathbf{z}, \mathbf{z}') = (1 - \lambda)\left(k_h(\mathbf{h}, \mathbf{h}') + k_x(\mathbf{x}, \mathbf{x}')\right) + \lambda k_h(\mathbf{h}, \mathbf{h}')k_x(\mathbf{x}, \mathbf{x}'). \tag{5.3}$$

It is worth highlighting a key benefit of our formulation over alternative hierarchical methods discussed in Section 5.2.2: rather than dividing our data into a subset for each combination of categories (i.e. one dataset, and hence one surrogate model, for each setting of $\mathbf{h}$), we instead leverage all of our acquired data at every stage of the optimisation. This is made possible by the fact that our kernel is able to combine information from data within the same category as well as from different categories, as shown in Equation (5.2), which improves its modelling performance. We compare the regression performance of the CoCaBO kernel and a one-hot encoded kernel on some synthetic functions in Section 5.4.2.

Relationship between k_h and the RBF kernel

The form of the categorical kernel k_h in Equation (5.2) follows not only from the intuitive argument, that it allows us to model the degree of similarity between two selections $\mathbf{h}$ and $\mathbf{h}'$, but also directly as a special case of an RBF kernel. Consider the standard RBF kernel with unit variance evaluated between two scalar locations a and a':

$$k(a, a') = \exp\left(-\frac{1}{2}\frac{(a - a')^2}{l^2}\right). \tag{5.4}$$

The lengthscale in Equation 5.4 defines the similarity between the two inputs, and, as the lengthscale becomes smaller, the distance between locations that would be considered similar (i.e. high covariance) shrinks. The limiting case $l \to 0$

Algorithm 8 CoCaBO batch selection

Input: Surrogate data $\mathcal{D}_{t-1}$.
Output: The batch $\mathcal{B}_t = \{\mathbf{z}_t^{(1)}, \ldots, \mathbf{z}_t^{(b)}\}$.
$\mathbf{H}_t = \{\mathbf{h}_t^{(1)}, \ldots, \mathbf{h}_t^{(b)}\} \leftarrow$ EXP3-M$(\mathcal{D}_{t-1})$.
$(\mathbf{u}_1, v_1), \ldots, (\mathbf{u}_q, v_q)$ are the unique categorical values in $\mathbf{H}_t$ and their counts.
Initialise $\mathcal{B}_t = \varnothing$ and $\mathcal{D}'_{t-1} = \mathcal{D}_{t-1}$.
for $j = 1, \ldots, q$ **do**
  $\{\mathbf{x}_i\}_{i=1}^{v_j} \leftarrow \mathrm{KB}(\mathbf{u}_j, \mathcal{D}'_{t-1})$.
  $\mathbf{Z}_j = \{\mathbf{u}_j, \mathbf{x}_i\}_{i=1}^{v_j}$ and $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \mathbf{Z}_j$.
  $\mathcal{D}'_t \leftarrow \mathcal{D}'_{t-1} \cup \{\mathbf{Z}_j, \mu(\mathbf{Z}_j)\}$.
end for
return $\mathcal{B}_t$.

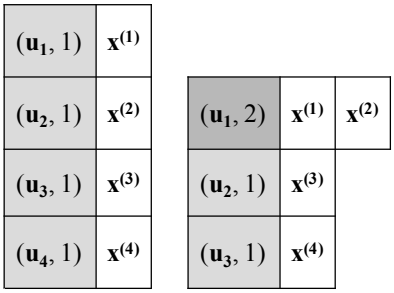


Figure 5.3: Two example cases for selecting a batch ($b = 4$)

states that if two inputs are not exactly the same as each other, then they provide no information for inferring the GP posterior's value at each other's locations. This causes the kernel to turn into an indicator function as in Equation (5.2) above (Kulis and Jordan, 2011):

$$k(a, a') = \begin{cases} 1, & \text{if } a = a' \\ 0, & \text{otherwise.} \end{cases} \tag{5.5}$$

By adding one such RBF kernel with $l \to 0$ for each categorical variable in h and normalising the output we arrive at the form in Equation (5.2).

5.3.3 Batch CoCaBO

Our focus on optimising computer simulations and modelling pipelines provides a strong motivation to extend CoCaBO to select and evaluate multiple tasks at each iteration, in order to better utilise available hardware resources (Snoek et al., 2012; Wu and Frazier, 2016; Shah and Ghahramani, 2015; Contal et al., 2013).

The batch CoCaBO algorithm uses the "multiple plays" formulation of EXP3, called EXP3.M (Auer et al., 2002b), which returns a batch of categorical choices, and combines it with the Kriging Believer (KB)[2] (Ginsbourger et al., 2010) batch

[2]Note that our approach can easily utilise other batch selection techniques if desired.

method to select the batch points in the continuous domain. See Section 2.4.3 for a description of the EXP3.M method.

We choose KB for the batch creation, as it can consider already-selected batch points, including those with different categorical values, without making significant assumptions that other popular techniques may make, e.g. local penalisation (Gonzalez et al., 2016) and our PLAyBOOK method discussed in Chapter 4 (Alvi et al., 2019) assume that f is Lipschitz continuous. KB on the other hand makes no further assumptions than those already made in the surrogate model, which in our case leverages the kernel in Equation (5.3).

Our novel contribution is a method for combining the batch points selected by EXP3.M with batch BO procedures for continuous input spaces. Assume we are selecting a batch of b points $\mathcal{B}_t = \{\mathbf{z}_t^{(i)}\}_{i=1}^b$ at iteration t. A simple approach is to select a batch of categorical variables $\{\mathbf{h}_t^{(i)}\}_{i=1}^b$ and then choose a corresponding continuous variable for each categorical point as in the sequential algorithm above, thus forming $\{\mathbf{z}_t^{(i)}\}_{i=1}^b = \{\mathbf{h}_t^{(i)}, \mathbf{x}_t^{(i)}\}_{i=1}^b$. However, such a batch method may not identify b unique locations, as some values in $\{\mathbf{h}_t^{(i)}\}_{i=1}^b$ may be repeated, which is even more problematic when the number of possible combinations for the categorical variables, $\prod_{i=1}^c N_i$, is smaller than the batch size b, as we would never identify b unique values for $\mathbf{h}$.

Our batch selection method, outlined in Algorithm 8, allows us to create a batch of unique choices by allocating multiple continuous batch points to categories associated with higher rewards. The key idea is to first collect all of the unique categorical choices and how often they occur from the MAB. These counts define how many continuous batch points will be selected for each categorical choice. For each unique $\mathbf{h}$, we select a number of batch points equal to its number of occurrences in the MAB batch using a continuous-space batch BO method, since EXP3.M can select the same values for $\mathbf{h}$ multiple times in the same batch, but we want a batch of unique configurations $\mathbf{z}_i$.

This is illustrated in Figure 5.3 for two possible scenarios. The benefit of using KB here is that the algorithm can take into account selections across the different $\mathbf{h}$ to impose diversity in the batch in a consistent manner.

5.3.4 Learning the hyperparameters in the CoCaBO kernel

We now discuss how we learn the parameters of our CoCaBO surrogate. The CoCaBO kernel is reproduced here:

$$k_z(\mathbf{z}, \mathbf{z}') = (1 - \lambda)\left(k_h(\mathbf{h}, \mathbf{h}') + k_x(\mathbf{x}, \mathbf{x}')\right) + \lambda k_h(\mathbf{h}, \mathbf{h}')k_x(\mathbf{x}, \mathbf{x}'). \tag{5.6}$$

As before, we follow the standard practice of learning hyperparameters maximising the log marginal likelihood $\mathcal{L}(\theta, \mathcal{D})$ of the GP surrogate

$$\theta^* = \arg\max_{\theta} \mathcal{L}(\theta, \mathcal{D}), \tag{5.7}$$

where we collected the the hyperparameters of both kernels as well as the CoCaBO hyperparameter into $\theta = \{\theta_h, \theta_x, \lambda\}$. We restate the log marginal likelihood and its derivative here (introduced in Chapter 2) for ease of access:

$$\mathcal{L}(\theta) = -\frac{1}{2}\mathbf{y}^\intercal \mathbf{K}^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K}| + \text{constant}, \tag{5.8}$$

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{1}{2}\left(\mathbf{y}^\intercal \mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\mathbf{K}^{-1}\mathbf{y} - \text{tr}\left(\mathbf{K}^{-1}\frac{\partial \mathbf{K}}{\partial \theta}\right)\right), \tag{5.9}$$

where $\mathbf{y}$ are the function values at sample locations and $\mathbf{K}$ is the kernel matrix of $k_z(\mathbf{z}, \mathbf{z}')$ evaluated on the training data.

