Sample-efficient Optimization Using Neural Networks

by

Mashall Aryan

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2020

Abstract

The solution to many science and engineering problems includes identifying the minimum or maximum of an unknown continuous function whose evaluation inflicts non-negligible costs in terms of resources such as money, time, human attention or computational processing. In such a case, the choice of new points to evaluate is critical. A successful approach has been to choose these points by considering a distribution over plausible surfaces, conditioned on all previous points and their evaluations. In this sequential bi-step strategy, also known as Bayesian Optimization, first a prior is defined over possible functions and updated to a posterior in the light of available observations. Then using this posterior, namely the surrogate model, an infill criterion is formed and utilized to find the next location to sample from. By far the most common prior distribution and infill criterion are Gaussian Process and Expected Improvement, respectively.

The popularity of Gaussian Processes in Bayesian optimization is partially due to their ability to represent the posterior in closed form. Nevertheless, the Gaussian Process is afflicted with several shortcomings that directly affect its performance. For example, inference scales poorly with the amount of data, numerical stability degrades with the number of data points, and strong assumptions about the observation model are required, which might not be consistent with reality. These drawbacks encourage us to seek better alternatives. This thesis studies the application of Neural Networks to enhance Bayesian Optimization. It proposes several Bayesian optimization methods that use neural networks either as their surrogates or in the infill criterion.

This thesis introduces a novel Bayesian Optimization method in which Bayesian Neural Networks are used as a surrogate. This has reduced the

computational complexity of inference in surrogate from cubic (on the number of observation) in GP to linear. Different variations of Bayesian Neural Networks (BNN) are put into practice and inferred using a Monte Carlo sampling. The results show that Monte Carlo Bayesian Neural Network surrogate could performed better than, or at least comparably to the Gaussian Process-based Bayesian optimization methods on a set of benchmark problems.

This work develops a fast Bayesian Optimization method with an efficient surrogate building process. This new Bayesian Optimization algorithm utilizes Bayesian Random-Vector Functional Link Networks as surrogate. In this family of models the inference is only performed on a small subset of the entire model parameters and the rest are randomly drawn from a prior. The proposed methods are tested on a set of benchmark continuous functions and hyperparameter optimization problems and the results show the proposed methods are competitive with state-of-the-art Bayesian Optimization methods.

This study proposes a novel Neural network-based infill criterion. In this method locations to sample from are found by minimizing the joint conditional likelihood of the new point and parameters of a neural network. The results show that in Bayesian Optimization methods with Bayesian Neural Network surrogates, this new infill criterion outperforms the expected improvement.

Finally, this thesis presents *order-preserving generative models* and uses it in a variational Bayesian context to infer *Implicit Variational Bayesian Neural Network* (IVBNN) surrogates for a new Bayesian Optimization. This new inference mechanism is more efficient and scalable than Monte Carlo sampling. The results show that IVBNN could outperform Monte Carlo BNN in Bayesian optimization of hyperparameters of machine learning models.

Acknowledgments

I would like to thank all the people whose assistance has been vital in completion of this thesis. First, I wish to express my deepest gratitude to the best academic supervisory team ever, A/Prof. Marcus Frean, A/Prof. J P Lewis and Prof. Stephen Marsland, for their continued guidance, encouragement and support. Marcus is a great scholar who taught, guided and inspired me. His immense erudition is accompanied by a remarkable depth of humanity. I am truly grateful to him not only for the many hours of intellectual and academic interactions, but also for instances of personal support and guidance that he extended always with great kindness and benevolence. J P has always been a source of encouragement and inspiration and has enthusiastically provided me with fantastic supervisions and thoughtful advice; a perfect, responsible, caring and kind supervisor and a true friend. Stephen joined my supervisory team at some point in the middle of my project, but his influence on this accomplishment is no less. His pragmatism, constructive criticism, insightful suggestions and persistent help have been instrumental in completion of this thesis.

I would like to extend my sincere thanks to lovely Patricia Stein, Suzan Hall and Mark Davies whose work made my life easy and facilitated the research. I would love to thank my mother, Tufan Niknejad, my brother, Arya Aryan as well as Sogand and Kiu Niknejad (my aunt and uncle) for their love, unwavering encouragement and sacrifices they made for me. Most importantly, I wish to thank my loving and supportive wife, Marjan, who provided unending inspiration.

Contents

List of Figures

Chapter 1

Introduction

In a great many domains, solving a problem involves a ”mission critical” step in which there is a need to tune some set of parameters of the system to get the best possible outcome. Often background knowledge is at a premium and, as a result, this tuning has to be based predominantly on the results of previous experience with particular settings. How might one arrive at ”optimal”, or at least very good, settings without needing too much experience? Heuristics and guess-work may be sufficient, but when the expense of gaining that previous experience is high, the question of where best to try next becomes important enough to warrant theoretical investigation in its own right. The resulting field, known as Bayesian Optimization, has made heavy use of machine learning models, however, it has not been obvious how to use one in particular despite its general popularity, namely neural networks. This thesis introduces novel methods using which neural networks can be leveraged in pursuit of the best samples when tuning a system.

1.1 Problem Statement

Optimization is the task of finding the maximum or minimum value of some objective function over the set of allowable inputs. It has been ex-

tensively studied for for a long time [200, 115, 126], not least because it has applications in various parts of science and engineering, and many algorithms have been identified for different types of problem.

When the objective function is differentiable, also known as *smooth*, some form of gradient descent is commonly used, which leads to a local optimum, i.e., while there is a guarantee that the solution found is optimal, this only holds within a local neighbourhood, not over the entire function range. A stronger guarantee, of global optimality, can be given in the case where the function and set of potential solutions are both convex.

Global optimization of *non-convex black-box* functions has been broadly studied over the previous decades [171]. Black-box in this context means that neither there is initially an analytical insight into the function, nor its derivatives are accessible. Whereas, the non-convexity implies that the function might possibly have multiple locally optimal points. If such an objective function is cheap to evaluate, then an extensive number of function evaluations can be made, and a variety of established approaches are available.

However, the focus of this thesis is on global optimization of black-box where the objective function is expensive to evaluate, i.e., the time and resources required for calculation of the target values are *non-negligible* under a *limited budget*. Consequently, in such problems the use of conventional global optimization methods is very much restricted, if feasible at all. This gives rise to a family of *Sequential Model-based Optimization* methods [99] in which the goal is to find the global optimum of the target function in an *efficient* way, i.e., to locate the optimum, while minimizing the total costs of the optimization. This is known as *Efficient Global Optimization* [107].

Sequential Model-based Optimization methods have been studied in design and simulation of engineering experiments, where the target functions are mostly computationally intensive codes germane to the simulation of complex systems [58]. A plethora of these applications exist, with examples ranging from speech recognition [209], robot gait optimization [20, 197, 133]

and clinched joints enhancement [182] to design of automotive body structure [155] and sub-orbital space planes [47]. More recently, in the *Machine Learning* community, this optimization strategy has been applied for *Automated Machine Learning*, where it has proven effective in hyperparameter tuning of machine learning models [108, 55, 207].

Assumptions about properties of the *expensive target function* are naturally application dependent. The most simplistic assumption is to consider the objective function as *deterministic*, i.e., the output values (observations) are error free. However, this assumption is often wrong, for example if the observations are collected from a physical process where the measurements are not %100 accurate. In such circumstances, it is reasonable to assume the target function to be stochastic and hence include a noise model to account for these inaccuracies in the optimization.

When a large number of direct evaluations is not possible, a successful practice is to maintain a *cheap-to-evaluate* interpolating model of the expensive objective function [154]. This so-called *surrogate* or *meta-model* acts as a *proxy* for the costly objective function and entails the most current assumptions and information about the characteristics of the objective function. Then, by using the information summarized in the surrogate model, an acquisition mechanism suggests new points that are directly evaluated under the expensive target function. Using this acquisition mechanism, the underlying function is only sampled where a new observation is expected to contribute in identifying its global optimum.

The general scheme of such an optimization is as follows: at the onset of the process, the surrogate model is trained based on an initial set of already-evaluated samples that are selected using a formalized *space-filling* strategy. Then, the search proceeds by sequentially 1) finding a new sample, termed an *infill point*, 2) evaluating it under the expensive target function and 3) including the result into the meta-model. The infill point is obtained by doing a thorough search over the surface of an *acquisition function*, also known as *infill criterion*, which itself is a wrapper around the surrogate

model.

The simplest form of this optimization scheme uses a deterministic regression model, such as *Support Vector Regressor* [59], *Radial Basis Function* [56, 178] and polynomial model, also known as a *response surface* [7]. As these surrogate models could only provide point estimates for their inputs, the infill criterion used with them is simply the model function. In this case, infill points are extrema of the model surface - for instance minima in case of minimization.

The performance of a sequential model-based optimization method is very much contingent on its ability to maintain a balance between the local and global elements of the search. The global element element (*exploration*) is the mechanism to obtain knowledge about the characteristics of the objective function, whereas the local element (*exploitation*) is the process of using this knowledge to find the best possible point. In sequential model-based optimization algorithms with deterministic surrogate, the space-filling strategy is the only exploratory phase in the whole search procedure. If not performed properly, the search method may quickly be deceived and get stuck in local optima. This inherent weakness is due to their inability to represent uncertainty about the surface of target function. As a result, they do not possess the very essential characteristics of a global search method, i.e., having the ability to pay attention to less-explored regions of the search space [106]. However, if the surrogate model is able to provide an uncertainty measure with its prediction, then a set of infill criteria can be defined, which utilizes this uncertainty to drive the exploration.

The Bayesian formalism is the de-facto approach to incorporate uncertainty assumption into modelling. Under this framework, it is possible to incorporate our knowledge about the characteristics of the objective function into the meta-model. This can easily be done by defining an *a priori* distribution over the possible target functions. The advantages of using Bayesian models in optimization are three-fold. First, they can cope with inaccuracies in the target function values, making them suitable for

modelling both deterministic and stochastic target functions. Second, they could provide an uncertainty measure with their predictions based on clear and simple a priori assumptions about the characteristics of the objective function. Subsequently, a set of theoretically motivated infill criteria is defined which could potentially balance the trade-off between exploration and exploitation. Last but not least, they can provide an adequate termination condition for the optimization process.

Methods that use Bayesian surrogate models have constantly appeared in the optimization literature under various titles of *Bayesian Optimization* [15], *Efficient Global Optimization* [107], *Gaussian Process Optimization* [61] and *Kriging-Based Optimization* [106]. Provided that correct priors are used, the Bayesian approach to optimization could not only locate the global optimum of any continuous target function in an asymptotic regime, but also minimize a deviation from the global optimum for any fixed number of observations [152]. Due to these advantages, it is assumed that Bayesian Optimization methods are generally at least as good as, or even superior to, other global optimization methods especially when the evaluation budget is limited [151].

This thesis concerns utilizing **Neural Networks** (NN) under a **Bayesian framework** in **Bayesian Optimization** (BO) in order to enhance its characteristics; **Neural Networks** (NN) possess several desirable qualities which make them appealing candidates to be used in Bayesian optimization. For example, they have time and space complexities that are lower than *Gaussian process*s (GP), the most commonly used models in BO. Interestingly, in a Bayesian framework, NNs behave similarly and under certain circumstances are proven to be equal to GPs [143, 125, 158, 104]. However, as opposed to GPs whose inference has a simple analytical solution, for Neural Networks, as well as the majority of other statistical models, conventional Bayesian inference techniques are either intractable or too complex. Consequently, until recently, methods beyond Gaussian Process have not been approached by Bayesian optimization researchers. Recent advancements in

computation power and parallel processing solutions as well as emergence of more efficient general purpose Bayesian inference techniques, encourage us to investigate this possibility.

1.2 Motivation

In this section, we first go through the major challenges in the optimization of expensive-to-evaluate functions and elaborate shortcomings of the classical Bayesian optimization in face of them. Then, we elaborate our reasons for using NNs as a potential solution to these challenges.

1.2.1 Challenges of Gaussian Process Optimization

Bayesian optimization algorithms have been shown to be useful in a wide range of applications from experiments which involve interaction with the physical world, such as therapeutic spinal electro-stimulation [195], to the ones that comprised of computationally intensive simulation code, for example simulation of Computational Fluid Dynamics for aircraft aerodynamic design [216]. The set of assumptions about the characteristics of the objective function might vary largely between different applications. In the simplest scenario, it could be a deterministic, continuous and stationary (i.e. the covariance between two outputs is invariant to translations in input space) function defined on a low dimensional domain. However, there are more complicated cases where the observations from the expensive function are noisy, also known as aleatoric uncertainty. In a more dramatized situation the noise rate might be position dependent, that is referred to as heteroscedasticity. It is possible that the continuity assumption is violated, for example if the underlying objective function is from the class of neural networks or regression trees. The stationarity assumption might not be applicable and the covariance structure might change depending on the location in the input space, for example in the optimal design of

NASA rocket booster [73]. Last but not least, the objective function might be defined on a high dimensional domain.

A desirable Bayesian optimization method should be able to cope with the above-mentioned complications while keeping the model building/update cost reasonably lower than the cost imposed by evaluating the underlying objective function. Classic Bayesian optimization, i.e., Bayesian optimization with an *Expected Improvement* (see section 2.4.3) infill criteria and *Maximum likelihood* inference for the Gaussian Process, is unable to overcome most of these difficulties. It also contains some problems that are inherent to its own model structure. In the following, a brief explanation is given to some of these problems and their state-of-the-art workarounds.

As a non-parametric model, one of the most obvious shortcomings of Gaussian Process is the computational cost of performing inference. Exact inference in a Gaussian Process requires inversion of the covariance matrix which, using standard methods, scales cubically with the number of observations. If the covariance function is parametrized, as it usually is, optimization of the (hyper)parameters involves inversion of an extended covariance matrix with each new observation. This is very unfavourable in the efficient global optimization realm as the computational cost of model-fitting soon exceeds the cost of evaluating the objective function and makes the rest of optimization inefficient. A wide range of approximating methods has been introduced to address this problem including using subset of data [188] sparse approximations [42, 64, 213, 172, 103, 212, 53], low-rank and sparse covariance functions [145, 17, 87], variational inference [199, 82, 123], distributed learning [35, 140] and random feature expansions [33, 174]. Although efficient, these solutions are approximates methods, and therefore utilizing them in the optimization framework adds a new source of inaccuracy to the surrogate model.

In Bayesian optimization, exploitation and exploration are performed by looking at regions with low prediction mean and high prediction variance, respectively. A major problem with classical Bayesian optimization is

that it is biased towards exploitation rather than exploration. The problem has roots in the way that the GP underestimates the prediction variance, whose calculation includes the assumption that covariance values are independent of model output. However, when parametrized covariance functions are used, this assumption is violated and the model parameters (covariance values) are tied to the model output via tunable hyperparameters. Furthermore, values for these hyperparameters are usually computed in the light of data by maximizing a (possibly) multi-modal likelihood. All these defects result in a biased variance that may even worsen when the dimensionality of the model increases.

To obtain a robust and unbiased variance in a Gaussian process, one can either consider a fully Bayesian method, define priors and perform sampling in the hyperparameter space, or carry out resampling over the observation set. Despite their effectiveness, Bayesian optimization algorithms that use these techniques are less efficient than the classic version as they impose the overhead of creating multiple Gaussian processes.

Many problems contain non-stationary objective functions where the smoothness varies across the input domain. However, classical Bayesian optimization is not able to fully model this behaviour, as vanilla Gaussian process, GP with maximum likelihood estimate over its hyperparameters, uses a stationary covariance function in which the value of hyperparameters (length scales) are set for the entire search space. Solutions to this problem usually end up with multiple local Gaussian processes defined in either the hyper-parameter space or the underlying search space. Needless to say, performing inference for several additional Gaussian processes at each search step of the Bayesian optimization drastically reduces the efficiency of the method. Moreover, omitting the possible discrepancies between these local Gaussian processes might require additional effort.

Bayesian Optimization is primarily designed to work with deterministic objective functions. However, it is possible that observations gathered from a physical environment are polluted with noise. When the noise rate is

constant throughout the search space, Bayesian optimization is able to handle it with a small modification in its infill criterion [91]. However, this condition does not always hold; in many applications the noise variance is a function of the input data, but GPs can only deal with Gaussian white noise. The noise variance is mostly treated as a hyperparameter. Hence, the solution is similar to the ones for input dependent smoothness, e.g., in [112] a Gaussian process prior is used to model noise rate variations over the search space. This approach is advantageous because it can be integrated in any version of a Gaussian process although it imposes some computation overhead to the expensive optimization steps.

1.2.2 Why Neural Networks

Neural Networks (NN) are powerful function approximators with successful applications in a wide range of machine learning (ML) tasks, such as image classification [48, 79], visual recognition [78, 69, 215, 1], speech recognition [75, 74] and sequence modelling [28]. Based on approximation theory, any continuous function defined over a compact space can be approximated to arbitrary accuracy using standard feed-forward neural networks [32, 88] with a finite number of parameters, provided that the selected architecture for the neural network has enough expressive power.