We optimised this quantity via multi-started gradient descent. The gradient in Equation (5.9) relies on the gradient of the kernel k_z w.r.t. each of its parameters:

$$\frac{\partial k_z}{\partial \theta_h} = (1 - \lambda)\frac{\partial k_h}{\partial \theta_h} + \lambda k_x \frac{\partial k_h}{\partial \theta_h}, \tag{5.10}$$

$$\frac{\partial k_z}{\partial \theta_x} = (1 - \lambda)\frac{\partial k_x}{\partial \theta_x} + \lambda \frac{\partial k_x}{\partial \theta_x} k_h, \tag{5.11}$$

$$\frac{\partial k_z}{\partial \lambda} = -(k_h + k_x) + k_h k_x, \tag{5.12}$$

where we used the shorthand $k_z = k_z(\mathbf{z}, \mathbf{z}')$, $k_h = k_h(\mathbf{h}, \mathbf{h}')$ and $k_x = k_x(\mathbf{x}, \mathbf{x}')$.

5.4 Experiments

We compared CoCaBO against a range of existing methods which are able to handle problems with mixed type inputs: GP-based Bayesian optimisation with one-hot encoding (One-hot BO) (GPyOpt authors, 2016), SMAC (Hutter et al., 2011) and TPE (Bergstra et al., 2011). For all the baseline methods, we used their publicly available Python packages[3]. CoCaBO and One-hot BO both use the UCB acquisition function (Srinivas et al., 2010) with $\kappa = 2.0$, as is the popular default choice. We did not compare against EXP3BO (Gopakumar et al., 2018) because we focus on optimisation problems involving *multiple* categorical inputs with *multiple* possible values, and EXP3BO is able to handle only one categorical input with few possible values as discussed in Section 5.2.2.

In all experiments, we tested four different λ values for our method[4]: $\lambda = 1.0, 0.5, 0.0, \mathsf{auto}$, where $\lambda = \mathsf{auto}$ means λ is optimised as a hyperparameter. This leads to four variants of our method: CoCaBO-1.0, CoCaBO-0.5, CoCaBO-0.0 and CoCaBO-auto. We used a Matérn-52 kernel for k_x, as well as for One-hot BO, and used the indicator-based kernel in Equation (5.2) for k_h. For both our method and One-hot BO, we optimised the GP hyperparameters by maximising the log marginal likelihood every 10 iterations using multi-started gradient descent. When λ is learned as a hyperparameter in CoCaBO-auto, we learn its value every 25 iterations.

We tested all these methods on a diverse set of synthetic and real problems in both sequential and batch settings. TPE is only used in the sequential setting because its package HyperOpt does not provide a synchronous batch implementation. For all the problems, the continuous inputs were normalised to $\mathbf{x} \in [-1, 1]^d$ and we started each optimisation method with 24 random initial points. We ran each sequential optimisation for $T = 200$ iterations and each batch optimisation for $T = 80$ iterations. All experiments were conducted on a 36-core 2.3GHz Intel Xeon processor with 512 GB RAM.

[3]One-hot BO: https://github.com/SheffieldML/GPyOpt, SMAC: https://github.com/automl/pysmac, TPE: https://github.com/hyperopt/hyperopt

[4]Implementation available at https://github.com/rubinxin/CoCaBO_code

5.4.1 Experiment definitions

We defined a number of synthetic and real-world optimisation tasks for our empirical evaluation.

Synthetic tasks

We generated the following synthetic test functions: `func-2C`, `func-3C`, `ackley-5C5` and a series of functions `ackley-`c`C` for different values of c. Each function's input space comprises of both continuous variables and multiple categorical variables and in all cases the categorical inputs each have multiple possible values.

Table 5.1: Continuous and categorical input range of the synthetic test functions

Function f	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input conversion
`func-2C` ($d = 2$, $c = 2$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
`func-3C` ($d = 2$, $c = 3$)	h_1	$\{\text{ros}(\mathbf{x}), \text{cam}(\mathbf{x}), \text{bea}(\mathbf{x})\}$
	h_2	$\{+\text{ros}(\mathbf{x}), +\text{cam}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x}), +\text{bea}(\mathbf{x})\}$
	h_3	$\{+5 \times \text{cam}(\mathbf{x}), +2 \times \text{ros}(\mathbf{x}), +2 \times \text{bea}(\mathbf{x}), +3 \times \text{bea}(\mathbf{x})\}$
	$\mathbf{x}$	$[-1, 1]^2$
`ackley-`c`C` for $c = \{2, 3, 4, 5\}$ ($d = 1$, $N_i = 17$)	h_i for $i = 1, 2, \ldots, 5$	$\{z_i = -1 + 0.125 \times (j - 1),\ \text{for } j = 1, 2, \ldots, 17\}$
	$\mathbf{x}$	$[-1, 1]$
`ackley-5C5` for ($d = 1$, $c = 5$, $N_i = 5$)	h_i for $i = 1, 2, \ldots, 5$	$\{z_i = -1 + 0.125 \times (j - 1),\ \text{for } j = 1, 2, \ldots, 5\}$
	$\mathbf{x}$	$[-1, 1]$

`func-2C` is a test problem with 2 continuous inputs ($d = 2$) and 2 categorical inputs ($c = 2$). The categorical inputs decide the linear combinations between three 2-dimensional global optimisation benchmark functions: beale (bea), six-hump camel (cam) and rosenbrock (ros)[5]. `func-3C` is similar to `func-2C` but with 3 categorical inputs ($c = 3$) which leads to more complicated linear combinations

[5]The analytic forms of these functions are available at https://www.sfu.ca/~ssurjano/optimization.html

among the three functions. The `ackley-cC` functions, where c defines the number of categorical inputs, were generated for $c = \{2, 3, 4, 5\}$ categorical inputs and 1 continuous input ($d = 1$). Here, we converted c dimensions of the $c + 1$-dimensional Ackley function into 17 categories each by subdividing the space into equal parts and choosing the midpoint of each section as the chosen value. Lastly, we generated a variant of `ackley-5C`, named `ackley-5C5`, which divides 5 dimensions of the 6-D Ackley function into 5 categories each.

The value ranges for both continuous and categorical inputs for all of these functions are shown in Table 5.1. We report the highest function value found by the BO algorithm in the evaluation on these synthetic tasks in the following sections.

Real-world tasks

We defined three real-world tasks of tuning the hyperparameters for ML algorithms: `SVM-Boston`, `NN-Yacht` and `XG-MNIST`.

`SVM-Boston` outputs the negative mean square error of support vector machine (SVM) for regression on the test set of Boston housing dataset. We used the Nu Support Vector regression algorithm in the scikit-learn package (Pedregosa et al., 2011) and used 30% of the data for testing.

`NN-Yacht` returns the negative log likelihood of a single-hidden-layer neural network regressor on the test set of the Yacht hydrodynamics dataset (Gerritsma et al., 1981). We followed the MC Dropout implementation and the random train/test split on the dataset proposed in Gal and Ghahramani (2016)[6]. The neural network was trained using the mean squared error objective for 20 epochs with a batch size of 128. We ran 100 stochastic forward passes in the testing stage to approximate the predictive mean and variance.

Finally, `XG-MNIST` returns classification accuracy of a XGBoost algorithm (Chen and Guestrin, 2016) on the test set of the MNIST dataset. We used the *xgboost* python package[7] and adopted a stratified train/test split of 7 : 3.

[6]Code and data are available at https://github.com/yaringal/DropoutUncertaintyExps
[7]https://github.com/dmlc/xgboost

Table 5.2: Continuous and categorical input ranges of the real-world problems

Problems	Inputs $\mathbf{z} = [\mathbf{h}, \mathbf{x}]$	Input values
`SVM-Boston` ($d = 3$, $c = 3$)	kernel type h_1	{linear, poly, RBF, sigmoid}
	kernel coefficient h_2	{scale, auto}
	shrinking h_3	{shrinking on, shrinking off}
	penalty parameter x_1	$[0, 10]$
	tolerance for stopping x_2	$10^{[10^{-6}, 1]}$
	lower bound of the fraction of support vector x_3	$[0, 1]$
`NN-Yacht` ($d = 3$, $c = 3$)	activation type h_1	{ReLU, tanh, sigmoid}
	optimiser type h_2	{SGD, Adam, RMSprop, AdaGrad}
	suggested dropout value h_3	$\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5\}$
	learning rate x_1	$10^{[-5, -1]}$
	number of neurons x_2	$2^{[4,7]}$
	aleatoric variance x_3	$[0.2, 0.8]$
`XG-MNIST` ($d = 5$, $c = 3$)	booster type h_1	{gbtree, dart}
	grow policies h_2	{depthwise, loss}
	training objective h_3	{softmax, softprob}
	learning rate x_1	$[0, 1]$
	maximum depth x_2	$[1, 2, \ldots, 10]$
	minimum split loss x_3	$[0, 10]$
	subsample x_4	$[0.001, 1]$
	regularisation x_5	$[0, 5]$

We show the hyperparameters of each ML task optimised by the different BO approaches in Table 5.2. Note that we present the unnormalised range for the continuous inputs in Table 5.2 but rescaled each continuous input to $[-1, 1]$ in our experiments. All the remaining hyperparameters of the models are set to their default values.

5.4.2 Evaluation on synthetic tasks

We tested the different CoCaBO methods and baselines on the synthetic functions outlined in Table 5.1. `func-2C`, `func-3C` and `ackley-5C5` are examples with a small number of categorical variables, with few choices each. `ackley-`c`C` with $c =$

$\{2, 3, 4, 5\}$ tests the algorithms for more complex cases with many categorical choices.

Predictive performance of the CoCaBO posterior

We first investigated the quality of the CoCaBO surrogate by comparing its modelling performance against a standard GP with one-hot encoding. We used the test functions `func-2C`, `func-3C`, `ackley-`c`C` with $c \in \{2, 3, 4, 5\}$ in order to evaluate the surrogate models' accuracies for increasingly complex tasks.

Table 5.3: Mean and standard error of the predictive log likelihood of the CoCaBO and the One-hot BO surrogates on synthetic test functions. Both models were trained on 250 samples and evaluated on 100 test points. We see that the CoCaBO surrogate can model the function surface better than the One-hot surrogate as the number of categorical variables increases.

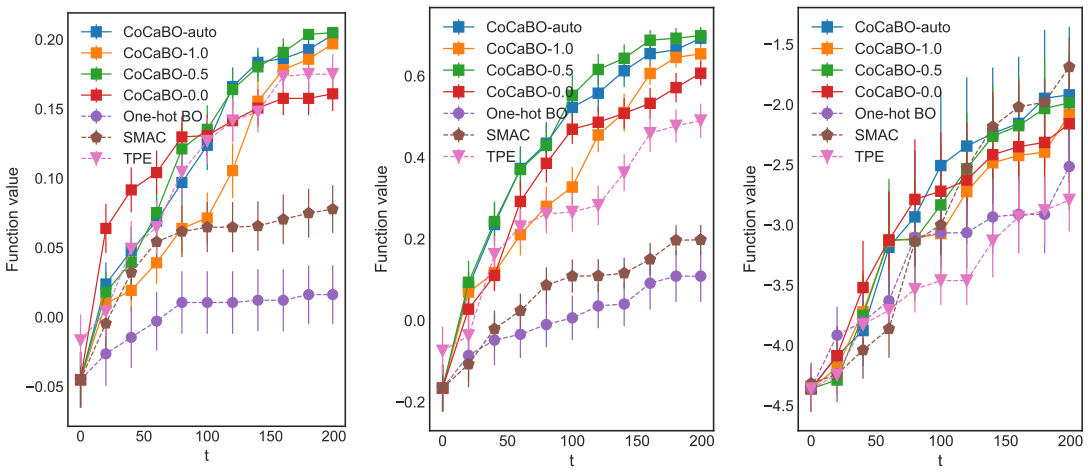
	`func-2C`	`func-3C`	`ackley-2C`	`ackley-3C`	`ackley-4C`	`ackley-5C`
CoCaBO	-531 ±260	**-435** ±85.7	**-74.7** ±9.42	**-47.2** ±9.20	**-28.3** ±13.7	**23.5** ±5.50
One-hot	**-254** ±98.0	-748 ±42.4	-77.9 ±14.2	-73.4 ±18.3	-59.8 ±18.0	7.98 ±12.5

We trained each model on 250 randomly sampled data points and evaluated the predictive log likelihood on 100 test data points. The mean and standard error over 10 random initialisations are presented in Table 5.3. The results showcase the benefit of using the CoCaBO kernel over a kernel with one-hot encoded inputs, especially when the number of categorical inputs grows. The CoCaBO kernel, which allows the GP to learn a richer set of variations from the data, leads to better out-of-sample predictions. Particularly as the number of categorical choices increases from `ackley-2C` to `ackley-5C`, our CoCaBO approach increasingly outperforms the one-hot model.

Optimisation performance of CoCaBO on synthetic test functions

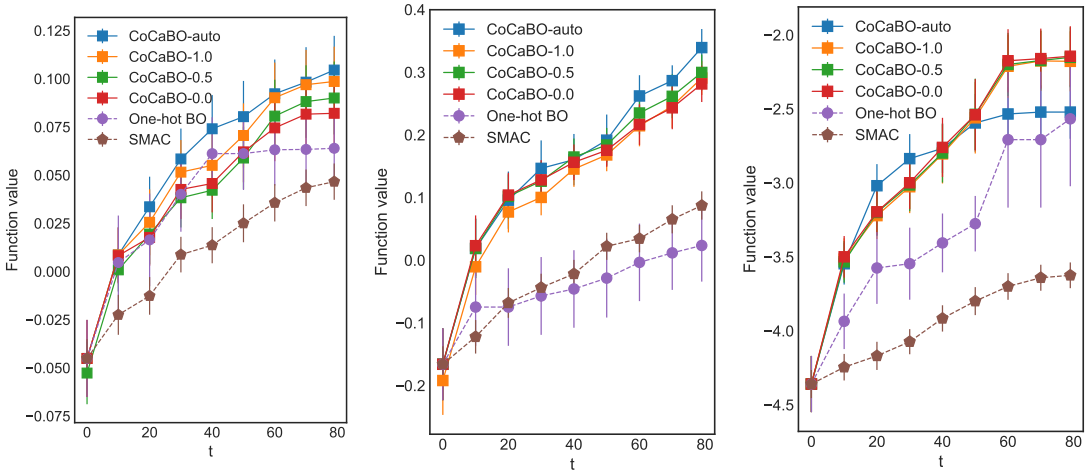
Having confirmed the modelling strength of our proposed kernel, we now move to testing the CoCaBO optimisation algorithm against the baselines.

We tested the BO methods for the sequential setting with $T = 200$ iterations and in the batch setting $b = 4$ with $T = 80$ iterations on `func-2C`, `func-3C`, `ackley-5C5` and `ackley-5C`. We repeated each experiment for 20 different random



(a) `func-2C` (b) `func-3C` (c) `ackley-5C`

Figure 5.4: Performance of CoCaBO and existing methods on synthetic test functions in the sequential setting ($b = 1$).