Regression models such as random forests [31, 130, 66] and other ensemble methods [146] have the ability to provide uncertainty with their prediction. However, this uncertainty is rather heuristic, whereas the Bayesian framework provides a principled way to deliver the same quality. A Bayesian Neural Network is built by defining a prior over the weight space of a feed-forward neural network and then updating it in the light of observations.

Training a neural network usually consists of finding a single neural network (specified by the weights of the connections between neurons) able to represent the available data to sufficient accuracy. However, in a Bayesian

treatment, the goal is instead to infer a distribution over all possible models that express the available observations equally well and whose parameters comply with the prior assumptions. The term *Bayesian neural networks* refers to a set of neural networks sampled from this distribution. The connection between neural networks with random parameters and Gaussian processes is well-established, and the asymptotic equality of the posterior distribution represented by a fully connected neural network and Gaussian process is proven in the limit as the width or depth of the network goes to infinity [125, 143, 156]. Despite this similarity, neural networks have some desirable properties that could give them an advantage over Gaussian processes in building potentially more capable Bayesian optimization methods.

To begin with, as opposed to Gaussian processes (in which the number of parameters grows with the size of training set), neural networks have a fixed set of parameters. Hence, their inference time scales only linearly with the number of observations. Moreover, in neural networks the under-estimated variance problem, which was previously mentioned as being present in maximum-likelihood Gaussian processes, is easily rectified by applying Bayesian inference method under appropriate assumptions. Using Bayesian methods in neural networks leads to an expressive family of distributions with a rich multi-modal posterior. Last but not least, with neural networks it is possible to incorporate background knowledge, about the specifications of the problem, into the design of the model architectural. For example, the locality and hierarchical structure of the correlations in images suggest that a convolutional layer [124] is a better choice than a fully connected layer in an image classification problem. Accordingly, in circumstances where the noise (aleatoric uncertainty) and smoothness (covariance structure) are functions of the input (they are position dependent), this could potentially be included into the neural network architecture and addressed using only a single model.

1.3 Research Objectives

The overall goal of the work presented in this dissertation is to enrich the Bayesian Optimization framework by introducing algorithms that entail potential solutions to the problems imposed by using Gaussian processes in classical Bayesian Optimization. This research proposes several Bayesian optimization methods in which Neural Network models are used as a main or complementary component of the algorithm. The overall goal of this study requires the fulfilment of the following intermediate research objectives:

1. *Developing a new Bayesian Optimization method with a computational complexity lower than and a performance which is competitive to classical BO's.*

   The objective of this work is to replace the GP surrogate of the BO with neural network-based alternatives without compromising its performance. The direct implication of replacing Gaussian processes with Neural Network-based models is reducing the computational complexity of the surrogate building process. Moreover, since Gaussian process priors are quite different from the ones represented by Neural Networks (with finite number of hidden units), we expect the new Bayesian Optimization method to behave differently from their Gaussian process-based counterparts and aim at studying their differences.

2. *Developing a fast and simple neural network -based Bayesian Optimization method with an efficient surrogate building procedure.*

   Despite the theoretical advantages of Bayesian Neural Networks over Gaussian processes in terms of computational and space complexity, in practice the efficiency of their inference is highly dependent on their inference method. Traditionally, Markov Chain Monte Carlo (MCMC) sampling methods are the baseline Bayesian inference tech-

nique used in learning Bayesian neural networks. However, sampling methods are known to be slow and scale poorly as the dimensionality of the input increases. This work aims at using neural network based-surrogates with a fast training procedure that still fits into the Bayesian inference scheme.

3. *Enhanceing the classical Bayesian Optimization by improving on its infill criterion*

   The objective of this work is to enhance the performance of Bayesian Optimization via introducing an infill criteria which is more robust to the possible inaccuracies in the surrogate model. The commonly used infill criteria, such as Expected Improvement, are defined based on the the assumption that the most recent surrogate is a correct model of the underlying function and does not contain any mis-specification or inaccuracy. This is a strong assumption which is not always valid and since infill criteria do not include a mechanism to handle such an inaccuracies, their results are largely affected by them. This problem was first mentioned in [106] and has been largely ignored for a long time. In this work, we provide a solution to this problem using an infill criterion in which, in order to be robust to the inaccuracies in the surrogate, a neural network model is used to find infill points.

4. *Developing a fast and scalable inference for Bayesian Neural Network surrogates in Bayesian Optimization.*

   Variational inference is an optimization-based, scalable, fast alternative to the Monte Carlo sampling. However, while the latter guarantees an asymptotically exact recovery of the true posterior, the former provides a biased estimation of the underlying posterior. One of the factors that affect the quality of approximations produced by a variational inference method is the assumption that the designer makes on the family of the underlying distribution. This work presents a Bayesian Optimization algorithm in which a Variational Bayesian

neural network is used as surrogate. To be able to get as close as possible to the posterior, it is assumed that the posterior distribution over the neural network weights comes from a rich neural network-based distribution. In this objective, rather than replacing the Bayesian neural network surrogate of the Bayesian Optimization with a simpler model, the focus is on improving the efficiency and effectiveness of the inference method.

1.4 Major Contributions

1. We propose simple and fully Bayesian neural networks as alternatives to Gaussian processes in the Bayesian optimization framework. The computational complexity of the surrogate building process in the proposed BO is significantly lower than classical BO's. In this work, we use a sampling-based inference method, namely Monte Carlo sampling, to build Bayesian Neural Networks. The advantage of using Monte Carlo sampling is that they are able to produce asymptotically exact approximations to the posterior of the model parameters. Moreover, this study investigates the possibility of using an ensemble of Random Neural Networks, also known as Extreme Learning Machines, as a surrogate and introduces the empirical Expected Improvement as an infill criterion for ensemble-based surrogate models. As opposed to the analytical Expected Improvement, which is bound to a certain family of posterior distributions, the proposed infill criteria can be used with any posterior probability over a function space without deviating from the main definition of the Expected Improvement. The methods are compared against classical Bayesian Optimization on a set of benchmark continuous test functions. The experiments demonstrate that Bayesian optimizations with a Bayesian neural network surrogate perform better than or are at least competitive with classical Bayesian optimization on a number of test functions.

2. We propose a Bayesian Optimization algorithm using Bayesian Random Vector Functional-link Networks (BRVFL). These are shallow neural network models with a Bayesian final layer and randomly set values for the parameters of the first layer. Although the computational complexity of inference in Bayesian Neural Network is theoretically lower than GP's, in practise Monte Carlo-based inference methods are not scalable and need considerable amount of time to get a decent approximation of the posterior. In this research, we address this problem by reducing the number of parameters used in the inference. While in a Bayesian Neural Network we look for the posterior over the entire parameter space, in BRVFL the inference is recast as a Bayesian linear inverse problem over a subset of the model parameters. We describe a simple and effective initialization scheme for *relu* BRVFLs. To the best of our knowledge, this is the first time BRVFLs are used in a Bayesian optimization framework. We also investigate the possibility of utilizing a Bayesian ensemble of Random Vector Functional-link Networks formed using theoretically sound randomization technique. Proposed methods are compared to the state-of-art Bayesian optimization methods on a set of benchmark synthetic functions and hyperparameter optimization problems. These experiments show that BRVFL-based BO could outperform state-of-the-art state-of-art Bayesian optimization methods.

3. We propose a new infill criterion based on the notion of maximum *a posteriori* estimation. This contribution is based on the idea of conditional maximum likelihood in single-stage optimization [106] and extends it to enable it to be included in an infill criterion. The proposed acquisition function finds the expected optimum value of the target function using the Expected Improvement and then locates new infill points by computing a conditional maximum *a posteriori* solution over the joint space of the parameters of a neural network and location of the optimum. This process reduces the sensitivity

of the infill criterion to the inaccuracies in the surrogate model. Experiments with Benchmark hyperparameter optimization problems shows that in Bayesian Optimization methods with Bayesian Neural Network surrogates the proposed acquisition function is superior to Expected Improvement.

4. We propose a Bayesian optimization algorithm that uses Implicit Variational Bayesian Neural Networks (IVBANN) as surrogate. In this study, we utilize Neural Network-based generative models in a Variational Bayesian context to approximate the posterior over the parameters of the Neural Network-based surrogate in BO. Inference in the proposed surrogate is faster and more scalable than Monte Carlo sampling-based Bayesian Neural Networks. This is due to the use of a novel extension to the deep implicit distributions, that we introduce as Order-preserving Generative Models, with the ability to preserve order between the input (latent space) and output (underlying) distributions. In Bayesian Optimization, Order-preserving Generative Models enables us to get better approximations of the posterior distribution compared to non-order preserving deep implicit distribution. In a broader context, this new inference scheme is useful where one requires control over how to sample from a neural network-based generative model. This research shows that the new Bayesian Optimization method outperforms the state-of-the-art on a set of benchmark hyperparameter optimization problems.

1.5 Organization of the Thesis

The remainder of the dissertation is organized as follows. Chapter 2 covers the background work on Bayesian inference and Bayesian Optimization. Chapter 3 to 5 include the main contributions of this study. The focus of Chapter 3, 4 and 6 is on the enhancement of Bayesian Optimization

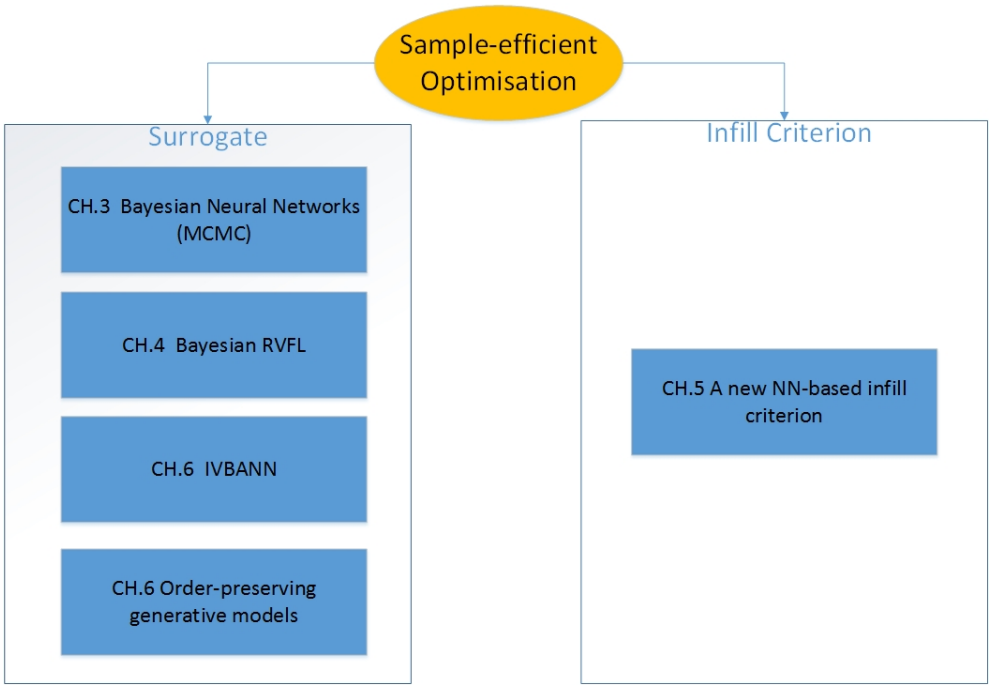


Figure 1.1: The overall structure of the thesis contributions.

by improving on its surrogate model, whereas Chapter 5 concerns with achieving this objective by introducing a new infill criterion. The overall organisation of the contributions of this research is shown in Figure 1.1

Chapter 2

Background and Literature Review

2.1 Introduction

Bayesian and sample-efficient global optimization has been a subject of study for many years. In this chapter we review the background material and existing work in this field. Section 2.2 provides a brief introduction to sample-efficient global optimization and its building blocks. Section 2.3 is devoted to the Bayesian modeling and inference for regression and covers both parametric and non-parametric models. In section 2.4, we introduce various existing possibilities for the acquisition function as a building block for Bayesian optimization. Finally, a conclusion is given in section 2.5.

2.2 Bayesian Optimization

For a long time Bayesian Optimization (BO) [150, 107, 163] has been the *de facto* approach for solving engineering optimization problems in which merely evaluating a solution carries significant cost [4, 100, 61, 109, 113, 39, 127, 102]. BO is a sequential strategy to find the global optima of a

real-valued and possibly *non-convex* function whose analytical form is not accessible, hence, it is a *black-box* and the only way of getting information about its surface is by sampling. It is assumed that, under *limited budget*, every evaluation of the function at suggested locations inflicts *non-negligible* costs, therefore, the goal is to find the optimum point with the least number of evaluations. In practice, we usually limit the number of function evaluations to a few hundreds and hope to get a near optimum solution.

In general, BO can be formulated as a standard function minimization problem:

$$\begin{aligned} &minimize && f(\boldsymbol{x}) \\ &subject\ to && \boldsymbol{x} \in \boldsymbol{\chi} \subset \mathbb{R}^d \end{aligned} \tag{2.1}$$

where $f : \boldsymbol{\chi} \to \mathbb{R}$ is the *real-valued* and possibly *nonlinear non-convex* function defined over a d dimensional space; $\boldsymbol{\chi}$ denotes a compact set of $\mathbb{R}^d$ and the ultimate goal of the optimization is to find a global minimum $y^* = f(\boldsymbol{x}^*)$ as well as its corresponding minimizer $\boldsymbol{x}^*$.

To address this problem efficiently, BO methods maintain a distribution over possible functions and uses this to select new points at which to make evaluations [151]. They include two essential elements. First, there is a Bayesian regression model which is built using the history of previously sampled points and includes our prior assumptions about the characteristics of the target function. This model acts as a cheap-to-evaluate *surrogate* of the expensive target function and usually gets updated whenever a new evaluated point is added to the history. The second element is a so-called *infill criterion* or *acquisition function* which is a heuristic to acquire the next sampling point. This function is a wrapper around the surrogate model's output and serves as a means to automatically balance exploration and exploitation. In the following sections, we elaborate on these two elements. First, the Bayesian treatment of statistical models is explained from different points of view. Then, the existing infill criteria for these models are covered.

2.3 Bayesian Regression

Suppose we have a dataset of the form $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n},\ \boldsymbol{x}_i \in \boldsymbol{\mathcal{X}}$ and $y_i \in \mathbb{R}$ which can also be denoted in compact form as $\{\boldsymbol{X},\ \boldsymbol{y}\}$ where $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_n]^\intercal$ is the $n \times d$ the design matrix and $\boldsymbol{y} = [y_1, ..., y_n]$ is the random vector of corresponding response values. The goal of regression is to find a parametrized mapping function $f : \boldsymbol{\mathcal{X}} \to \mathbb{R}$ that can map input points to their corresponding observations and also give accurate predictions for any set of unseen data. To build such a mapping, non-Bayesian methods only take the existing data into account with no assumption about the parameters of the prediction model. In a Bayesian approach, however, presumptions about the values of the mapping parameters are also included into the model. These presumptions typically consist of simple criteria such as a preference for smaller over large values for the parameters of the mapping function. Often it is assumed that observations are noisy and the data generating process has the following general form

$$y_i = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + \epsilon(\boldsymbol{x}_i) \tag{2.2}$$

where $\boldsymbol{x}_i \in \boldsymbol{\mathcal{X}}$ is the input vector, $f_{\boldsymbol{\theta}}(.)$ is a parametrized function that can take any linear or nonlinear form, $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ is the set of parameters for f, y_i denotes the function value at $\boldsymbol{x}_i$ and $\epsilon(.)$ is the noise function.

2.3.1 Distribution over Parameters

The rationale behind modelling the mapping function is the assumption that in the absence of noise the data generating process can be precisely described by a parametrized deterministic function of a certain class. Consequently, if the *observation*s are noisy, the resulted residuals can be assumed to be independently and identically distributed (i.i.d). A common choice of error model in this case is zero centred Normal distribution. Hence, output of $\epsilon(.)$ in 2.2 is independent of its input:

$$\epsilon(\boldsymbol{x}_i) \sim \mathcal{N}(0, \sigma_n^2) \tag{2.3}$$

where σ_n^2 is noise variance. For the sake of simplicity from here on we will refer to $\epsilon(\boldsymbol{x}_i)$ as ϵ_i.

In the above scenario, the goal is to find the most probable set of values for the function parameters. More ideally we might want to compute the credibility of all possible parameter values w.r.t. observations as well as our prior preference over possible parameters. To this end, we rewrite equation 2.2 as $y_i \mid \boldsymbol{x}_i \sim \mathcal{N}(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i),\ \sigma_n^2)$ by which it is possible to compute the probability of a single data point, given function parameters and noise variance. Subsequently, the conditional probability of the whole dataset, viewed as the *likelihood*, is computed as the product of individual probabilities:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) = \prod_{i=1}^{n} p(y_i \mid \boldsymbol{x}_i, \boldsymbol{\theta}, \sigma_n^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_n^2}} e^{-\frac{|y_i - f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)|^2}{2\sigma_n^2}} \tag{2.4}$$

When *likelihood* is used as credibility measure, the set of parameter values found by maximizing the equation 2.4 is called *maximum likelihood estimate* (MLE).