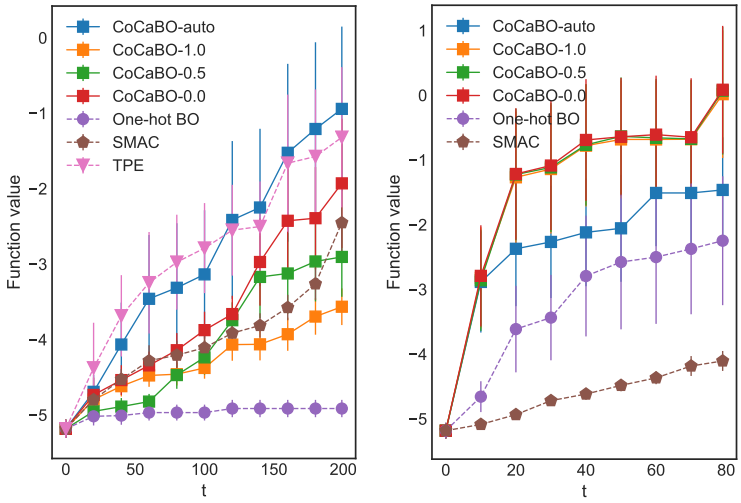


(a) `func-2C` (b) `func-3C` (c) `ackley-5C`

Figure 5.5: Performance of CoCaBO and existing methods on synthetic functions in the batch setting ($b = 4$).

initialisations and report the mean and standard error of the best function value identified by each algorithm.

In the sequential setting (Figures 5.4 and 5.6a) the four CoCaBO variants outperform the baseline methods, where we should note that TPE is the strongest competing method, having performed better than CoCaBO-0.0 once (in the `func-2C`) task, but in general showing weaker performance than CoCaBO. CoCaBO-auto (blue) and CoCaBO-0.5 (green) show the best results most often across our



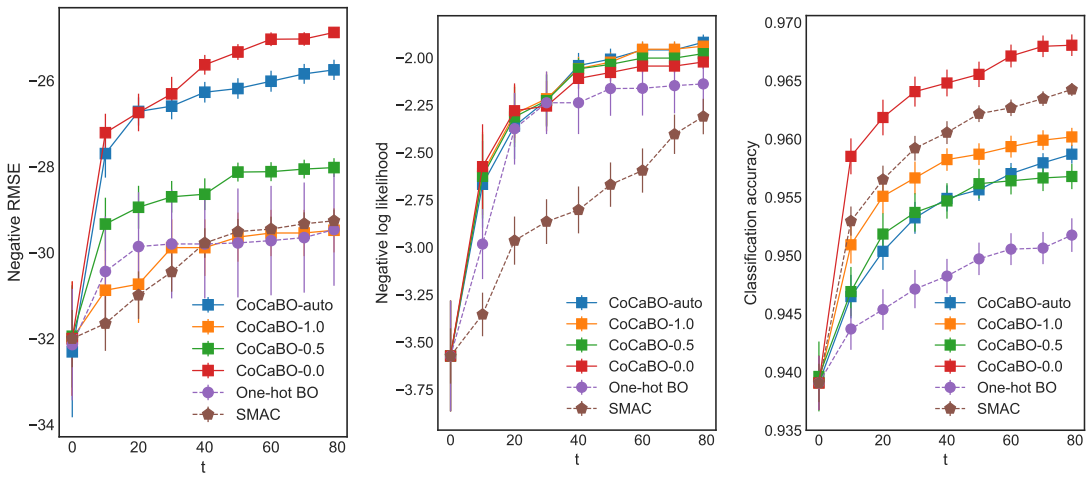
(a) Sequential setting ($b = 1$) **(b)** Batch setting ($b = 4$)

Figure 5.6: Performance of CoCaBO and existing methods on `ackley-5C5` ($N_i = 5$). The results show the comparison in both sequential (a) and batch (b) setting.

evaluation. As the number of categories increases, TPE's performance falls off and CoCaBO remains the best-performing method. In the batch setting (Figures 5.5 and 5.6b) CoCaBO outperforms both baseline methods in all of the synthetic tasks with CoCaBO-0.5 and CoCaBO-auto again demonstrating the best performance across all tasks in the batch setting.

In both the sequential and batch settings, SMAC and One-hot BO often are the weakest method, with One-hot BO being the weakest method 5 times, compared to SMAC's 3 times. We note that CoCaBO outperformed One-hot BO on the `func-2C` optimisation task, despite its surrogate performing worse in the prediction experiment in Table 5.3. We attribute this to the strength of CoCaBO in selecting better configurations at each iteration compared to One-hot BO.

This evaluation on synthetic tasks indicates that CoCaBO may indeed be a strong method for dealing with challenging global optimisation tasks.



(a) `SVM-Boston` **(b)** `NN-Yacht` **(c)** `XG-MNIST`

Figure 5.7: Performance of CoCaBO and existing methods on real-world tasks in the batch setting ($b = 4$).

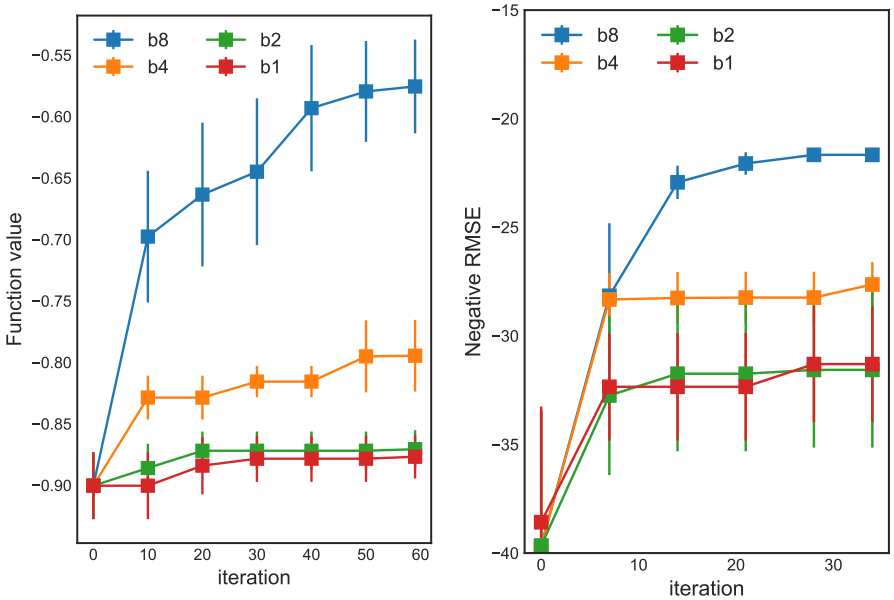
5.4.3 Evaluation on real-world tasks

Now we move to experiments on real-world tasks of hyperparameter tuning for machine learning algorithms.

The mean and standard error of the optimisation performance on our real-world optimisation tasks (Table 5.2) over 10 random repetitions for batch size $b = 4$ is presented in Figure 5.7. CoCaBO methods again show superior performance over other batch methods in these real-world problems. In the `XG-MNIST` task (Figure 5.7c), where all the categorical inputs have only binary choices, SMAC performs well but it is still overtaken by CoCaBO-0.0. We note that despite this, CocaBO-auto still remains very competitive. The strong performance of CoCaBO-0.0 may indicate a level of independence between the categorical and continuous input spaces in these real-world tasks, making the additive kernel structure sufficient to optimise the model parameters. A similar behaviour is observed in the `SVM-Boston` task (Figure 5.7a), where again CoCaBO-0.0 is the strongest method, followed by CoCaBO-auto.