The model built by MLE can predict the existing dataset well. However, it does not guarantee that this regression model can produce good estimations for unseen data. The problem is particularly exacerbated if the dataset is small. The Bayesian treatment of this problem is to take our *prior* assumptions about characteristics of the target function into account. We have already imposed a part of this assumptions by choosing a specific class of parametric models. We can take one step further and define a prior over possible functions. This can be achieved indirectly by imposing a preference over the model parameters, which usually consists of favoring smaller parameter values so that the resulted model have a smoother output. To this end, a prior distribution is defined over the function parameters and is plugged into the model building process through *Bayes rule*. Having a *prior* over parameters, $p(\boldsymbol{\theta})$, we can apply the Bayes formula to compute their *posterior distribution*:

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) = \frac{p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) p(\boldsymbol{\theta})}{p(\boldsymbol{y} \mid \boldsymbol{X})} \tag{2.5}$$

$$p(\boldsymbol{y} \mid \boldsymbol{X}) = \int_{\boldsymbol{\theta}} p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) p(\boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} \tag{2.6}$$

where $p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2)$ is the *posterior probability* of the parameters, $p(\boldsymbol{\theta})$ is its *prior distribution* and $p(\boldsymbol{y} \mid \boldsymbol{X})$ is a *normalizing constant* a.k.a the *evidence* or *marginal likelihood*.

A common choice of prior is a zero mean Normal distribution:

$$p(\boldsymbol{\theta}) \ \sim \ \mathcal{N}(0, \boldsymbol{\Sigma}_p) \tag{2.7}$$

where $\boldsymbol{\Sigma}_p$ is the *covariance matrix* for the parameters, with non-diagonal entries usually set to zero.

If point estimates are satisfactory for the purpose, predictions can be performed using *maximum a posteriori* (MAP) model, i.e. finding the single set of parameters which best represents the data. This can be found by maximizing the posterior probability over parameter set $\boldsymbol{\theta}_{MAP} = \operatorname{argmax}_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2)$. However, the Bayesian way is to take into account all the posteriori probable models. Hence, for a new input point like $\hat{\boldsymbol{x}}$, a probability distribution over possible output values, a.k.a *posterior predictive distribution* (or *predictive distribution* for short), is computed by integrating out the model parameters:

$$p(\hat{y} \mid \hat{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) = \int_{\boldsymbol{\theta}} p(\hat{y} \mid \boldsymbol{\theta}, \hat{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) \, \mathrm{d}\boldsymbol{\theta} \tag{2.8}$$

Exact Inference

The availability of closed form expressions for the *posterior* and *predictive* distributions is very desirable as it eliminates the complications of performing numerical or approximate inference methods. Bayesian linear regression models are the most basic types of Bayesian models. Despite their limited expressiveness, they have a closed form posterior provided that a suitable prior distribution is defined over their parameters. This "proper" prior, a.k.a *conjugate*, is a distribution that when combined with the likelihood gives a posterior of the same form. In noise-free linear regression, f is simply a weighted sum of input variables $f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{x}^\intercal \boldsymbol{\theta}$. In

case of known noise variance σ_n^2 and a Gaussian prior, $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$, the parameters of the posterior distribution are computed as follows:

$$p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2, \boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p) \propto p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) p(\boldsymbol{\theta}) \tag{2.9}$$

$$\boldsymbol{\mu} = \boldsymbol{\Sigma}(\sigma_n^{-2} \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{\Sigma}_p^{-1} \boldsymbol{\mu}_p) \tag{2.10}$$

$$\boldsymbol{\Sigma} = (\sigma_n^{-2} \boldsymbol{X}\boldsymbol{X}^\intercal + \boldsymbol{\Sigma}_p^{-1})^{-1} \tag{2.11}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote the posterior mean and covariance matrix, respectively. Consequently, the predictive distribution is also analytically available:

$$\begin{aligned} p(\hat{y} \mid \hat{\boldsymbol{x}}, \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) &\sim \mathcal{N}(\mu(\hat{\boldsymbol{x}}), \sigma^2(\hat{\boldsymbol{x}})) \\ \mu(\hat{\boldsymbol{x}}) &= \hat{\boldsymbol{x}}^\intercal \boldsymbol{\mu} \\ \sigma^2(\hat{\boldsymbol{x}}) &= \hat{\boldsymbol{x}}^\intercal \boldsymbol{\Sigma} \hat{\boldsymbol{x}} \end{aligned} \tag{2.12}$$

The expressiveness of this simple linear model can be increased by projecting the input space to a higher order feature space using a set of basis functions. However, in order to have an analytic posterior the bases are required to be fixed functions, i.e. independent of $\boldsymbol{\theta}$ [211]. Therefore, for the many regression models which do not satisfy this criterion, an analytically tractable inference method does not exist.

Approximate Inference

In the absence of a closed form posterior distribution function, one could choose from a large set of posterior approximation methods. In this section, we will briefly go through some of the most widely used ones applicable to *Bayesian regression models*.

One approach to cope with analytically intractable posterior probabilities is to approximate them with a large set of representative samples [120]. Having a set of n samples drawn from the posterior probability of the model parameters, the predictive distribution can be approximated as

follows:

$$
\begin{aligned}
p(\hat{y} \mid \hat{\boldsymbol{x}}; \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) &= \int p(\hat{y} \mid \boldsymbol{\theta}, \hat{\mathbf{x}}, \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \\
&\approx \frac{1}{n} \sum_{i=1}^{n} p(\hat{y} \mid \boldsymbol{\theta}_i, \hat{\mathbf{x}}, \boldsymbol{y}, \boldsymbol{X}) \\
\boldsymbol{\theta}_i &\sim p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X}, \sigma_n^2)
\end{aligned}
\tag{2.13}
$$

To get a representative sample set, sampling methods known as *Markov chain Monte Carlo* (MCMC) are applicable. MCMC methods can be applied to sample from virtually any distribution. However, they are particularly useful when the distribution density is only available up to a normalizing constant. This is particularly advantageous in the task of approximating a posterior probability distribution, as it enables us to do so using only likelihood and prior probability, disregarding the awkward integral in its denominator.

A simple yet powerful MCMC method is *Metropolis Hastings* (MH). In this algorithm, the representative sample set is produced sequentially by taking a random walk through the parameter space. The distribution we want to sample from is usually called *stationary* or *invariant distribution*. The algorithm starts from an arbitrary point. Each step in the random walk consists of an attempt to move to a new randomly chosen adjacent point. This new point is generated from a *proposal distribution* which is *symmetric* and usually bell-shaped in order to give priority to small steps rather than large ones. The transition succeeds if the new point has a higher relative probability than the current position w.r.t the invariant distribution; otherwise its success chance depends on how low its relative probability is. MH algorithm steps are as follows:

The sequence of samples produced by MCMC usually needs some post processing. First, the initial samples, a.k.a the *Burn-in*, are not good representatives of the *stationary distribution* and should be discarded. If the proposal distribution has tunable parameters, the Burn-in samples are also used to tune them. Second, successive samples resulting from the random

Algorithm 1: Metropolis Hastings

Input: number of samples M, starting point $\boldsymbol{\theta_0}$, proposal distribution $Q(.)$, an unnormalized posterior distribution function $p(.)$

Result: a sequence of samples $\boldsymbol{\Theta}$

set $\boldsymbol{\Theta} \longleftarrow \{\}$

set $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta_0}$

for i =1,...,M **do**

  sample $\boldsymbol{\theta}' \sim Q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$

  with probability $min(1, \frac{p(\boldsymbol{\theta}'|\boldsymbol{y},\boldsymbol{X},\sigma_n^2)}{p(\boldsymbol{\theta}|\boldsymbol{y},\boldsymbol{X},\sigma_n^2)})$

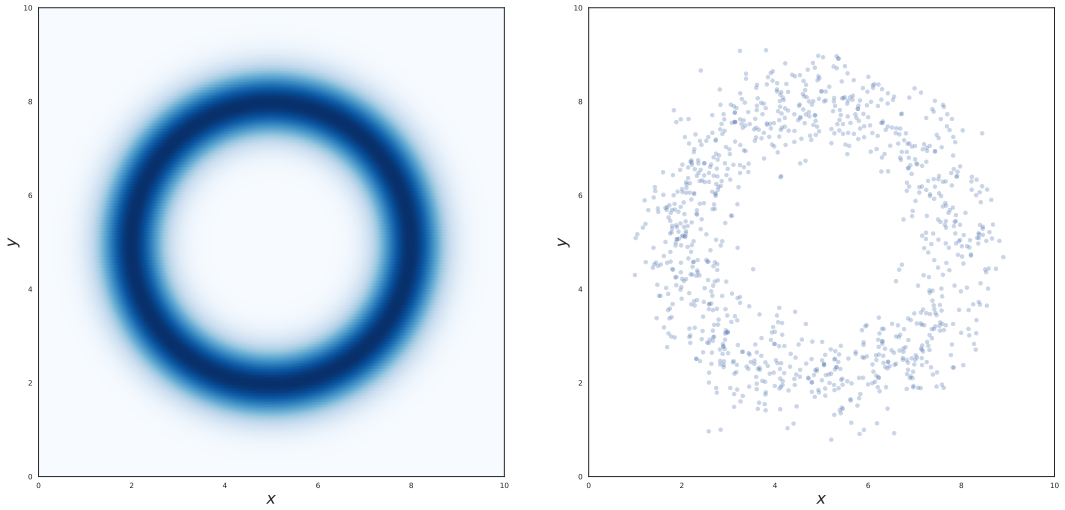
    set $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta}'$

  set $\boldsymbol{\Theta} \longleftarrow \{\boldsymbol{\theta}\}$

end

walk are not independent, whereas we need i.i.d draws from the posterior. To impose the i.i.d assumption a common practice is to only consider every kth sample and discard the rest.

The simplicity of MH comes with a price. It has a very slow convergence speed, staying in one region for a long time and producing highly correlated samples. This poor mixing as well as low acceptance rate are due to its random walk behavior which can be intensified if the posterior probability is high dimensional or has a complicated multivariate density function. An example of such a behavior is shown in Figure 2.2 where MH is applied to draw 1000 samples from a ring-shaped distribution (see Figure 2.1). As can be seen, as a result of poor mixing, MH has drawn many samples from the vicinity of each position before moving to a new region. Figure 2.3 corresponds to the sample values of the same sample-set across each dimension. It clearly shows how the low acceptance rate has caused the values of each variable to stay steady for long periods during the simulation. It is also worth mentioning that the cost per independent sample for MH is $O(n^2)$ [86], where n denotes the number of observations, which could be considered high for a method with all these drawbacks.



(a) (b)

Figure 2.1: (a) A ring-shaped toy distribution and (b) 1000 samples directly drawn from it via *Inverse Transform Sampling*. The sample set is a very good representative of the underlying distribution. However, this is only possible if the inverse of the underlying cumulative distribution function (or a good approximation to it) is available.

A solution to MH shortcomings is presented in *Hamiltonian Monte Carlo* (HMC) [157, 10, 43] where the random walk behavior is greatly reduced in favour of deterministic, gradient-guided movements with q physical interpretation originating in Newton's law of motion. Furthermore, in HMC the cost per independent sample is $O(n^{5/4})$. HMC samples by imitating the movement of a frictionless puck that moves freely over an uneven surface. In Hamiltonian dynamics, a particle is defined by its position $\boldsymbol{\theta}$ and momentum $\boldsymbol{r}$. Movement is driven by the particle's potential $U(\boldsymbol{\theta})$ and kinetic energies $K(\boldsymbol{r})$ which are summed up in the *Hamiltonian function* $H(\boldsymbol{\theta}, \boldsymbol{r})$ to form the particle's total energy $H(\boldsymbol{\theta}, \boldsymbol{r}) = U(\boldsymbol{\theta}) + K(\boldsymbol{r})$. This quantity is constant and known. As the puck moves through the time, the changes to its

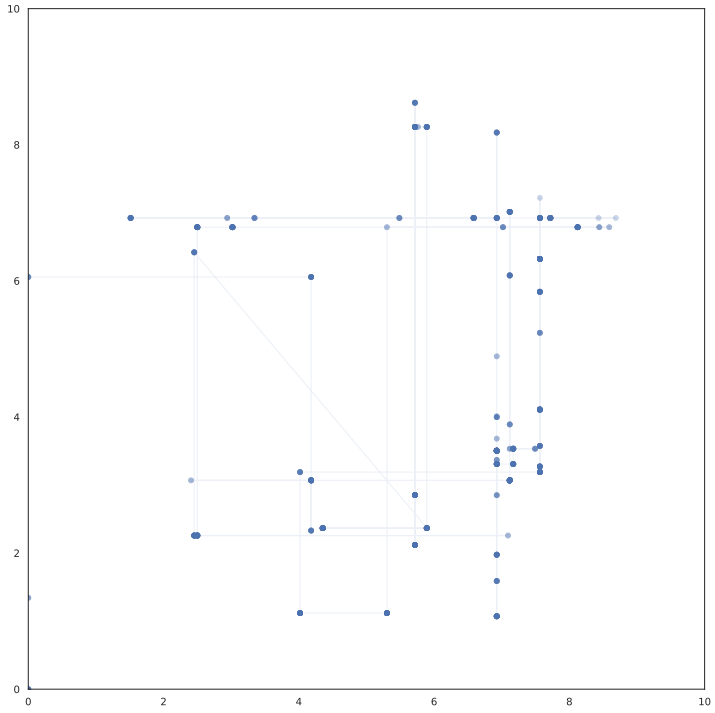


Figure 2.2: Set of 1000 samples, including 100 *burn-in*s, generated by MH from the toy ring-shaped distribution of the Figure 2.1. The high rejection rate has caused the sampler to cover only a small portion of the state-space and produce many overlapping samples.

potential and kinetic energies are governed by the Hamiltonian equations:

$$\begin{aligned} \frac{d}{dt}\theta_i &= \frac{\partial H(\boldsymbol{\theta}, \boldsymbol{r})}{\partial r_i} = \frac{\partial K(\boldsymbol{r})}{\partial r_i} \\ \frac{d}{dt}r_i &= \frac{\partial H(\boldsymbol{\theta}, \boldsymbol{r})}{\partial \theta_i} = \frac{\partial U(\boldsymbol{\theta})}{\partial \theta_i} \end{aligned} \tag{2.14}$$

where θ_i is the i^{th} component of $\boldsymbol{\theta}$.

The "Hamiltonian" can be considered as the negative log of a joint probability distribution over position and momentum, and HMC uses the Hamiltonian dynamics to sample from this distribution:

$$p(\boldsymbol{\theta}, \boldsymbol{r}) = e^{-H(\boldsymbol{\theta}, \boldsymbol{r})} \tag{2.15}$$

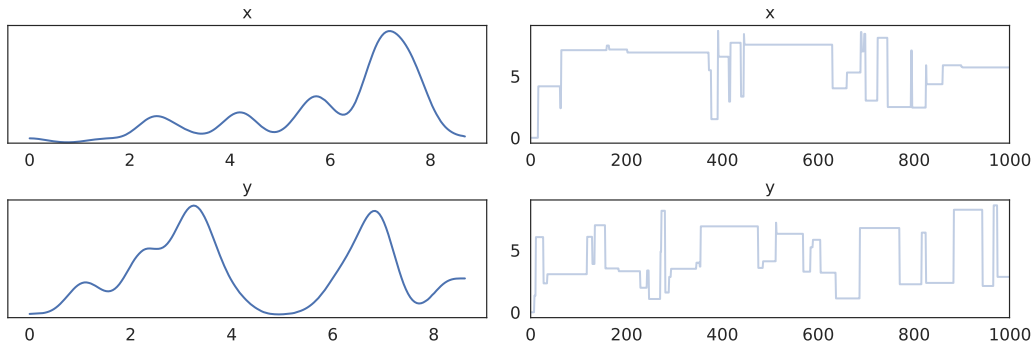


Figure 2.3: Sample values drawn using MH from a ring-shaped distribution across each dimension. (Left) Kernel density plot of the sampled values (Right) Sequence of values drawn during the simulation. It is clear that the chain is poorly mixed, for example, x values are concentrated around 6.5 to 7.5 (upper left) and have not changed for long periods of time (upper right). Similar phenomenon could be seen in y values.

In sampling from a posterior probability, $p(\boldsymbol{\theta} \mid \mathcal{D})$, the potential energy function, $U(.)$, is proportional to the target probability density function i.e. minus the log probability density of the posterior distribution over $\boldsymbol{\theta}$, whereas the kinetic energy function, $K(.)$, represents the negative log of an auxiliary distribution function defined over the momentum. The kinetic energy function is selected from a set of possible options depending on the characteristics of the target distribution [10]. This choice has a significant effect on the properties of the sampler [132]. The 'vanilla' HMC uses an *Euclidean-Gaussian* Kinetic energy:

$$K(\boldsymbol{r}) = \boldsymbol{r}^\intercal \boldsymbol{M}^{-1} \boldsymbol{r} + \log |\boldsymbol{M}| + const \tag{2.16}$$

where $\boldsymbol{M}$, the *mass matrix*, is usually an identity matrix. The Hamiltonian in the 'vanilla' HMC is of the following form

$$-\log\ p(\boldsymbol{\theta}, \boldsymbol{r}) \propto H(\boldsymbol{\theta}, \boldsymbol{r}) = -\log\ p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})\ +\ \frac{1}{2} \boldsymbol{r}^\intercal \boldsymbol{M}^{-1} \boldsymbol{r} \tag{2.17}$$

where $p(\boldsymbol{\theta}, \boldsymbol{r})$ denotes the unnormalized joint probability of position and momentum. Another popular choice of the energy function is the *Riemannian-*

Gaussian energy [67, 68] which is more suitable if the target distribution has a non-Gaussian form.

HMC has two important parameters, namely ϵ and L, that control the step-size and trajectory length of each move, respectively. Similar to MH, the algorithm starts from an arbitrary position $\boldsymbol{\theta}_0$. At the onset of each iteration, the momentum, which represents the direction of the next move, is randomly drawn from a Gaussian $\boldsymbol{r} \sim \mathcal{N}(0, \boldsymbol{M})$. Then, to reach the next proposal point, L steps are taken by sequentially updating the momentum by moving in the direction of the gradient of log posterior. Finally, an MH accept/reject step decides the acceptance of the new proposal.

Pseudocode for HMC is presented in Algorithm 2 where $\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$ denotes the negative gradient of the logarithm of posterior probability of $\boldsymbol{\theta}$. In the algorithm 2, the set of actions between two half-step changes to the momentum variable inclusive are *leapfrog steps*. A single *leapfrog* step consists of a half-step update to the momentum, followed by a full-step update to the position variable and finally another half-step update to the momentum variable:

$$\boldsymbol{r} = \boldsymbol{r} - \frac{\epsilon}{2} \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) \tag{2.18}$$

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \epsilon \boldsymbol{r} \tag{2.19}$$

$$\boldsymbol{r} = \boldsymbol{r} - \frac{\epsilon}{2} \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta})$$

Elimination of the random walk behavior gives the HMC the power to produce distant proposals. However, its efficiency is highly dependent on a good choice of step size, ϵ, and trajectory length, L, which must be hand tuned. This can be tricky as well as tedious and usually requires some initial experiments and a bit of expertise. Moreover, it is problem-specific. Therefore, any change to the shape of posterior probability density function (e.g. due to availability of new observations) invalidates the currently selected values of ϵ and L. Nevertheless, trajectory length has a significant impact on the efficiency of the sampler – a too small value for L leads to a highly correlated chain, whereas a too large value makes the sampling

Algorithm 2: Hamiltonian Monte Carlo

Input: number of samples M, step size ϵ, trajectory length L, starting point $\boldsymbol{\theta_0}$, potential energy function $U(.)$

Result: a sequence of samples $\boldsymbol{\Theta}$

Set $\boldsymbol{\Theta} \longleftarrow \{\boldsymbol{\theta_0}\}$

for i =1,...,M **do**

  sample $\boldsymbol{r}_{current} \sim \mathcal{N}(0, \boldsymbol{M})$

  set $\boldsymbol{\theta}_{current} \longleftarrow \boldsymbol{\theta}_{i-1}$, $\boldsymbol{\theta}_{proposal} \longleftarrow \boldsymbol{\theta}_{i-1}$, $\boldsymbol{r}_{proposal} \longleftarrow \boldsymbol{r}_{current}$

  /* A half step for momentum at the begining */

  $\boldsymbol{r}_{proposal} \longleftarrow \boldsymbol{r}_{proposal} - \frac{\epsilon}{2}\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}_{proposal})$

  /* full steps for position and momentum */

  for j=1,...,L **do**

    $\boldsymbol{\theta}_{proposal} \longleftarrow \boldsymbol{\theta}_{proposal} + \epsilon \boldsymbol{r}_{proposal}$

    if $j < L$ **then**

      $\boldsymbol{r}_{proposal} \longleftarrow \boldsymbol{r}_{proposal} - \epsilon \nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}_{proposal})$

    end

  end

  /* A half step for momentum at the end */

  $\boldsymbol{r}_{proposal} \longleftarrow \boldsymbol{r}_{proposal} - \frac{\epsilon}{2}\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}_{proposal})$

  /* MH accept/reject step */

  With probability $min\big(1, \frac{p(\boldsymbol{\theta}_{proposal}, \boldsymbol{r}_{proposal})}{p(\boldsymbol{\theta}_{current}, \boldsymbol{r}_{current})}\big)$:

    Set $\boldsymbol{\theta}_{current} \longleftarrow \boldsymbol{\theta}_{proposal}$

  Set $\boldsymbol{\Theta} \longleftarrow \boldsymbol{\Theta} \cup \{\boldsymbol{\theta}_{current}\}$

end

process unreasonably long by taking extra needless steps.

A solution to this problem is given by *No U Turn Sampler* (NUTS) [86] in which the trajectory length is adaptively set. To this end, rather than setting L to a prespecified number, the leapfrog steps are taken up until the trajectory loops back on itself. More precisely, starting from a specific

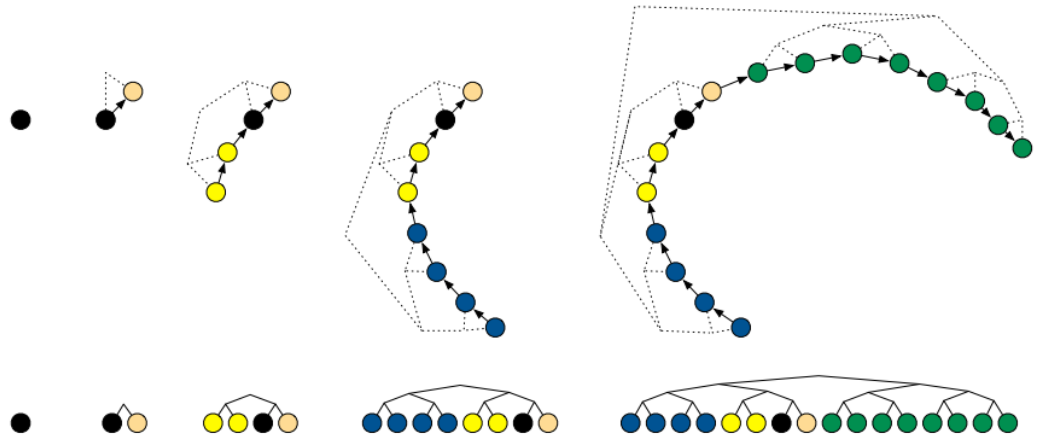


Figure 2.4: Example of incremental trajectory building process of NUTS in a *2d* space (figure adopted from [86]).

initial position, the trajectory is built incrementally by randomly flipping the sign of gradient and taking 2^j leapfrog steps, where j denotes the increment number, i.e 2^j steps are added to the beginning or end of the current trajectory. This imitates going back or forth through the fictitious time by 2^j units per increment. The process is stopped when the distance between the beginning and end of the trajectory does not increase any more. This is detected by monitoring the derivative of half the squared distance between the two ends of the trajectory w.r.t time, which is in turn equal to the dot product between the momentum vector and the vector from the initial position to the current position:

$$\frac{d}{dt}\frac{(\boldsymbol{\theta}_{current} - \boldsymbol{\theta}_0)^2}{2} = (\boldsymbol{\theta}_{current} - \boldsymbol{\theta}_0)\frac{d}{dt}(\boldsymbol{\theta}_{current} - \boldsymbol{\theta}_0) = (\boldsymbol{\theta}_{current} - \boldsymbol{\theta}_0)\boldsymbol{r}_{current} \tag{2.20}$$

The set of traversed states are leaves of an implicit balanced binary tree which is built in a depth first manner (see Figure 2.4). The proposal position is then drawn with uniform probability from this set of candidate positions. To reduce the memory usage only a subset of these states are kept in the memory. The proposal position is then drawn with uniform proba-

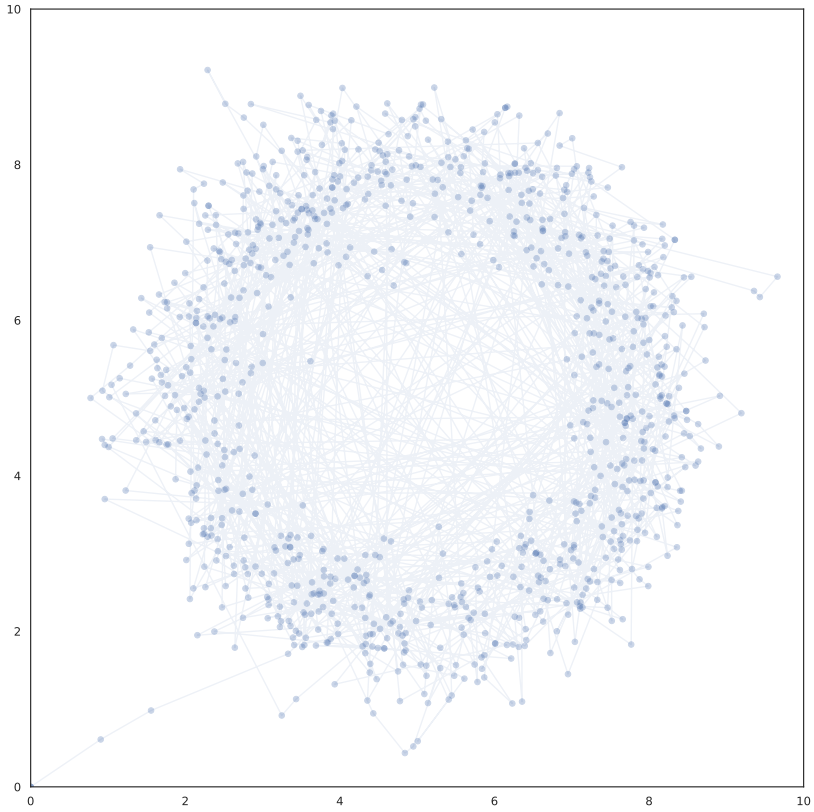


Figure 2.5: Set of 1000 samples, including 100 *burn-in*s, generated by NUTS from the toy ring-shaped distribution of the Figure 2.1. Compared with MH, the sample set produced by NUTS is a much better representative of the underlying distribution.

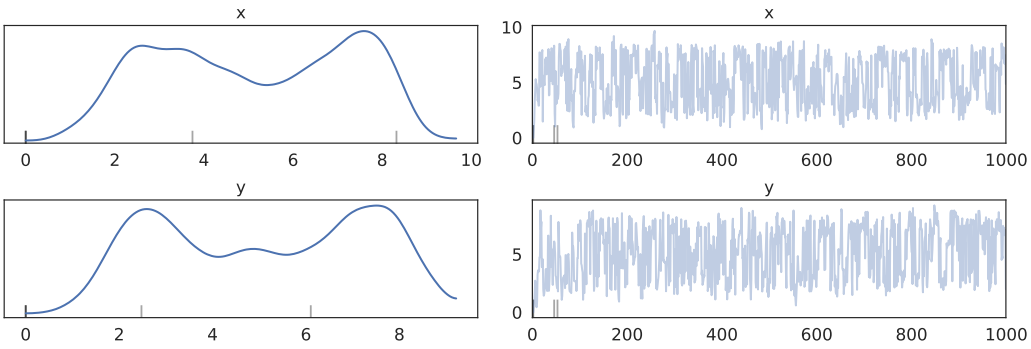


Figure 2.6: Sample values drawn using NUTS from a ring-shaped distribution across each dimension. (Left) Kernel density plot of the sampled values (Right) Sequence of values drawn during the simulation. The resulting chain is well-mixed (samples are iid), thanks to the NUTS's ability to adaptively set the trajectory length and use the gradient information.

bility from this set of candidate positions provided that their probability is higher than a certain amount. NUTS is capable of producing arbitrarily long trajectory lengths which drastically reduces the dependence between consecutive samples (see Figure 2.5 and Figure 2.6).

MCMC methods are able to produce asymptotically exact approximations, however, they are computationally expensive. Even HMC, in which the sampling is guided by gradient information, could become impractically slow in data intensive problems, simply because evaluation of the gradients of the potential energy requires going through the entire dataset. Although it is possible to obtain noisy estimates of the gradients via subsampling [25, 5, 23], the resulting approximation could be non-negligibly inaccurate [9].

A widely used alternative strategy to MCMC sampling is *Variational Inference* (VI) [203, 12, 60, 176]. Unlike MCMC, which results in an empirical distribution, in VI the approximating distribution, also known as *Variational Distribution*, is selected from a known family of densities defined over the latent space. Restricting the space of approximating solutions to a

certain family makes it possible to represent the inference as a variational optimization problem, i.e. optimization over a functional space, where the goal is to find the closest member of the family to the true posterior. By far, the most popular closeness measure, also called *variational objective*, used in VI is the *Kullback-Leibler* (KL) divergence. The VI problem is defined as:

$$q^*(\boldsymbol{\theta}) = \underset{q(\boldsymbol{\theta})\in\Omega}{\operatorname{argmin}} \mathrm{KL}[\, q(\boldsymbol{\theta}) \mid p(\boldsymbol{\theta} \mid \mathcal{D}) \,] \tag{2.21}$$

where $q(\boldsymbol{\theta})$ denotes the variational distribution, Ω is the restricted family of distributions, $p(\boldsymbol{\theta} \mid \mathcal{D})$ is the posterior probability of the model parameters and $\mathrm{KL}[\, . \mid . \,]$ is the KL divergence:

$$\mathrm{KL}[\, q(\boldsymbol{\theta}) \mid p(\boldsymbol{\theta} \mid \mathcal{D}) \,] = \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \mathcal{D})} d\boldsymbol{\theta} \tag{2.22}$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})}[\, \log q(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta} \mid \mathcal{D}) \,] \tag{2.23}$$

From (2.23) it is clear that minimizing the KL distance of (2.21) requires evaluation of the expectation of $\log p(\boldsymbol{\theta} \mid \mathcal{D})$ under the variational distribution. However, this evaluation entails computing the marginal likelihood (2.5) which we have deemed to be impractical (otherwise we would have been better off using an exact inference method). Fortunately, VI has a workaround to this problem.

Let us rewrite the KL in (2.22) with a minor change: we multiply the numerator and denominator of the fraction by the marginal likelihood $p(\mathcal{D})$:

$$\begin{aligned}\mathrm{KL}[\, q(\boldsymbol{\theta}) \mid p(\boldsymbol{\theta} \mid \mathcal{D}) \,] &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta} \mid \mathcal{D})} \frac{p(\mathcal{D})}{p(\mathcal{D})} d\boldsymbol{\theta} \\ &= \int q(\boldsymbol{\theta}) \log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}, \mathcal{D})} d\boldsymbol{\theta} + \log p(\mathcal{D})\end{aligned}$$

then we can rearrange it to the following form:

$$p(\mathcal{D}) - \mathrm{KL}[\, q(\boldsymbol{\theta}) \mid p(\boldsymbol{\theta} \mid \mathcal{D}) \,] = \int q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} \tag{2.24}$$

Clearly, the negative of the right hand side in (2.24) is equal to the KL distance between the variational distribution and the posterior, albeit up to a constant. Therefore, rather than a direct minimization of KL, the same results can be achieved by maximizing the right hand side of (2.24) which is a lower bound to the marginal likelihood, hence, known as the *Evidence Lower Bound* (ELBO):

$$p(\mathcal{D}) \geqslant \mathcal{L}(q, \mathcal{D}) = \int q(\boldsymbol{\theta}) \log \frac{p(\boldsymbol{\theta}, \mathcal{D})}{q(\boldsymbol{\theta})} d\boldsymbol{\theta} \tag{2.25}$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})}[\log p(\boldsymbol{\theta}, \mathcal{D}) - \log q(\boldsymbol{\theta})] \tag{2.26}$$

$$= \mathbb{E}_{q(\boldsymbol{\theta})}[\log p(\mathcal{D} \mid \boldsymbol{\theta}) - \mathbb{E}_{q(\boldsymbol{\theta})}[\log q(\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})]] \tag{2.27}$$

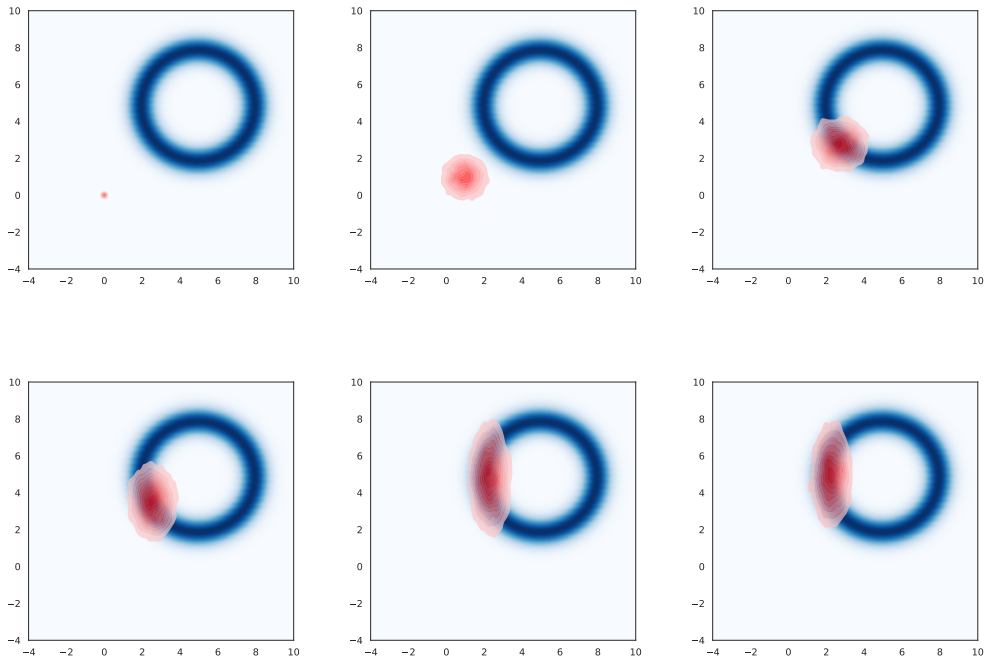


Figure 2.7: Incremental minimization of the KL distance between a Gaussian variational distribution and a ring-shaped target distribution.