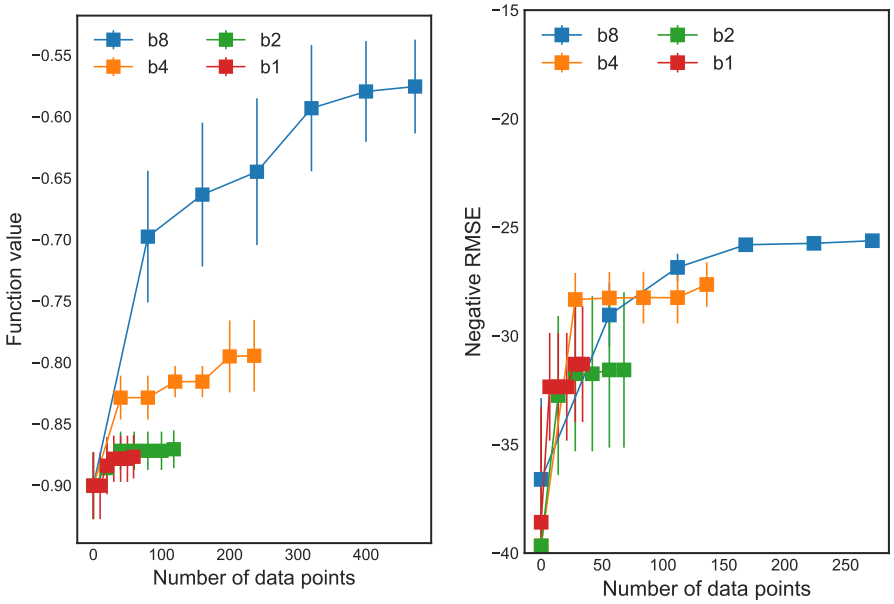
In the neural network optimisation task `NN-Yacht` (Figure 5.7b), the different CoCaBO variants perform similarly well, followed by One-hot BO and SMAC.

5.4.4 Performance for different batch sizes



(a) `ackley-5C` (b) `SVM-Boston`

Figure 5.8: Performance of CoCaBO-auto for different batch sizes $b \in \{1, 2, 4, 8\}$. We see that increasing the batch size provides a good performance improvement per iteration.



(a) `ackley-5C` (b) `SVM-Boston`

Figure 5.9: Performance of CoCaBO-auto for different batch sizes $b \in \{1, 2, 4, 8\}$. Optimisation performance shown against number of data points. We see that increasing the batch size provides a performance increase also in terms of sample efficiency.

Having shown that CoCaBO outperforms the state of the art on synthetic and real-world hyperparameter optimisation tasks in both the sequential as well as the batch setting, we finally evaluated the effectiveness of the CoCaBO batch algorithm (Algorithm 8) for different batch sizes. Here we want to ascertain the ability of the algorithm to identify batches that explore the input space efficiently. If the batch algorithm is effective, then we expect to see an improved per-iteration optimisation performance as we increase the batch size.

In Figures 5.8 we show CoCaBO-auto with different batch sizes $b \in \{1, 2, 4, 8\}$ on the `ackley-5C` and `SVM-Boston` tasks in terms of iteration number. In both cases, we see that increasing the batch size leads to more efficient optimisation, which suggests that the CoCaBO batch algorithm does indeed identify useful batches that explore the search space efficiently. This is confirmed in Figure 5.9, where we show the optimisation performance for different batch sizes against the amount of data acquired by the BO algorithm. Increasing the batch size does indeed also provide a benefit when measured against the number of samples, too.

This may indicate that the CoCaBO sequential procedure might be more exploitative than expected. The added diversity enforced by the batch setting could alleviate this behaviour and allow the algorithm to explore the parameter space more effectively.

5.5 Conclusion

Existing BO literature uses one-hot transformations or hierarchical approaches to encode real-world problems involving mixed continuous and categorical inputs. We presented a solution from a novel perspective, called Continuous and Categorical Bayesian Optimisation (CoCaBO), that harnesses the strengths of multi-armed bandits and GP-based BO to tackle this problem. We find CoCaBO to offer a very competitive alternative to existing approaches. Our method uses a new kernel structure, which allows us to capture information within categories as well as across different categories. This leads to more efficient use of the acquired data and

improved modelling power. We extended CoCaBO to the batch setting, enabling parallel evaluations at each stage of the optimisation.

Looking at the CoCaBO results across all of the experiments, using CoCaBO-auto or CoCaBO-0.5 leads to consistently strong performance, and so we recommend choosing one of these variants. If further information is known about the problem domain that may inform the kernel choice, then that could suggest choosing one of the other variants, but we believe $\lambda = 0.5$ or treating λ as a learnable hyperparameter provides a reliable optimisation algorithm.

6
Conclusions and Further Work

Contents

6.1 Conclusions

In this thesis we have explored research questions that arise when applying Bayesian optimisation to uncertain-input domains as well as hyperparameter tuning.

We considered how to choose the BO surrogate, if the data contains noise on both the inputs as well as the observed values. This scenario is relevant if the quantity being modelled is measured on a spatial grid, with the locations being determined by noisy technologies, such as triangulation or GPS. Gaussian noise on the observations is easily incorporated into the standard GP regression model via its observation likelihood, but Gaussian noise on the inputs is more challenging.

There exist a number of approaches that extend GPs to allow for input uncertainty to be modelled, with the vast majority of these extensions focussing on the autoregressive modelling use case. Despite the specific application context, all

of these models were derived by considering input uncertainty more generally. We conducted what is, to the best of our knowledge, the first empirical head-to-head analysis of uncertain-input GP methods on uncertain-input training data. We focussed on evaluating their performance as regression, rather than autoregressive, models, which reflects how they would be used in BO.

We developed a novel extension to the GP, termed UTGP, which leverages the unscented transform to model input uncertainty. A remarkable result from this empirical comparison was the fact that a *standard GP outperformed all of the existing uncertain-input methods that formed our baselines* in the regression tasks we tested. Our UTGP outperformed both the standard GP and the uncertain-input baselines, making it the best method for modelling uncertain-input data. The standard GP provided good modelling accuracy, both in terms of squared error and predictive log likelihood on held-out test data, and also avoided the computational overhead of explicitly modelling input uncertainty.

We then moved to applying BO to machine learning hyperparameter tuning, and discussed the question of running parallel BO while optimally utilising parallel computing resources. Running multiple configurations on separate machines allows for the hyperparameter space to be explored more effectively per iteration, without adversely affecting the runtime of the optimisation.

We advocated for asynchronous parallel execution of batches, to allow for the computing resources to be utilised more efficiently. We developed PLAyBOOK, a penalisation-based approach, which achieves state-of-the-art performance compared to existing asynchronous BO methods on both synthetic and real-world optimisation tasks. We also showed that the nature of penalisation-based approaches makes them more effective for asynchronous BO, and hence they should be considered primarily as asynchronous BO methods. A key result of our analysis was that there seems to be *no disadvantage to running parallel BO asynchronously.* Whether the chosen approach is an existing batch BO approach from the literature, or our proposed PLAyBOOK approach, running the algorithm asynchronously provides optimisation runtime and hardware utilisation benefits, *without sacrificing sample efficiency.*

Finally, we analysed BO for hyperparameter spaces that consist of both categorical and continuous variables. When optimising machine learning models, the hyperparameters are often a mixture of continuous and categorical variables, but BO approaches usually assume a continuous search space. Categorical variables are often dealt with by applying one-hot transformations and treating the resulting transformed space as continuous. This approach has disadvantages, both for the surrogate GP, as well as the optimisation of the acquisition function.

We developed a framework, termed CoCaBO, achieves optimisation performance superior to both one-hot encoded standard BO, as well as BO methods that replace the GP with a tree-structured surrogate to allow categorical variables to be modelled more easily. Our framework utilises a kernel structure that avoids one-hot transforming the categorical variables and instead explicitly models the similarity between configurations with different degrees of overlapping categorical values. CoCaBO also utilises a MAB to perform the selection of the categorical variables and a continuous-space BO algorithm for the continuous ones. Using a MAB to choose the values of the categorical variables bypassed the undesirable computational and modelling limitations outlined above, and also resulted in better sample efficiency compared to one-hot encoded BO and hierarchical baselines.

We extended CoCaBO by developing a batch-selection scheme that ensures that there are no redundant selections in the batches. The key requirement for the scheme was to maximise the diversity of each selected batch, which was reflected in the superior performance of batch CoCaBO, when compared to the batch BO baselines. We also found that CoCaBO's batch setting gave even better sample efficiency than its sequential setting, further incentivising parallel BO in general over the sequential setting.

6.2 Further work

The work presented in this thesis goes some way towards making BO more widely applicable, but there still remain a number of exciting related avenues of research that we have not yet investigated. Promising paths for future work could include

developing extensions to the methodologies we presented along with ideas that emerged over the course of our research but have not been explored yet.