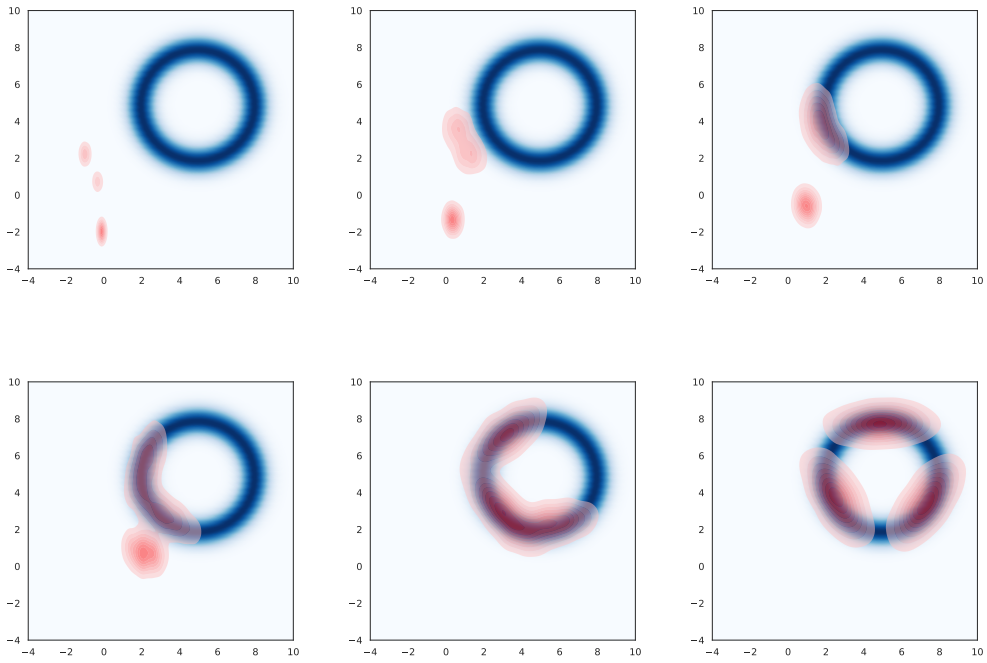


Figure 2.8: Incremental minimization of the KL distance between the variational distribution and a ring-shaped target distribution. The variational distribution is a Gaussian Mixture density with three components.

Restricting to a class of *variational distributions* results in a biased estimation, however, this is the cost we pay to achieve tractability. As suggested in [11] the variational distribution can be specified in two possible ways. First, it can be chosen from the family of parametric distributions, $q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$. In this case, the ELBO is a function of $\boldsymbol{\omega}$. If the variational distribution is chosen from conditionally conjugate exponential family, then the expectations $\mathbb{E}_{q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}[\,.\,]$ are analytically available [218], otherwise any off-the-shelf optimization technique, such as gradient-based methods, can be used to optimize the ELBO w.r.t the parameters of the approximating distribution, $\mathcal{L}(\boldsymbol{\omega}, \mathcal{D})$.

Some of the most widely used parametric variational distributions con-

sist of *Multivariate Gaussian*s [136, 161, 118, 196, 114, 160], *Finite Mixture of distribution*s [144, 29, 204, 149, 65], and the ones based on invertible Neural networks, a.k.a *Neural Flow*s such as models built using *Normalizing Flow* [180, 137, 8, 119], *Masked Autoregressive Flow* [165], *Block Neural Autoregressive Flow* [34] and *Neural Autoregressive Flow*[90]. In the end, it is worth mentioning that VI could also be performed using implicit generative models [153, 198, 166]. Figure 2.7 and Figure 2.8 show the VI process using a Gaussian and Gaussian mixture model variational distributions, respectively.

2.3.2 Distribution over functions

Another way of looking into the regression model of equation 2.2 is to assume that the mapping function f does not have enough expressiveness. Therefore, the function's output is polluted by modelling error which is not independent any more [30, 91]. This dependence shows up in the error term in the form of off-diagonal elements of the covariance matrix:

$$\epsilon(\boldsymbol{X}) \sim \mathcal{N}(0, \mathbf{K} + \sigma_n^2 \mathbf{I}) \tag{2.28}$$

where σ_n^2 is the noise variance; $\mathbf{I}$ is identity matrix; $\mathbf{K}$ is a $n \times n$ covariance matrix with elements $k_{i,j}$ representing the correlation between modelling errors in $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ input points. The k_{ij} values are computed using a parameterized covariance function $k(.,.)$.

Having two parameterized terms in the regression model is undesirable and makes the model-building task complicated. Furthermore, to the extent that the error is modelled accurately, using a simplistic mapping function (e.g. replacing it with a constant) suffices. In this way, we can remove the mapping function and express the regression model fully in terms of the error model $\boldsymbol{y} \sim \mathcal{N}(0, \mathbf{K} + \sigma_n^2 \mathbf{I})$. Alternatively, the same result is obtained by considering a GP prior over a set of n latent variables $\mathbf{f} = \{f_1, ..., f_n\}^\intercal$ corresponding to the function value for the input points $\boldsymbol{X}$ as well as an

independent likelihood:

$$p(\mathbf{f} \mid \boldsymbol{X}) \sim \mathcal{N}(0, \mathbf{K}) \qquad p(\boldsymbol{y} \mid \mathbf{f}) = \mathcal{N}(\mathbf{f}, \sigma_n^2 \mathbf{I}) \tag{2.29}$$

Subsequently, the likelihood can be written as follows:

$$p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid \mathbf{K} + \sigma_n^2 \mathbf{I} \mid^{\frac{1}{2}}} \exp(-\frac{1}{2} \boldsymbol{y}^\intercal (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{y}) \tag{2.30}$$

in which σ_n^2 is the noise variance and $\boldsymbol{\theta}$ is the vector of the parameters of the covariance function. In the absence of prior knowledge about these parameters, a good estimate for their values is given by maximizing the likelihood function. Equivalently, log-likelihood can be used for the same purpose while numerically being more convenient:

$$\log\ p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n^2) = -\frac{n}{2} \log(2\pi) - \frac{1}{2} \log(|\mathbf{K} + \sigma_n^2 \mathbf{I}|) - \frac{1}{2} \boldsymbol{y}^\intercal (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{y} \tag{2.31}$$

Making predictions for a new set of m data points, say $\hat{\boldsymbol{X}}$, starts by calculating its joint probability with previously observed points $p(\boldsymbol{y}, \hat{\boldsymbol{y}})$:

$$\begin{bmatrix} \boldsymbol{y} \\ \hat{\boldsymbol{y}} \end{bmatrix} \sim N\left(0\ , \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{k} \\ \mathbf{k}^\intercal & \hat{\mathbf{K}} + \sigma_n^2 \mathbf{I} \end{bmatrix} \right) \tag{2.32}$$

This is equal to forming an augmented covariance matrix by computing $\mathbf{k}$, the covariance between error at new and old data points, as well as $\hat{\mathbf{K}}$, the covariance matrix for new data point themselves. The only unknown in the equation 2.32 is $\hat{\boldsymbol{y}}$, i.e the vector of function values for the new data points. Substituting the augmented covariance matrix in equation 2.31, an augmented log-likelihood is acquired.

The maximum of this function and its curvature give us the mean and variance for predictive distribution respectively. To this end, we take the first and second derivatives of the augmented log-likelihood w.r.t $\hat{\boldsymbol{y}}$, set them to zero and solve for $\hat{\boldsymbol{y}}$ [106]. Consequently, the resulting equations

for the parameters of the Gaussian predictive distribution are computed as follows:

$$\begin{aligned} \hat{\boldsymbol{\mu}} &= \hat{\mathbf{k}}^\intercal (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \boldsymbol{y} \\ \hat{\boldsymbol{\Sigma}} &= \hat{\mathbf{K}} - \hat{\mathbf{k}}^\intercal (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \hat{\mathbf{k}} \end{aligned} \tag{2.33}$$

where $\hat{\mu}$ and and $\hat{\Sigma}$ are predictive mean and covariance; $\hat{\mathbf{K}}$ is a $m \times m$ matrix covariance between points in $\hat{\boldsymbol{X}}$; $\hat{\mathbf{k}}$ is a $m \times n$ matrix in which each $\hat{k}_{ij}$ is $k(\epsilon(\hat{\boldsymbol{x}}_i), \epsilon(\boldsymbol{x}_j)),\ \hat{\boldsymbol{x}}_i \in \hat{\boldsymbol{X}}, \boldsymbol{x}_j \in \boldsymbol{X}$. In practice, Cholesky decomposition, an efficient method for solving systems of linear equations, is applied to simplify the matrix inversions of equation 2.33. The resulting algorithm is shown below [211].

Algorithm 3: Predictive mean, variance and log-likelihood for Gaussian Process Regression

Input: $\boldsymbol{X}$, $\mathbf{y}$(observations), k(.,.) (covariance function), σ_n^2 (noise variance), $\hat{\boldsymbol{x}}$ (new data point)

Result: $\hat{\mu}$ (predictive mean), $\hat{\sigma_n^2}$ (predictive variance), logp (log-likelihood)

Set L = cholesky($\mathbf{K} + \sigma_n^2 \mathbf{I}$)

Set $\boldsymbol{\alpha}$ = $L^\intercal \setminus (L \setminus \boldsymbol{y})$

Set $\hat{\mu}$ = $\hat{\mathbf{k}}^\intercal \boldsymbol{\alpha}$

Set $\mathbf{v} = L \setminus \hat{\mathbf{k}}$

Set $\hat{\sigma}^2 = k(\hat{\boldsymbol{x}}, \hat{\boldsymbol{x}}) - \mathbf{v}^\intercal \mathbf{v}$

Set $\log p(\boldsymbol{y} \mid \boldsymbol{X}) = -\frac{1}{2} \boldsymbol{y}^\intercal \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$

return $\hat{\mu}, \hat{\sigma}^2, \log p(\boldsymbol{y} \mid \boldsymbol{X})$

Covariance Functions

The performance of a GP is highly affected by the choice of its covariance function. However, when the mean is set to zero, as in our case, the covariance function is the only element that specifies a GP and carries our

assumptions about the behavior of the underlying function. A multitude of covariance functions exist in the literature. Here, we follow [211] to highlight a few of them.

One of the most widely used covariance functions is the *squared exponential*(SE), which is a stationary covariance function of the form

$$k_{SE}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \exp(-r^2(\boldsymbol{x}, \boldsymbol{x}')), \qquad r^2(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{d} \frac{(x_i - x_i')^2}{2\theta_i^2} \tag{2.34}$$

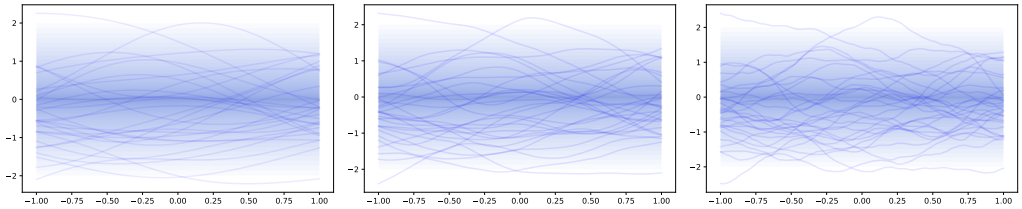
where x_i is the value at i^{th} dimension of the vector $\boldsymbol{x}$, $\theta_0 \in \mathbb{R}^+$ is the scaling factor denoting the variance of the function values and $\theta_i \in \mathbb{R}^+$ is the characteristic length scale of the i^{th} dimension, specifying the smoothness of the function in that direction. Due to having a separate length-scale per dimension, this covariance function allows for automatic relevance determination (ARD), however, setting $\theta_i = \theta_j,\ 0 < i, j \geq d$ leads to the isotropic version of SE kernel. This covariance function is unrealistically smooth and not appropriate in many applications. This has given rise to a generalization of this kernel, namely *Matérn class*, with further parameters that provide more control over the function smoothness:

$$k_{Matern}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} r(\boldsymbol{x}, \boldsymbol{x}') \right)^{\nu} K_\nu \left(\sqrt{2\nu} r(\boldsymbol{x}, \boldsymbol{x}') \right) \tag{2.35}$$

where $K_\nu(.)$ is the modified Bessel function and ν is a positive smoothing parameter. Similar to SE covariance function, depending on how we treat the length-scales, Matérn class kernels can be either isotropic or ARD. From this family the ones with $\nu = 3/2$ and $\nu = 5/2$ are more frequently used in the machine learning community:

$$k_{Matérn_{3/2}}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \left(1 + \sqrt{3} r(\boldsymbol{x}, \boldsymbol{x}') \right) \exp \left(- \sqrt{3} r(\boldsymbol{x}, \boldsymbol{x}') \right) \tag{2.36}$$

$$k_{Matérn_{5/2}}(\boldsymbol{x}, \boldsymbol{x}') = \theta_0 \left(1 + \sqrt{5} r(\boldsymbol{x}, \boldsymbol{x}') + \frac{5}{3} r^2(\boldsymbol{x}, \boldsymbol{x}') \right) \exp \left(- \sqrt{5} r(\boldsymbol{x}, \boldsymbol{x}') \right) \tag{2.37}$$



(a) (b) (c)

Figure 2.9: Samples from zero-centered GP priors with a) SE b) Matern$_{\nu=3/2}$ and c) Matern$_{\nu=5/2}$ covariance functions. GPs with SE covariance function entail much stronger smoothness assumption than the ones that use Matern$_{\nu=3/2}$ or Matern$_{\nu=5/2}$. The degree of smoothness provided by SE is very high, whereas Matern covariance functions are used to model functions that are less smooth.

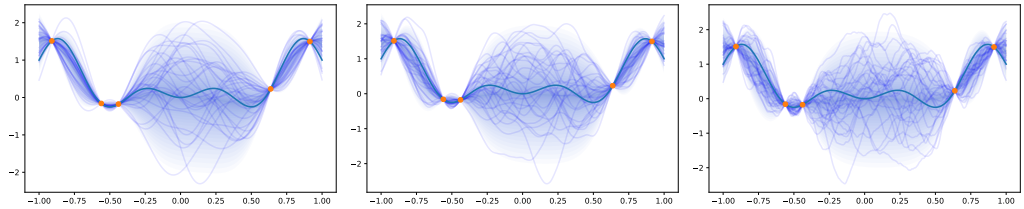
Another extension to the SE covariance function is the *Rational Quadratic* kernel

$$k_{RQ} = (1 + \frac{||\boldsymbol{x} - \boldsymbol{x}'||^2}{2\alpha\theta}) \tag{2.38}$$

in which α is the scale mixture parameter, and $\theta \in \mathbb{R}^+$ is the characteristic length-scale.

2.4 Infill Criteria

The BO task consists of iteratively building a regression model w.r.t the observed data, finding a new promising point using the regression model, evaluating the new point with the expensive objective function and finally adding it to the set of observed points. Thus far, we have reviewed the methods to build a Bayesian regression model. The next step is to define a basis, a.k.a infill criterion, upon which the next sampling point is selected. As we deal with Bayesian models, infill criteria can benefit from using parameters of a predictive distribution instead of only point estimates. This is mostly useful to create a balance between local and global search, a.k.a



(a) (b) (c)

Figure 2.10: Samples from the posterior of a zero-centered GP with a) SE b) Matern$_{\nu=3/2}$ and c) Matern$_{\nu=5/2}$ covariance functions given a set of five points from a toy sinusoidal function.

exploitation and exploration. Below we follow author [106] and [58] to explain some of the well-known infill criteria. Unless specified, all these functions are based on the assumption of a Gaussian predictive distribution.

2.4.1 Statistical Lower Bound

Recalling that our problem is a minimization problem, the next point to sample can be simply found by minimizing the sum of prediction mean and standard deviation. To better control the rate of exploration we can weight the standard deviation with a positive coefficient. The resulting criterion, the Statistical Lower Bound a.k.a Upper Confidence Bound (UCB), has the following form:

$$UCB(\boldsymbol{x}) = \mu(\boldsymbol{x}) - k\,\sigma(\boldsymbol{x}),\ k > 0 \tag{2.39}$$

in which $\mu(\boldsymbol{x})$ denotes the (mean) prediction of the regression model at $\boldsymbol{x}$, $\sigma(\boldsymbol{x})$ is its standard deviation, and k controls the exploration/exploitation rate. By setting k to a small value, the next point is very likely to be close to the previous ones. On the other hand, setting k to large values makes the effect of $\mu(\boldsymbol{x})$ negligible, consequently, the search becomes similar to pure exploration. Using this criterion, the important question is how to pick a proper value for k. Unfortunately, this is not clear in advance and depends

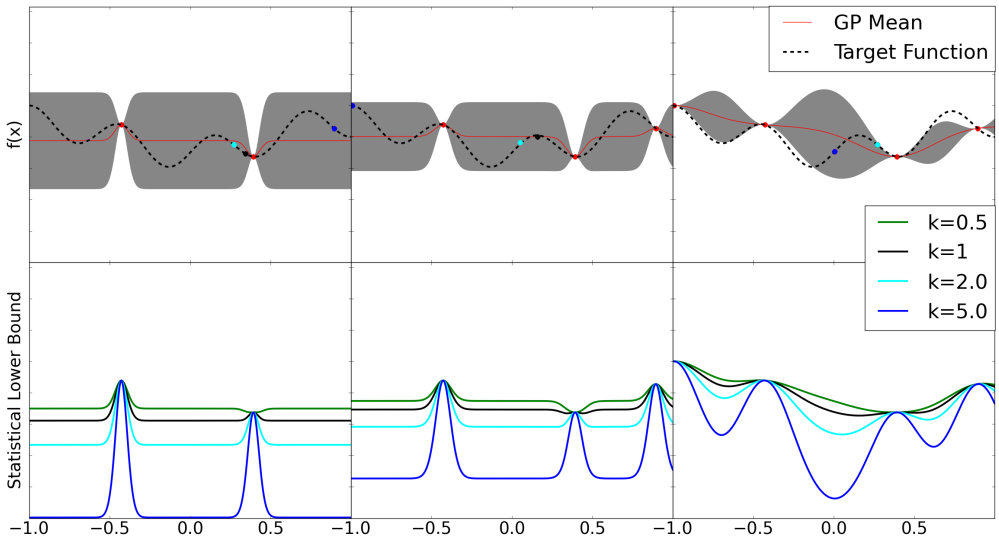


Figure 2.11: From left to right three consecutive snapshots of optimization over a one dimensional sinusoidal function using UCB criterion. The upper row shows the underlying function (dashed black line), available observations (red points), the mean (solid red line) and 2 times standard deviation (gray area) of the GP and the next points to sample from, proposed by UCB's with various ks. The lower row shows the surface of UCB formed using each of four k values.

on the accuracy of the error estimate which is varying during the search process. Figure 2.11 shows how the value of k affects the distance between the next sampling point and the best observation so far. In this example, the proposals produced by four values of k are shown. When $k = 0.5$ the next proposal is exactly the same as the previous best.

2.4.2 Probability of Improvement

It is reasonable to measure the quality of a candidate point by comparing it to the best observation found so far, $\boldsymbol{x}_{min}$. If the function value in a given point is lower than the best observed value so far, $f(\boldsymbol{x}) < y_{min} = f(\boldsymbol{x}_{min})$,

then sampling at this point leads to an improvement with its magnitude being the difference between predicted function value for the point and observed value of the best so far $I(\boldsymbol{x}) = y_{min} - \mu(\boldsymbol{x})$.

A more flexible approach is to replace y_{min} with a user defined target value $T \leq y_{min}$. Subsequently, the probability of improvement (POI) is computed w.r.t this target value:

$$POI(\boldsymbol{x}) = p(I(\boldsymbol{x}) > 0) = \Phi\left(\frac{T - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}\right) \tag{2.40}$$

where $\Phi(.)$ is the Gaussian cumulative distribution function, T is the target value, and $\mu(.)$ and $\sigma(.)$ are the regression model's predictive mean and standard deviation for a given point $\boldsymbol{x}$. A target value can be set w.r.t the best observation so far:

$$T = y_{min} - \alpha|y_{min}| \ , \ \alpha \geq 0 \tag{2.41}$$

in which α controls the minimum amount of improvement, e.g. $\alpha = 0.5$

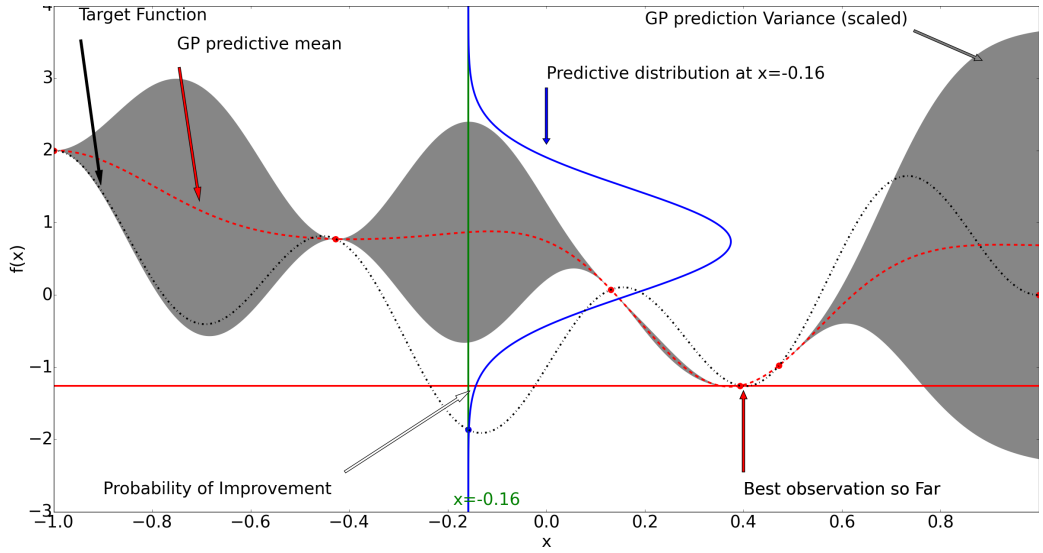


Figure 2.12: Probability of Improvement ($\alpha = 0$).

means search is performed amongst points with at least 50% improvement.

Setting the value of T plays an important role in the search process. Small values produce a bias towards local search where a multitude of infill points are samples from around the best point before starting to move to the global search. On the other hand, if T is set to a large value, the algorithm spends most of its time doing a global search and fails to fine-tune its solutions. One way to cope with this problem is to use several values for T and perform multiple parallel searches per iteration, each corresponding to a different target value. Target values must be set so the spectrum of local and global search is considered. This results in a good combination where the searches that use large target values suggest globally good solutions and the ones with low T fine-tune solutions around these points. Although this solution might work well, similar to the previous infill criterion, its dependence on a hand-tuned parameter is a downside which we try to avoid. Figure 2.12 shows the area whose integral is the POI in the point $x = -0.16$.

2.4.3 Expected improvement

Parameter free criteria are always preferable to the heuristic ones with hand-tuned parameters. This is perhaps the reason behind popularity of the expected improvement (EI) criterion. Given the regression model's prediction mean $\mu(\boldsymbol{x})$ and variance $\sigma(\boldsymbol{x})$ at a given point, $\boldsymbol{x}$, the likelihood of improvement $I(\boldsymbol{x})$, according to its definition in the previous section, has the following probability distribution function [15]:

$$\frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{y_{min}-I(x)-\boldsymbol{\mu}(x)}{2\sigma_n^2}\right) \tag{2.42}$$

Integrating over this density function gives us the EI equation:

$$\begin{aligned} \mathrm{EI}(\boldsymbol{x}) &= \mathbb{E}[\min(0, y_{min} - \mu(\boldsymbol{x}))] \\ &= \int_{-\infty}^{y_{min}} [y_{min} - \mu(\boldsymbol{x})] p(\mu(\boldsymbol{x})) d\mu(\boldsymbol{x}) \\ &= \int_{0}^{+\infty} I(\boldsymbol{x}).\left(-\tfrac{y_{min} - I(\boldsymbol{x}) - \mu(\boldsymbol{x})}{2\sigma_n^2}\right) dI(\boldsymbol{x}) \\ &= \sigma(\boldsymbol{x})[u\Phi(u) + \phi(u)] \end{aligned} \tag{2.43}$$

$$u = \frac{y_{min} - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$$

where $\mu(\boldsymbol{x})$ and $\sigma(\boldsymbol{x})$ are prediction mean and standard deviation at $\boldsymbol{x}$, respectively; $p(.)$ is a normal distribution $\mathcal{N}(\ \mu(\boldsymbol{x}), \sigma(\boldsymbol{x})\ )$; $I(\boldsymbol{x})$ is improvement; $\Phi(.)$ and $\phi(.)$ are normal cumulative distribution and density functions respectively. Using equation 2.43, the next sampling point is selected by maximizing the EI. Figure 2.13 demonstrates three successive steps of GPO over a one-dimensional toy function. The top row depicts changes to the mean and variance of the GP surrogate along with the observed and sampling points proposed by EI, POI and SLB. The bottom row shows the corresponding EI surfaces.

Finding the global minimum is guaranteed by utilizing EI [106]. Moreover, it is a parameter free criterion, as opposed to the criteria presented previously. It also provides a natural termination condition for the optimization – stop when the expected improvement becomes zero. However, this method is very sensitive to the accurate estimation of standard error. Gaussian Processes can greatly underestimate standard deviation if samples are adversely chosen during the initial steps of optimization. Under this condition, obviously the balance between local and global optimization is lost in favour of the latter.

2.4.4 Information-theoretic Infill Criteria

The surface created by the previous Infill criteria could be considered as unnormalized probability density functions of the optimum point. Hence,

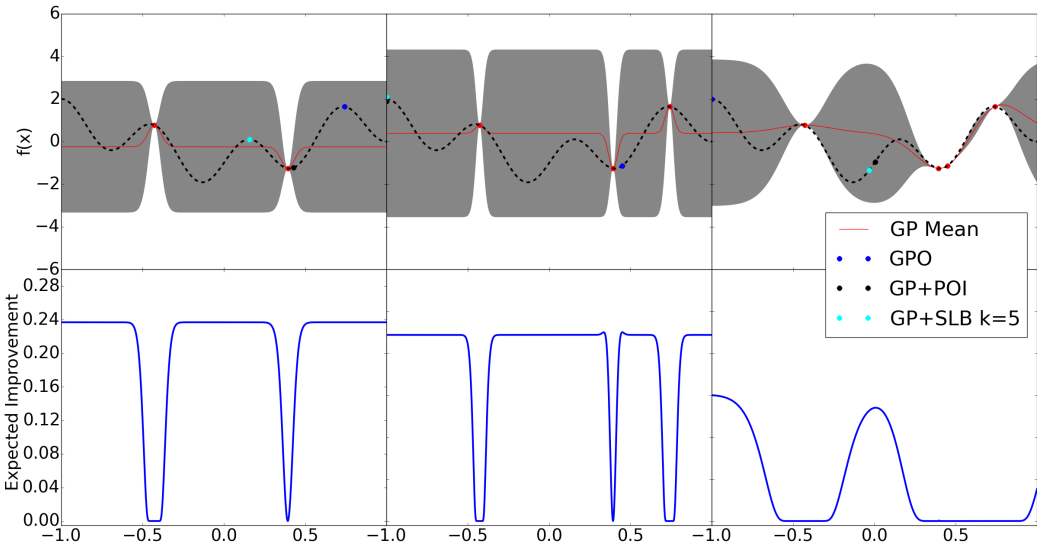


Figure 2.13: Expected Improvement

optimization of these acquisition functions provides a way to sample from the expected position of the global optimum at each optimization step. This is not necessarily the most efficient sampling strategy if the goal is to find the global optimum at the horizon of N function evaluations, rather than the current step. Using information-theoretic infill criteria [80, 83, 202], however, the next query point can be selected such that the information about the position of the global optimum is maximized and consequently the likelihood of reaching it at the horizon is increased.

Entropy Search

In Entropy Search (ES) this is achieved by minimizing the expected differential entropy of the conditional distribution of the global optimum [80]:

$$ES(\boldsymbol{x}) = H[p(\boldsymbol{x}^* \mid \mathcal{D})] - \mathbb{E}_{p(y|\boldsymbol{x},\mathcal{D})}[H(\boldsymbol{x}^* \mid \mathcal{D} \cup \{\boldsymbol{x}, y\})] \tag{2.44}$$

where $p(\boldsymbol{x}^* \mid \mathcal{D}) = p(\boldsymbol{x} = \operatorname{argmin}_{\boldsymbol{x}\in\boldsymbol{\chi}} f(\boldsymbol{x}) \mid \mathcal{D})$ denotes conditional distribution of the global optimum and $H(.)$ is its differential entropy. Since

both terms on the RHS of (2.44) are not analytically available, for this computation to be tractable several trade-offs are needed.

Predictive Entropy Search

Predictive Entropy Search (PES) is another method to compute the same quantity in an easier way by rewriting it in terms of mutual information:

$$PES(\boldsymbol{x}) = H[p(y \mid \boldsymbol{x}, \mathcal{D})] - \mathbb{E}_{p(\boldsymbol{x}^*|\mathcal{D})}[H[p(y \mid \boldsymbol{x}, \boldsymbol{x}^*, \mathcal{D})]] \tag{2.45}$$

where $H[p(y \mid \boldsymbol{x}, \mathcal{D})]$ is the entropy of the predictive distribution at $\boldsymbol{x}$ and $p(y \mid \boldsymbol{x}, \boldsymbol{x}^*, \mathcal{D}) = \int p(y \mid f(\boldsymbol{x}))p(f(\boldsymbol{x} \mid x^*, \mathcal{D}))df(\boldsymbol{x})$ is the prediction distribution at $\boldsymbol{x}$, given the location of the optimum. Using PES, the first term on the RHS can be computed in closed form and the second one can be approximated after applying a series of simplifications [83].

2.4.5 Noisy Infill Criteria

While dealing with noisy functions, most of the common infill criteria might show a sub-optimal performance as no consideration about the noise has been explicitly made in their design. For example, calculation of the EI acquisition function (2.43) relies on y_{min} whose true value is not accessible if the underlying function is noisy.

Augmented Expected Improvement

In the first attempt to address this, [210] adapted the definition of Improvement to noisy environments by replacing y_{min} with the model prediction at the best point found so far

$$\mathrm{I}_n(\boldsymbol{x}) = \mu(\boldsymbol{x}_{min}) - \mu(\boldsymbol{x}) \tag{2.46}$$

where $\mu(.)$ is the predictive mean of the surrogate model. The EI using this definition accounts for the uncertainty in the function value of the best

solution so far. Yet, using this definition the value of the EI at $\boldsymbol{x}$ becomes zero, $EI(\boldsymbol{x}_{min}) = 0$, therefore, it does not allow for repetitive evaluations in $\boldsymbol{x}_{min}$, which should naturally be possible in a noisy environment in order to reduce the uncertainty about the function value.

To address this, *Augmented Expected Improvement* (AEI) is presented in [91] as an extension of EI which is suitable for noisy environments:

$$\text{AEI}(\boldsymbol{x}) = \mathbb{E}[\min(0, \mu(\boldsymbol{x}_{ebs}) - \mu(\boldsymbol{x}))]\left(1 - \frac{\sigma_n}{\sqrt{\sigma^2(\boldsymbol{x}) + \sigma_n^2}}\right) \tag{2.47}$$

Here σ_n is the standard deviation of noise, $\sigma^2(.)$ denotes the predictive variance and $\boldsymbol{x}_{ebs}$ is called *effective best solution* and is determined by using a separate utility function $u(.)$

$$\boldsymbol{x}_{ebs} = \underset{\boldsymbol{x} \in \boldsymbol{X}}{\text{argmax}}(u(\boldsymbol{x})), \qquad u(\boldsymbol{x}) = -\mu(\boldsymbol{x}) - k\sigma(\boldsymbol{x}) \tag{2.48}$$

where $\boldsymbol{X}$ contains the available set of observed locations. Obviously, in a noise-free environment where $\sigma_n^2 = 0$ AEI and EI are equal. However, with a non-zero noise variance AEI promotes exploration by putting more weight on locations in which the predictive variance is higher.

Expected Quantile Improvement

Another variant of EI with the ability to deal with i.i.d noise is *Quantile-based* EI [170, 185]. It is based on a new definition of *improvement* for stochastic functions: *Improvement* is defined as the decrease from the lowest β-quantile of the current observed points. The β-quantile, $\beta \in [0.5, 1]$, at an existing or new location $\boldsymbol{x}$ is computed by using the posterior predictive distribution as follows

$$q(\boldsymbol{x}) = \inf\{u \in \mathbb{R}, p(f(\boldsymbol{x}) < u) \geq \beta\} = \mu(\boldsymbol{x}) + \Phi^{-1}(\beta)\sigma(\boldsymbol{x}) \tag{2.49}$$

where $\Phi(.)$ is Normal cumulative distribution function, $\mu(.)$ the predictive mean and $\sigma(.)$ the predictive standard deviation. The minimum β-quantile

of the available observed points at n^{th} stage of optimization is then computed as $q_{min} = \min_{\boldsymbol{x}\in\boldsymbol{X}}[q(\boldsymbol{x})]$. Subsequently, the quantile improvement function is defined as follows

$$I_Q = \min(0, q_{\min} - Q(\boldsymbol{x})) \tag{2.50}$$

in which $Q(.)$ is the random β-quantile based on the available observations and a new unobserved location $\boldsymbol{x}$. Then, the *expected quantile improvement* (EQI) can be defined as

$$\text{EQI}(\boldsymbol{x}, \sigma_n^2) = \left(q_{\min} - \mu_{Q(\boldsymbol{x})}(\boldsymbol{x})\right) \Phi\left(\frac{q_{\min} - \mu_{Q(\boldsymbol{x})}}{\sigma_{Q(\boldsymbol{x})}}\right) + \sigma_{Q(\boldsymbol{x})}\phi\left(\frac{q_{\min} - \mu_{Q(\boldsymbol{x})}}{\sigma_{Q(\boldsymbol{x})}}\right) \tag{2.51}$$

where σ_n^2 is the variance of the future noise and the EQI depends on it through $Q(.)$. Here $\mu_{Q(\boldsymbol{x})}$ and $\sigma_{Q(\boldsymbol{x})}$ denote the conditional expectation and standard deviation of $Q(\boldsymbol{x})$, respectively.