6.2.1 Extensions of our work

- We developed two formulations of the UTGP, the simple and full UTGP, but focussed on the simple UTGP in our empirical evaluation due to the challenging computational requirements of the full UTGP. Comparing the simple and full UTGP indicated that we may be able to achieve even better performance in the regression setting by using the full UTGP. This would require a more scalable approach to computing the full UTGP to be developed.

  The key computational and memory bottlenecks in the full UTGP arise from the larger number of sigma points, and hence sigma GPs, compared to the simple UTGP. Training the full UTGP becomes challenging very quickly, as each hyperparameter update requires $O(N^3(2ND+1))$ matrix inversions – one $O(N^3)$ operation for each sigma GP. One possible avenue could leverage Cholesky update/downdate algorithms to speed up the training of the model. It may be possible to perform a smaller number of $O(N^3)$ operations and utilise the Cholesky update/downdate to compute the inverse covariance matrix for other sigma GPs more efficiently.

- Our analysis relating to the UTGP focussed on regression on a spatial grid, which is relevant to the BO context. We mentioned a popular domain for uncertain-input models is the development of autoregressive models. The strong performance of the UTGP, as well as its superior stability compared to some of the alternatives, make it a good candidate for autoregressive modelling. We believe that applied researchers utilising input uncertainty in their work could include the UTGP as one of the tools used in these contexts. Example domains include autoregressive time-series modelling and reinforcement learning (e.g. PILCO (Deisenroth and Rasmussen, 2011)).

- One aspect of the unscented transform the we did not explore in detail was how to set its parameters. IN our investigation we used the parameter settings recommended in the literature. Considering we have knowledge of the properties of the function we are modelling (in the form of the kernel hyperparameters), it may be possible to leverage this knowledge when specifying the UT parameters. One avenue that could be explored could look at how the lengthscale of the GP kernel could inform the spread of the sigma points.

- In PLAyBOOK, we approximately estimate the Lipschitz constant in the penalisers as the maximum gradient of the posterior mean. Due to the probabilistic nature of a GP, this approximation for the Lipschitz constant is likely to be incorrect for many sample functions drawn from the posterior. Alternative approaches for estimating the Lipschitz constant of a GP posterior could form an interesting research path that could be of use for PLAyBOOK, but also other areas of machine learning e.g. robust regression.

  A naïve approach to estimating a Lipschitz constant could be to sample a large number of functions from the posterior and use the highest gradient across the samples as the Lipschitz constant. Alternatively, recent work on quantifying robustness of a GP to input perturbations (Cardelli et al., 2019) could be adapted to compute bounds on the Lipschitz constant more efficiently.

  Due to our focus on parallel BO, we want to maintain the attractive computational complexity of the algorithm, so resorting to a expensive sampling-based estimate may not be adequate unless material performance gains are observed. Therefore it would be essential to evaluate such approaches in terms of their effect on the sample efficiency of the BO algorithm, while also being mindful of their computational complexity.

- Our CoCaBO framework proves that the combination of MABs and continuous BO can result in superior optimisation performance over alternative

approaches. There do remain a number of interesting research questions that would further explore some of the decisions we made.

- The MAB algorithm EXP3 is particularly well-suited to the non-stationary nature of the rewards, but the MAB literature covers a wide range of applications that can extend CoCaBO. For example contextual bandits (Zhou, 2015) could allow us to combine cost information into the algorithm, if the cost of taking a sample varies across the search space (Swersky et al., 2013).
- The categorical kernel we use in CoCaBO encodes our belief that different values of the categorical variables do not contribute to the covariance between two configurations. Intuitively this is a good assumption, as the kernel's definition still allows for covariances to exist between significantly differing configurations via the additive term. Alternative choices for the categorical kernel's form are of course possible and could constitute further research. These would probably be driven by properties of the model hyperparameters being optimised.
- Our batch algorithm for CoCaBO utilises Kriging Believer to ensure that selections across different categories are considered throughout the batch selection procedure. There are a plethora of batch BO algorithms that could take its place, and an exploration of their performance in this context could provide further insights.

6.2.2 New directions

Two directions that we did not explore in this thesis' research relate to the GP surrogate's hyperparameters. The first direction relates to the inference approach and frequency during a BO algorithm, and the second direction considers the link between BO and active learning.

Surrogate hyperparameter inference

There exists an extensive literature on acquisition functions, but comparatively little time and effort has been spent on the surrogate model. We have focussed on GPs in this thesis, but alternatives have been proposed over the years. Hutter et al. (2011) make use of random forests as surrogates in their method, and Snoek et al. (2015) investigated the use of deep neural networks as the surrogate, but still the GP remains the dominant choice in most cases.

An important question, that has received very little mention in the literature, is how best to perform hyperparameter inference on the surrogate GP hyperparameters. In the BO literature there are two approaches that each boast a strong following: MLE/MAP optimisation and marginalisation. There exist highly-cited research papers on both sides of the divide. And once we have decided which technique to use to optimise the surrogate, what should the optimisation frequency be? These are questions that we believe are of critical importance to explore.

If we decide to perform MAP inference on the hyperparameters every 10th iteration, then the runtime of each BO step will be much less than if we decided to marginalise the hyperparameters using a MCMC scheme at every iteration. Marginalising the GP hyperparameters is concerned with computing the following integral:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) = \int p(y^*|\mathbf{x}^*, \mathcal{D}, \theta) p(\theta|\mathcal{D}) \mathrm{d}\theta, \tag{6.1}$$

where $p(y^*|\mathbf{x}^*, \mathcal{D}, \theta)$ is the posterior GP evaluated at $\mathbf{x}^*$ given hyperparameters θ, and $p(\theta|\mathcal{D})$ is the hyperparameter posterior. Marginalising θ has the effect of inflating the uncertainty of the GP over that of a GP with fixed parameters. The inflated uncertainty stems from the uncertainty over the value of θ represented by the hyperparameter posterior. The dominant sampling scheme used in this case is slice sampling (Murray et al., 2010; Murray and Adams, 2010) as it can sample from unnormalised probability distributions. This is an important property, as we usually do not have access to the proper distribution $p(\theta|\mathcal{D})$, but instead only know the joint distribution $p(\mathcal{D}, \theta)$.

It is obvious that performing marginalisation is more computationally taxing than simple optimisation, both from a compute and memory point of view. There exists another class of methods that can be relevant in this context: approximate marginal GPs. Two GP extensions, BBQ (Osborne et al., 2012) and MGP (Garnett et al., 2014), both approximately marginalise the GP hyperparameters, resulting in an analytic form that has the attractive computational scaling of a MAP-trained GP, while their predictive variances also reflect the hyperparameter posterior, such as we would expect from a marginalisation.

The two key assumptions made in the MGP, which is the more recent of the two approaches, are a Laplace approximation for the hyperparameter posterior:

$$p(\theta|\mathcal{D}) \approx \mathcal{N}(\theta; \hat{\theta}, \Sigma), \tag{6.2}$$

where $\hat{\theta}$ is the MAP estimate of θ, and Σ is the Hessian of the hyperparameter posterior distribution at $\hat{\theta}$. Secondly, the hyperparameters' effect on the GP posterior is simplified to:

$$p(y|\mathbf{x}, \theta, \mathcal{D}) \approx q(y; \theta), \tag{6.3}$$

$$q(y; \theta) := \mathcal{N}\left(y; a^\top \theta + b, \nu^2\right), \tag{6.4}$$

for constants a, b and ν. These approximations give rise to an analytic posterior, where the mean of the marginal GP coincides with the mean of the parameterised GP $p(y^*|\mathbf{x}^*, \mathcal{D}, \theta)$ and the posterior variance becomes:

$$\tilde{V}(\mathbf{x}) := \frac{4}{3}\hat{V}(\mathbf{x}) + \frac{\partial \hat{m}(\mathbf{x})}{\partial \theta}^\top \Sigma \frac{\partial \hat{m}(\mathbf{x})}{\partial \theta} \tag{6.5}$$

$$+ \frac{1}{3\hat{V}(\mathbf{x})} \frac{\partial \hat{V}(\mathbf{x})}{\partial \theta}^\top \Sigma \frac{\partial \hat{V}(\mathbf{x})}{\partial \theta}, \tag{6.6}$$

where $\hat{m}(\mathbf{x})$ and $\hat{V}(\mathbf{x})$ are the posterior mean and variance of the parameterised GP respectively.