2.4.6 Goal Seeking

All the above infill criteria can fail. The problem lies in the two-stage nature of the process. In the first stage, a regression model is trained over available data. Although this model might not be a good representative of underlying objective function, its correctness is taken for granted. Accordingly, its predictions are used to form an infill criterion in the second stage.

The Goal seeking method tries to bypath this weakness by integrating these two steps into one. However, to do so at least we need an idea of the (expected) function value or improvement in the current optimization iteration. For now, suppose that this extra information is provided by an oracle in the form of an observed value for the yet unseen data point, say y_{goal}. Now, we can consider the unseen point, $\boldsymbol{X}_{goal}$, as a set of free parameters and maximize the log-likelihood equation 2.30 over $\boldsymbol{\theta} \cup \boldsymbol{X}_{goal}$ to find the position of next infill point i.e. $\boldsymbol{X}_{goal}$. In reality, y_{goal} is not accessible a priori. Instead, analogous to probability of improvement, we may use several different values for y_{goal}.

2.5 Conclusion

In this chapter, we explained the basic background and material required for understanding Bayesian Optimization and the rest of the thesis. A brief explanation of BO and its components was given. Since the BO machinery is largely based on Bayesian regression methods, we detailed the modeling and inference process in *Gaussian Processes*, as they are non-parametric Bayesian regression models with extensive use in BO as well as parametric models. Moreover, we introduced several existing infill criteria used in BO.

Chapter 3

Bayesian Optimization Using Bayesian Neural Networks

3.1 Introduction

Although GP is the most common choice of surrogate in Bayesian Optimization algorithms, it is not the only possibility. In fact, any type of regression model whose output resembles a distribution is a potential surrogate. Here we consider Artificial Neural Networks (NN) as alternative models to GP. These are parametric models in which the cost of learning grows linearly with the number of observations [191].

Our contributions in this chapter are two-fold. (1) We introduce a set of BO algorithms in which simple and fully Bayesian Neural Networks (BNN) are used as surrogate. In BNN the learning process is given a probabilistic interpretation and the product of learning is an ensemble of networks with plausible parameter values [139]. In this study, BNNs are inferred using Monte Carlo sampling. (2) We study the application of surrogates formed from ensembles of Random Feed-Forward Neural Networks, also called Extreme Learning Machines (ELM), in BO. In these models, the weights in the final layer are learned whereas the parameters of the first layers are randomly sampled from a prior distribution. (3) We propose Empirical

Expected Improvement, a simple approximation to the EI infill criterion. Although EI is available in closed form for a small number of distributions, it can only be applied if the predictive distribution is known in advance. The proposed empirical EI has no such restriction and can be used for any model that represents a probability distribution in function space. (4) We perform benchmarking and compare our proposed methods with classic BO using BBOB-2015, a testbed for comparison of real-parameter global optimisers. Although this benchmarking platform is very popular in evolutionary computing community, to our best knowledge this is the first time it is being used for benchmarking and comparison of BO methods.

3.2 Bayesian Neural Networks

In this study, we investigate the use of a prior that is different from the standard GP, i.e., we will use a prior represented by a Bayesian NN in its most commonly used configuration, namely a multi-layer Perceptron (MLP) consisting of a single hidden layer. The network function we use has the following form:

$$f(\boldsymbol{x}) = \psi(\boldsymbol{x} \cdot \boldsymbol{W}_{ih} + \boldsymbol{b}_{ih}) \cdot \boldsymbol{w}_{ho} + b_{ho} \tag{3.1}$$

where $\boldsymbol{x}$ indicates the input vector, $\psi(.)$ is a non-linearity, $\boldsymbol{W}_{ih} \in \mathbb{R}_{m \times h}$, $\boldsymbol{b}_{ih} \in \mathbb{R}_{1 \times h}$, $\boldsymbol{w}_{ho} \in \mathbb{R}_{h \times 1}$ and $b_{ho} \in \mathbb{R}$ are input-to-hidden and hidden-to-output weights and biases, receptively, with m being the dimensionality of the input and h the number of units in the hidden layer. To build a Bayesian NN, we define a prior $p(\boldsymbol{\theta})$ over the parameters of the model $\boldsymbol{\theta} = \{\boldsymbol{W}_{ih}, \boldsymbol{b}_{ih}, \boldsymbol{w}_{ho}, b_{ho}\}$, consider the support of data through the likelihood function $p(\mathcal{D} \mid \boldsymbol{\theta})$ and compute the posterior using Bayes rule $p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$. For a new point $\boldsymbol{x}$, the predictive distribution can be computed in principle by integrating out the latent parameters $p(y \mid \boldsymbol{x}, \mathcal{D}) = \int_{\boldsymbol{\theta}} p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{D}) d\boldsymbol{\theta}$. A neural network architecture together with a prior over weights and a scheme for doing this integral is

termed a Bayesian Neural Network (BNN).

Although a Bayesian NN with an infinite number of hidden units reverts to a Gaussian Process prior [158], with a finite number of hidden units it represents a prior with different characteristics, which may suit optimization problems in which the specifications of underlying function could be better modeled by a NN rather than a GP. For example, as can be seen in Figure 4.1, realizations from a Bayesian NN prior will typically exhibit large plateau regions, which are very different from the wavy-shaped samples drawn from a GP prior with stationary covariance function.

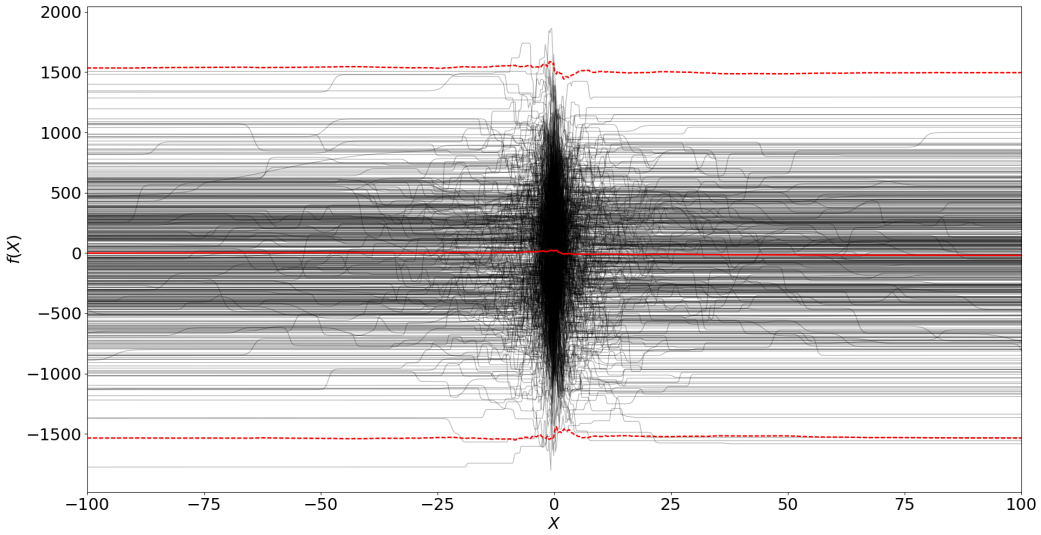


Figure 3.1: 500 NN samples from a simple Bayesian NN prior with 50 hidden units in a single layer. Weights and biases are drawn from zero-mean normal distributions with standard deviation of 100: $\mathcal{N}(0, 100)$. Each dark line is the prediction of one NN sample. Solid and dashed red lines show mean and ± 3 standard deviations of the predictions over all 500 samples.

For a Bayesian neural network (BNN), the MLP function is substituted for $f(.)$ in the regression model of equation 2.2, and we set zero-mean

Gaussian priors over each set of network parameters as in [158]:

$$\begin{aligned} \boldsymbol{w}_{ih}^{j} &\sim \mathcal{N}(0, \sigma_{\boldsymbol{w}_{ih}}^{2}\mathbf{I}_h), \qquad \boldsymbol{b}_{ih} \sim \mathcal{N}(0, \sigma_{\boldsymbol{b}_{ih}}^{2}\mathbf{I}_h) \\ \boldsymbol{w}_{ho} &\sim \mathcal{N}(0, \sigma_{\boldsymbol{w}_{ho}}^{2}\mathbf{I}_h), \qquad b_{ho} \sim \mathcal{N}(0, \sigma_{b_{ho}}^{2}), \end{aligned} \tag{3.2}$$

where $\boldsymbol{w}_{ih}^{j}$ is the jth row of the input-to-hidden weight matrix $\boldsymbol{W}_{ih}$, σ_z^2 is variance of the Gaussian prior over the set of parameters in $\boldsymbol{\theta}$ indicated by $z \in \boldsymbol{\theta}$ and $\mathbf{I}_h$ is a $h \times h$ identity matrix. In simple Bayesian Neural Networks (SBNN) hyperparameters are set to constant values. However, instead of making assumptions about their correct values, we can let them be set with respect to the observations and build a Hierarchical BNN (HBNN). Here, Gamma priors are defined over the standard deviations of the hyperparameters:

$$\begin{gathered} \sigma_{b_{ho}} \sim Gamma(\alpha_{b_{ho}}, \beta_{b_{ho}}) \qquad \sigma_{\boldsymbol{w}_{ho}} \sim Gamma(\alpha_{\boldsymbol{w}_{ho}}, \beta_{\boldsymbol{w}_{ho}}) \\ \sigma_{\boldsymbol{b}_{ih}} \sim Gamma(\alpha_{\boldsymbol{b}_{ih}}, \beta_{\boldsymbol{b}_{ih}}) \qquad \sigma_{\boldsymbol{w}_{ih}} \sim Gamma(\alpha_{\boldsymbol{w}_{ih}}, \beta_{\boldsymbol{w}_{ih}}) \\ \sigma_n \sim Gamma(\alpha_{noise}, \beta_{noise}) \end{gathered} \tag{3.3}$$

in which α_z and β_z are shape and rate parameters, respectively. In BNNs the posterior is analytically intractable. Therefore, approximate inference methods need to be applied. Historically, Hamiltonian Monte Carlo (HMC) [157] has been the most popular sampling method used in BNNs. However, its performance is very sensitive to a proper initialization of its two user-specified parameters, namely step size and number of steps, which are normally set by carrying out a number of initial experiments. This makes HMC inappropriate for the Expensive Optimization application, where new observations are incrementally added and the surrogate model undergoes change in each iteration. Fortunately, the No-U-Turn-Sampler (NUTS) [86], an adaptive version of HMC, resolves this limitation by eliminating the need for setting the number of steps in advance.

3.3 Random Feed Forward Neural Networks

A Random Feed Forward Neural Network, often called an Extreme Learning Machine (ELM) [187, 94, 95, 92, 221, 93, 147], is an MLP with a single hidden layer in which the input-to-hidden weights and biases are sampled from some known distribution, and not modified in response to the data. Only the parameters of the output layer are fitted to data, by finding the smallest norm least-squares solution of this linear system:

$$\hat{\mathbf{w}}_{ho} = \mathbf{H}^{\dagger}\mathbf{y} \tag{3.4}$$

where $\mathbf{H}$ is the matrix of hidden layer outputs and each of its elements h_{ij} is the output of the jth hidden unit for the ith observed point; $\mathbf{H}^{\dagger}$ is Moore-Penrose generalized inverse of matrix $\mathbf{H}$, i.e $\mathbf{H}^{\dagger} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ if $\mathbf{H}^T\mathbf{H}$ is non-singular otherwise $\mathbf{H}^{\dagger} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}$.

In ELMs, to have a predictive distribution it is sufficient to form an ensemble whose members are built by sampling the parameters of their first layer from some known distribution.

3.4 Empirical Expected Improvement

Having an ensemble of regressors sampled from $p(f(\boldsymbol{x}) \mid \mathcal{D})$, an approximation to $EI(\boldsymbol{x})$ is simply computed by averaging over all improvements yielded by ensemble members at point $\boldsymbol{x}$:

$$EI(\boldsymbol{x}) = \frac{1}{N}\sum_{\boldsymbol{\theta}\in\boldsymbol{\Omega}}[y_{min} - f(\boldsymbol{x},\boldsymbol{\theta})]\ ,\ \boldsymbol{\Omega} = \{\boldsymbol{\theta} \mid f_{\boldsymbol{\theta}}(\boldsymbol{x}) < y_{min}\} \tag{3.5}$$

where $\boldsymbol{\Omega}$ is the subset of regressors whose prediction results in improvement. In our case the base regressors are NNs, however, empirical EI could be used with any type of regressors, e.g regression trees, and is invariant under changes to the type of predictive distribution.

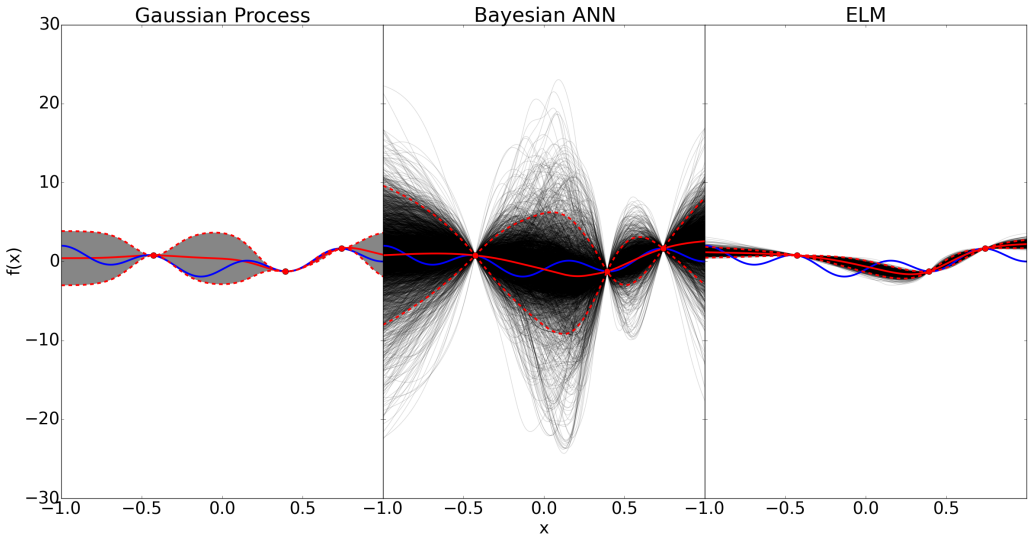


Figure 3.2: Mean (solid red) and prediction variance (dashed red) in GP (left), HBNN (middle) and ensemble of ELMs (right). All three Models are built using 3 observations from a sinusoidal function (solid blue).

3.5 Experiments

All have a single hidden layer where due to performance considerations the number of hidden units is limited to 10.

Six different optimization methods were formed from combinations of SBNN, HBNN and ELM surrogates with Gaussian and Empirical EI criteria. All surrogate models had similar structures of a single hidden layer where due to performance considerations the number of hidden units is limited to 10. SBNN and HBNN used sigmoid hidden units, whereas ELMs used *tanh* activation functions, as in our initial experiments we found ELMs-based optimizers to perform better with *tanh* rather than sigmoid units. To emulate a very low noise, with an expected standard deviation of 0.001, both SBNN and HBNN used Gamma priors over noise with shape and rate parameters being 2.0 and 200, respectively. In SBNN, standard errors of priors were set to 100, i.e., $\sigma_{bho} = \sigma_{who} = \sigma_{bih} = \sigma_{win} = 100$, whereas to have non-informative hyperpriors in HBNN the shape and rate of the

Gamma distribution of the priors over standard deviations of the network parameters were set to 2 and 0.02, respectively. Statistical modelling was performed using pyStan library [194, 21] and NUTS sampler (the adaptive version of HMC) was utilized as the inference mechanism. To prevent the sampler from stalling (exploring the posterior very slowly), the maximum tree-depth of NUTS was set to 8. For both SBNN and HBNN, a total of 3500 samples were generated, with each sample representing the parameters of an NN. The first 2500 samples of the chain were considered as burn-in and discarded, whereas the remaining samples were thinned by 2 and used to form the Bayesian model. The ELM surrogate consists of 500 NNs with input-to-hidden weights drawn from [-10,10] and the same topology as the BNN approach.