BO and active learning

BO and active learning (AL) can be considered to be closely related algorithms. BO can be thought of as an AL algorithm that efficiently reduces our uncertainty in the location of the optimum of an unknown function, whereas typical AL approaches reduce the uncertainty in our model parameters. Having discussed the reliance of BO on a good quality surrogate model, as well as the challenges relating to hyperparameter inference, we believe that introducing AL into the BO algorithm may be beneficial to the BO procedure.

Early iterations of BO are usually quite random, as the surrogate model is still relatively uninformative, so most locations in the search space are equally attractive. This means that selections are going to be affected primarily by the early-stopping criteria of the optimiser, as the acquisition function surface will be predominantly flat.

In this phase, a better strategy could be to utilise an active learning criterion, rather than the BO acquisition function, so that any samples that are taken provide maximal information about the model parameters, rather than leaving the early samples to chance. Incorporating BO with alternative sampling schemes has been proposed before in Ahmed et al. (2016), where the authors combined BO with random sampling. We believe that using a more intelligent alternative scheme than random sampling can provide more targeted benefits.

One potential active learning scheme that could be used is BALD (Houlsby et al., 2011), which identifies samples that are maximally informative about the GP hyperparameters. Garnett et al. (2014) showed how the MGP can be leveraged to make computing this criterion scalable, as they avoid the estimation of entropies that would require sampling schemes. A batch formulation, batchBALD, was also recently proposed by Kirsch et al. (2019).

Once an active learning criterion has been selected, how do we merge it with the BO acquisition function. We believe this choice can be dictated by the user's intuitions, for example we could alternate active learning and BO as in Ahmed et al. (2016). Alternatively, we could choose between AL and BO by sampling

from a Bernoulli distribution Ber(p) beginning with parameter $p = 1$ (focus on AL) and linearly decreasing to $p = 0$ (focus on BO) when the budget is exhausted. This imposes a preference for active learning early on in the budget and BO for later iterations.

Bibliography

Naoki Abe, Alan W Biermann, and Philip M Long. Reinforcement learning with immediate rewards and linear hypotheses. *Algorithmica*, 37(4):263–293, 2003.

Mohamed Osama Ahmed, Bobak Shahriari, and Mark Schmidt. Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization. *NIPS Workshop on Bayesian Optimization*, 2016.

Ibrahim Almosallam. *Heteroscedastic Gaussian processes for uncertain and incomplete data*. PhD thesis, University of Oxford, 2017.

Ahsan S Alvi, Binxin Ru, Jan Calliess, Stephen J Roberts, and Michael A Osborne. Asynchronous batch Bayesian optimisation with improved local penalisation. *arXiv preprint arXiv:1901.10452*, 2019.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 2002a.

Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM journal on computing*, 2002b.

Javad Azimi, Alan Fern, and Xiaoli Z Fern. Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, 2010.

Ashwinkumar Badanidiyuru, Robert Kleinberg, and Aleksandrs Slivkins. Bandits with knapsacks. In *IEEE 54th Annual Symposium on Foundations of Computer Science*, 2013.

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

James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*. Citeseer, 2013.

Julian Berk, Vu Nguyen, Sunil Gupta, Santu Rana, and Svetha Venkatesh. Exploration enhanced Expected Improvement for Bayesian optimization. In *Machine Learning and Knowledge Discovery in Databases*, 2019.

Donald A Berry and Bert Fristedt. Bandit problems: sequential allocation of experiments. *London: Chapman and Hall*, 1985.

Christopher M Bishop. *Pattern recognition and machine learning*. Springer, 2006.

Arno Blaas, Jose Maria Manzano, Daniel Limon, and Jan Calliess. Localised kinky inference. In *Proceedings of the European Control Conference*, 2019.

Leo Breiman. Random forests. *Machine learning*, 2001.

Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

Adam D Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 2011.

Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. An experimental comparison of Bayesian optimization for bipedal locomotion. In *International Conference on Robotics and Automation (ICRA)*. IEEE, 2014.

Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for Bayesian inference with Gaussian processes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*. ACM, 2016.

Yutian Chen, Aja Huang, Ziyu Wang, Ioannis Antonoglou, Julian Schrittwieser, David Silver, and Nando de Freitas. Bayesian optimization in AlphaGo. *arXiv preprint arXiv:1812.06855*, 2018.

Shein-Chung Chow and Mark Chang. Adaptive design methods in clinical trials – a review. *Orphanet journal of rare diseases*, 2008.

Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013.

Dennis D Cox and Susan John. A statistical method for global optimization. In *Proceedings of the International Conference on Systems, Man, and Cybernetics*. IEEE, 1992.

Patrick Dallaire, Camille Besse, and Brahim Chaib-Draa. An approximate inference with Gaussian process to latent functions from uncertain data. *Neurocomputing*, 2011.

Nando De Freitas, Alex Smola, and Masrour Zoghi. Exponential regret bounds for Gaussian process bandits with deterministic observations. *arXiv preprint arXiv:1206.6457*, 2012.

DeepMind. Alphago. https://ai.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html, 2016. [accessed 21-July-2019].

Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning*, 2011.

Li Deng and Yang Liu. *Deep Learning in Natural Language Processing*. Springer, 2018.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure discovery in nonparametric regression through compositional kernel search. In *International Conference on Machine Learning*, 2013.

Bob Fisher. Introduction to Unscented Kalman Filter. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0809/qi.pdf, 2009. [accessed 21-July-2019].

Peter I Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML-16)*, 2016.

Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *Journal of the ACM*, 2006.

Jacob Gardner, Chuan Guo, Kilian Weinberger, Roman Garnett, and Roger Grosse. Discovering and exploiting additive structure for Bayesian optimization. In *Artificial Intelligence and Statistics*, 2017.

Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. Bayesian optimization with inequality constraints. In *ICML*, 2014.

R Garnett, M Osborne, and P Hennig. Active learning of linear embeddings for Gaussian processes. In *30th Conference on Uncertainty in Artificial Intelligence*, 2014.

J Gerritsma, R Onnink, and A Versluis. Geometry, resistance and stability of the delft systematic yacht hull series. *International shipbuilding progress*, 1981.

Marzyeh Ghassemi, Tristan Naumann, Peter Schulam, Andrew L Beam, and Rajesh Ranganath. Opportunities in machine learning for healthcare. *arXiv preprint arXiv:1806.00388*, 2018.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. A multi-points criterion for deterministic parallel global optimization based on Gaussian processes. 2008.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems*. Springer, 2010.

David Ginsbourger, Janis Janusevskis, and Rodolphe Le Riche. Dealing with asynchronicity in parallel Gaussian process based global optimization. In *4th International Conference of the ERCIM WG on computing & statistics*, 2011.

Agathe Girard and Roderick Murray-Smith. Learning a Gaussian process model with uncertain inputs. *Department of Computing Science, University of Glasgow, Tech. Rep. TR-2003-144*, 2003.

Agathe Girard, Carl Edward Rasmussen, Joaquin Quinonero Candela, and Roderick Murray-Smith. Gaussian process priors with uncertain inputs application to multiple-step ahead time series forecasting. In *Advances in neural information processing systems*, 2003.

John Gittins, Kevin Glazebrook, and Richard Weber. *Multi-armed bandit allocation indices*. John Wiley & Sons, 2011.

John C Gittins and David M Jones. A dynamic allocation index for the discounted multiarmed bandit problem. *Biometrika*, 1979.

David González, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A review of motion planning techniques for automated vehicles. *Transactions on Intelligent Transportation Systems*, 2015.

Javier Gonzalez, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch Bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016.

Javier González, Michael Osborne, and Neil D Lawrence. GLASSES: Relieving the myopia of Bayesian optimisation. 2016.

Shivapratap Gopakumar, Sunil Gupta, Santu Rana, Vu Nguyen, and Svetha Venkatesh. Algorithmic assurance: An active approach to algorithmic testing using Bayesian optimisation. In *Advances in Neural Information Processing Systems*, 2018.

The GPyOpt authors. GPyOpt: A Bayesian optimization framework in python. http://github.com/SheffieldML/GPyOpt, 2016.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 2012.

James Hensman, Nicolo Fusi, and Neil D Lawrence. Gaussian processes for big data. In *Uncertainty in Artificial Intelligence*. Citeseer, 2013.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, 2014.

José Miguel Hernández-Lobato, James Requeima, Edward O Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In *International Conference on Machine Learning*, 2017.

Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 2011.

Ali Jalali, Javad Azimi, Xiaoli Fern, and Ruofei Zhang. A Lipschitz exploration-exploitation scheme for Bayesian optimization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013.

Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

Andrew H Jazwinski. *Stochastic processes and filtering theory*. Academic, 1970.

Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the Lipschitz constant. *Journal of optimization Theory and Applications*, 1993.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 1998.

Andre G Journel and Charles J Huijbregts. *Mining geostatistics*. Academic press London, 1978.

Simon J Julier. Skewed approach to filtering. In *Signal and Data Processing of Small Targets*. International Society for Optics and Photonics, 1998.

Simon J Julier. The scaled unscented transformation. In *Proceedings of the 2002 American Control Conference*. IEEE, 2002.

Simon J Julier and Jeffrey K Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Signal processing, sensor fusion, and target recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997a.

Simon J Julier and Jeffrey K Uhlmann. Consistent debiased method for converting between polar and Cartesian coordinate systems. In *Acquisition, Tracking, and Pointing XI*. International Society for Optics and Photonics, 1997b.

Simon J Julier and Jeffrey K Uhlmann. Unscented filtering and nonlinear estimation. *Proceedings of the IEEE*, 2004.

Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 1960.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional Bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, 2015.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised Bayesian optimisation via Thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, 2018.

Tarun Kathuria, Amit Deshpande, and Pushmeet Kohli. Batched Gaussian process bandit optimization via determinantal point processes. In *Advances in Neural Information Processing Systems*, 2016.

Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. BatchBALD: Efficient and diverse batch acquisition for deep Bayesian active learning. *arXiv preprint arXiv:1906.08158*, 2019.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Brian Kulis and Michael I Jordan. Revisiting k-means: New algorithms via Bayesian nonparametrics. *arXiv preprint arXiv:1111.0352*, 2011.

Sameer Kumar and Kevin B Moore. The evolution of global positioning system (GPS) technology. *Journal of science Education and Technology*, 2002.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 1964.

Tze Leung Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 1985.

Antonio Lavecchia. Machine-learning approaches in drug discovery: methods and applications. *Drug discovery today*, 2015.

Cheng Li, Sunil Gupta, Santu Rana, Vu Nguyen, Svetha Venkatesh, and Alistair Shilton. High dimensional Bayesian optimization using dropout. *arXiv preprint arXiv:1802.05400*, 2018.

Maxwell W Libbrecht and William Stafford Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 2015.

Yang Liu and Mingyan Liu. An online learning approach to improving the quality of crowd-sourcing. In *SIGMETRICS Performance Evaluation Review*. ACM, 2015.

Prasanta Chandra Mahalanobis. On the generalized distance in statistics. National Institute of Science of India, 1936.

Andrew McHutchon and Carl E Rasmussen. Gaussian process training with input noise. In *Advances in Neural Information Processing Systems*, 2011.

Mark McLeod, Michael A Osborne, and Stephen J Roberts. Optimization, fast and slow: optimally switching between local and Bayesian optimization. *arXiv preprint arXiv:1805.08610*, 2018.

Riccardo Miotto, Fei Wang, Shuang Wang, Xiaoqian Jiang, and Joel T Dudley. Deep learning for healthcare: review, opportunities and challenges. *Briefings in bioinformatics*, 2017.

Iain Murray and Ryan P Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Advances in Neural Information Processing Systems*, 2010.

Iain Murray, Ryan Prescott Adams, and David JC MacKay. Elliptical slice sampling. 2010.

Umberto Noè and Dirk Husmeier. On a new improvement-based acquisition function for Bayesian optimization. *arXiv preprint arXiv:1808.06918*, 2018.

José Nogueira, Ruben Martinez-Cantin, Alexandre Bernardino, and Lorenzo Jamone. Unscented Bayesian optimization for safe robot grasping. In *International Conference on Intelligent Robots and Systems*. IEEE, 2016.

Michael Osborne. *Bayesian Gaussian Processes for Sequential Prediction, Optimisation and Quadrature*. PhD thesis, PhD thesis, University of Oxford, 2010.

Michael A Osborne, Roman Garnett, and Stephen J Roberts. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization*, 2009.

Michael A. Osborne, David K. Duvenaud, Roman Garnett, Carl E. Rasmussen, Stephen J. Roberts, and Zoubin Ghahramani. Active learning of model evidence using Bayesian quadrature. In *Advances in Neural Information Processing Systems*, 2012.

Biswajit Paria, Kirthevasan Kandasamy, and Barnabás Póczos. A flexible multi-objective Bayesian optimization approach using random scalarizations. *arXiv preprint arXiv:1805.12168*, 1(7.3):3, 2018.

Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 1962.

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

William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

Vladimir Semenovich Pugachev. *Theory of Random Functions: And Its Application to Control Problems*. Pergamon Press, 1967.

Yao Quanming, Wang Mengshuo, Jair Escalante Hugo, Guyon Isabelle, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Filip Radlinski, Robert Kleinberg, and Thorsten Joachims. Learning diverse rankings with multi-armed bandits. In *Proceedings of the 25th international conference on Machine learning*. ACM, 2008.

Santu Rana, Cheng Li, Sunil Gupta, Vu Nguyen, and Svetha Venkatesh. High dimensional Bayesian optimization with elastic Gaussian process. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Carl Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

Paul Rolland, Jonathan Scarlett, Ilija Bogunovic, and Volkan Cevher. High-dimensional Bayesian optimization via additive models with overlapping groups. *arXiv preprint arXiv:1802.07028*, 2018.

Murray Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 1956.

Binxin Ru, Michael Osborne, and Mark McLeod. Fast information-theoretic Bayesian optimisation. In *International Conference on Machine Learning*, 2018.

Denis Sauré and Assaf Zeevi. Optimal dynamic assortment planning with demand learning. *Manufacturing & Service Operations Management*, 2013.

Steven L Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 2010.

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

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 2016.

Aleksandrs Slivkins. Introduction to multi-armed bandits. *arXiv preprint arXiv:1904.07272*, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2012.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian optimization using deep neural networks. In *International conference on machine learning*, 2015.

Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, 2010.

Cyrill Stachniss. Lecture: Robot mapping unscented Kalman filter. http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf-4.pdf, 2006. [University of Freiburg; accessed 19-July-2019].

Daniel M Steinberg and Edwin V Bonilla. Extended and unscented Gaussian processes. In *Advances in Neural Information Processing Systems*, 2014.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 2. MIT press Cambridge, 1998.

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task Bayesian optimization. In *Advances in neural information processing systems*, 2013.

Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013.

Taishi Uchiya, Atsuyoshi Nakamura, and Mineichi Kudo. Algorithms for adversarial bandit problems with multiple plays. In *International Conference on Algorithmic Learning Theory*. Springer, 2010.

George E Uhlenbeck and Leonard S Ornstein. On the theory of the Brownian motion. *Physical review*, 1930.

Joannes Vermorel and Mehryar Mohri. Multi-armed bandit algorithms and empirical evaluation. In *European conference on machine learning*. Springer, 2005.

Eric A Wan and Rudolph Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*. IEEE, 2000.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning*, 2017.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 2016.

Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

Peter Whittle. Restless bandits: Activity allocation in a changing world. *Journal of applied probability*, 1988.

Jian Wu and Peter Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems*, 2016.

Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 2016.

Li Zhou. A survey on contextual multi-armed bandits. *arXiv preprint arXiv:1508.03326*, 2015.