Table 3.1: Function Categories.

Functions	Group	Description
f1 – f5	G1	5 separable functions
f6 – f9	G2	4 functions with low or moderate conditioning[1]
f10 – f14	G3	5 functions with high conditioning, unimodal
f15 – f19	G4	5 multi-modal functions with adequate global structure
f20 – f24	G5	5 multi-modal functions with weak global structure

Gaussian Process Optimization (GPO) is considered as the baseline method. The underlying GP uses the `scikit-learn` [169] implementation of Gaussian Process regression, with an anisotropic square exponential covariance function. To find the maximum-likelihood values of the hyperparameters, after each new function evaluation the `fmin_cobyla` algorithm from `scipy.optimize` is applied with 50 random restarts. Opti-

[1] *Conditioning* or *condition number* of a function indicates the amount of change in the output of a function for a small change in the input.

mization over the EI surface is performed using L-BFGS-B [19] with 150 random restarts. L-BFGS-B is a variant of Broyden–Fletche–Goldfarb–Shanno (BFGS), a local optimization algorithm from the family of quasi-Newton methods where the search is guided by an estimate of the inverse Hessian matrix, and is used when the search space is constrained within a bounding box. Unlike other quasi-Newton methods, BFGS and its variants are able to produce acceptable results even if the underlying function is non-smooth.

We evaluate the optimization methods on the BBOB-2015 noiseless testbed, which consists of 24 functions divided into five groups in terms of difficulty (see Table 3.1). These functions are a part of Comparing Continuous Optimizers framework (COCO) [77], which is a popular benchmarking testbed for continuous black-box optimization algorithms and allows for comprehensive evaluation of optimization methods over a set of functions with a wide range of characteristics (please refer to [57] for more details on the benchmark and specifications of each function). Due to the limitation of time and resources, only 15 independent trials were preformed for each method on each function in a set of two, three, five, and ten dimensional problems. It is a common practice to limit the number of function evaluations to a few hundred [122, 128, 189]. Here, we assumed test functions to be expensive enough to limit the budget of each trial to 100 function evaluations. In each trials, the first two samples were drawn randomly and used to initialize the surrogate model. Hence, the results of the experiments also reflect the sensitivity of the algorithms to the position of the initial observations. The optimization process was terminated if the target function value of the underlying function was met, a previously evaluated point was revisited, or the allocated budget was fully used. If the allocated budget was not fully used at termination, the optimization was allowed to restart. To compare the performance of our algorithms, we utilized a hierarchical combination of the Mann-Whitney U test, with family-wise significance level $\alpha_F = 0.001$, and Area Under Curve (AUC) metrics of best-seen objective value. As we compare 7 different methods,

$\alpha_F = 0.001$ could be satisfied by setting $\alpha = 0.0005$ for the significance level of the pairwise tests (see [38] for more details on how to compute pairwise significance levels).

Using the Mann-Whitney U test for each test function and optimization step, a partial ranking amongst our optimization methods is computed. Following [38], the resulted ranks are then aggregated over each dimension and function category based on two schemes: *Borda* and the *number of first place finishes*.

Borda Count [131, 36, 214, 141, 3, 110] (or *Borda* for short) is a well-known "positional" rank aggregation method, in which a cumulative score is computed for each candidate based on its positions in a set of *rank list*s. Given a set of ranks $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_{N_r}$ for a group of candidates $c_1, \ldots, c_{N_c}$, a Borda score S_c^r is assigned to a candidate per each *rank list* which corresponds to the number of items ranked below that item in the list.

$$S_c^r = \sum_{i=1}^{N_c} \mathbb{1}[\boldsymbol{r}_r^i < \boldsymbol{r}_r^c], \qquad c \neq i,\ c \in [1, N_c],\ r \in [1, N_r] \tag{3.6}$$

where $\mathbb{1}[.]$ is the indicator function and $\boldsymbol{r}_r^i$ denotes the rank of i^{th} candidate in the r^{th} *rank list*. Then sum of these individual Borda scores is computed as the *Borda count* of each candidate $S_c = \sum_{r=1}^{N_r} S_c^r$.

3.6 Results and Discussion

The experimental results are summarized in Figure 3.3, Figure 3.4 and Figure 3.5 in the form of aggregated ranks of the optimization algorithms. Figure 3.3 contains histograms of the Borda scores and number of first place finishes of the optimization methods at the end of 100 function evaluations, in four different dimensions and five function groups. Figure 3.4 and Figure 3.5 show how these ranks change against the consumed budget during the optimization process. As can be seen in Figure 3.3, at the end of optimization, the overall performance of the BNN-based methods is better

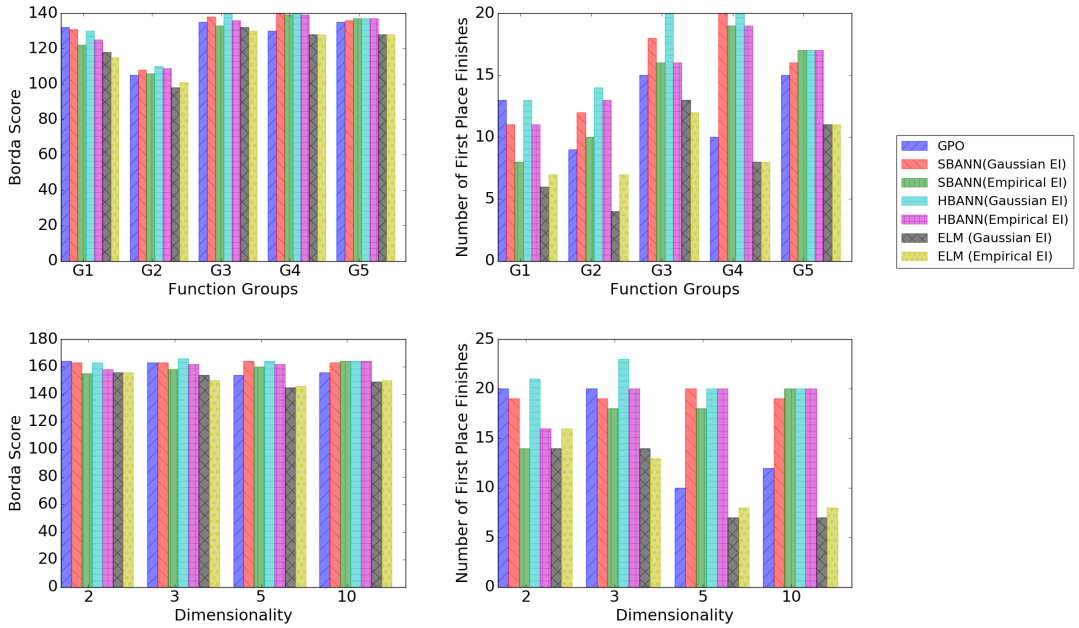


Figure 3.3: Histograms demonstrating the aggregated ranks of the proposed methods and GPO after 100 function evaluations. **Top row**: results of rank aggregation over five different function categories using (*left*) Borda, (*right*) number of first place finishes schemata. **Bottom row**: results of rank aggregation over 2, 3, 5 and 10 dimensional functions using (*left*) Borda and (*right*) number of first place finishes schemata. Each bin shows the score from the corresponding aggregation schema.

than the GPO in the majority of the cases, whereas ELM-based optimizers have completely failed to compete. HBNN with Gaussian EI calculation is, by and large, the best amongst all, whether considered across different dimensions or across different function groups.

Nonetheless, GPO performed exceptionally well in the set of separable functions, G1, as well as two- and three-dimensional problems. In G1, GPO scores highest in terms of the both aggregation schemata, while its number of first place finishes is comparable with HBNN with Gaussian EI. It also has the highest Borda score and the second highest number of first place

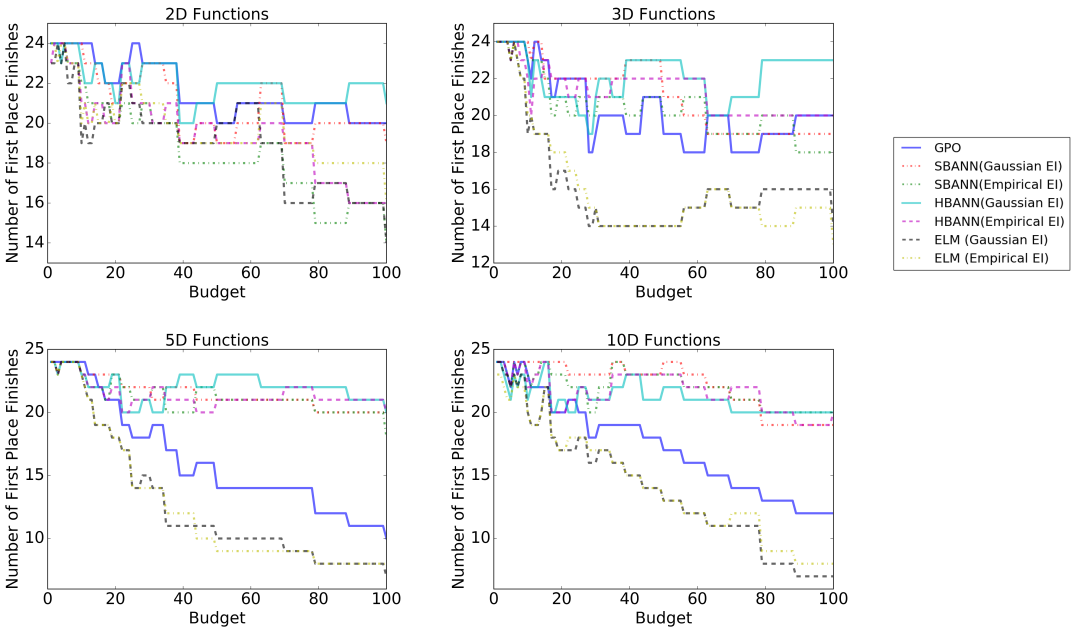


Figure 3.4: Aggregated ranks of different optimization methods over 100 optimization epochs, separated by dimensionality. The figures are germane to the number of first place finishes in partial ranks obtained from hierarchical application of the Mann-Whitney U test over best function values and AUCs. The plots contain results for 2 (*upper left*), 3 (*upper right*), 5 (*lower left*), and 10 (*lower right*) dimensional test functions, respectively.

finishes in 2D functions.

Increasing the dimensionality, however, causes both HBNN and SBNN to start outperforming GPO. In three dimensional functions, for example, GPO ranks second after HBNN (Gaussian EI) with respect to both aggregation schemata, comparable with SBNN (Gaussian EI) in its Borda score and HBNN (Empirical EI) in the number of first place finishes. However, in 5D and 10D functions all BNN-based methods outperform GPO in terms of Borda score as well as number of first place finishes. Almost the same pattern could be seen in the more challenging functions in Table 3.1. While GPO outperforms all other methods in the easiest family of functions (G1),

its superiority fades away in G2, G4 and G5, where all BNN-based optimizers rank higher. Amongst BNN-based optimizers, the ones with Gaussian EI clearly outperform their Empirical counterparts in the majority of the cases and HBNN methods have a slight edge over SBNN ones.

Figure 3.4 and Figure 3.5 demonstrate the changes to the ranks of optimization algorithms throughout the optimization. We noticed that the difference in how Borda and number of first place finishes rank the optimization algorithms is minor, hence only figures related to the latter is presented here. It can be seen in Figure 3.4 that HBNN (Gaussian EI) maintains a high rank during the optimization process almost everywhere and ELM-based methods show disappointing performance by being the worst, even in the early stages of the optimization. The graphs show that with an increase in the amount of budget, BNN-based methods perform better than GPO. Moreover, GPO starts to lose ground to BNN-based methods earlier and more consistently as the dimensionality of the functions increases.

In the 2D functions, the difference between performance of the optimization methods is not significant. GPO ranks slightly higher, sometimes similar to other BNN-based optimizers until around the 50^{th} epoch, from which HBNN (Gaussian EI) starts to perform either comparably to or better than GPO. Increasing the dimensionality causes the difference between performance of the BNN-based methods to become less significant, e.g., although HBNN and SBNN with Empirical EI perform poorly in 2D functions, they start to behave similarly to their Gaussian counterparts for 5D and 10D problems.

Figure 3.5 shows the aggregated ranks of the optimizers during optimization over each function category. Overall, we can discern that BNN-based methods mostly rank higher than GPO in the entire optimization process across all function groups, except G1, where none of the proposed methods could consistently maintain their rank above GPO up to the end of optimization. In all function groups, ELM-based methods were inferior to the others for the entire duration of the optimization except for the initial

few epochs. In groups G2 to G5, most of the time the SBNN and HBNN rank higher than GPO and as we approach the end of the process the difference between the performance of the methods becomes more evident. However, none of the optimizers have complete superiority over the others in all function groups. In G1 to G3, for example, HBNN (Gaussian EI) ranks higher than the others especially in the second half of the optimization and in G4 all BNN-based methods are equally good throughout the process, while GPO ranks as poorly as ELM-based optimizers. In G5, which is the most difficult group, optimizers with Empirical EI rank slightly higher than the rest. Moreover, ELM-based methods with Gaussian and empirical EIs are ranked almost identically during the optimization over all function groups, whereas similar patterns could not be seen in BNN-based optimizers.

A possible explanation for the superiority of the HBNN and poor performance of ELM-based methods, particularly in higher dimensional and more complicated problems, may lie in their predictive variance. As can be seen in Figure 3.2, HBNN has the largest predictive variance. Although this does not seem to be an advantage in optimization of simple low dimensional functions, it provides the HBNN-based methods with a more powerful global search element and enables them to perform well in higher dimensions and over more complicated underlying functions, while reducing their sensitivity to poor choice of initial points. On the other hand, ELM has a small predictive variance in comparison with GP, therefore, it mainly exploits rather than explores. Apparently, this lack of an effective global search element prevents ELM-based optimizers from performing as well as other methods, even for the simpler functions.

A surprising observation arises when comparing empirical with Gaussian EIs. We expected the optimizers with Empirical EI to perform better, or at least as well as, those with Gaussian EI. However, except for the ELM-based methods, the graphs show that the latter have higher final ranks and were more stable during the optimization process. For example, in

both Figure 3.4 (2D functions) and Figure 3.5 (G3), towards the end of the optimization the performance of both BNN-based methods with Empirical EI starts to degrade, whereas the same methods with Gaussian EI continue to maintain their ranks. We expect this to be a result of poor mixing and sub-optimal behavior of the MCMC method rather than an inherent problem of the empirical EI. To prevent the NUTS sampler from stalling we have put a limitation on its tree-depth parameter. This implicitly restricts the sampler's trajectory length and might prevent it from reaching a U turn [86] before each draw. Consequently, the correlation between successive samples increases, and to get a fair estimate of the posterior a larger sample size is required. On the other hand, to build the surrogate model in a timely manner we have used a relatively small sample size and short chain as well as warm-up period. As ELMs do not face this problem, their performance with either of the EI versions is similar.

Although in most cases this study ranks HBNN and SBNN more highly than GPO and ELM, the difference between the aggregated rank values is not significant. This is due to the high number of ties in the Mann-Whitney U tests. Looking at raw function values output by the optimizers over each function/dimension, we can see that in the majority of the cases the performance of the presented optimizers is quite similar and occasionally even disagrees with their aggregated ranks. Figure 3.6–3.8 contain the comparison graphs of the median function values outputted by the optimizers over these function/dimension pairs. As it can be seen, GPO performs well on the 5 and 10 dimensional f1 (Sphere) and f5 (Linear Slope) problems from G1. It is also significantly better than other optimizers over f1, where it converges to a near optimum solution through a few steps. In 10D f12 (Bent Cigar) from G3, again GPO achieves a better final result.

Even in the most difficult category of functions, G5, GPO performs outstandingly well on 10D f21, f22 and f23, i.e., Gallaghers Gaussian 101-me and 21-hi as well as Katsuura functions, respectively. However, in this category, HBNN is mostly the overall best method. Interestingly, ELM-

based optimizers had the best final results for the 2D and 3D functions of f2 (Ellipsoidal), their non separable counterpart f10, f12, as well as on the three-dimensional f5. ELM also performs very well on f6 (Attractive Sector) and f10 in 5D and 10D spaces. As can be seen from the 2D plots of the functions surfaces (first row in figures 3.6-3.8), this suggests that ELM might perform well on smooth functions with anisotropic or ridge-like structures.

3.7 Conclusion

This chapter made the following contributions:

- **BNNs for Bayesian optimization.** We introduced a novel Bayesian optimization method in which simple and hierarchical Bayesian Neural Networks are used as surrogate. It was illustrated that BOs that use a BNN surrogate are able to outperform GPO on a selected set of benchmark function. This is significant due to the fact that the computational complexity of inference in BNNs only scales linearly with the number of observations, whereas in GP it is cubic on the number of data points.

- **Ensemble of ELMs for Bayesian optimization.** For the first time we studied the application of ensemble of random neural networks as surrogate in BO. Although in this work ELM-based BO showed a poorer performance compared with other BO methods, the low computational cost of their inference encourages us to look for more effective methods of using random neural networks as surrogates in BOs.

- **New EI measure.** We proposed empirical expected improvement as a novel way to approximate EI in ensemble-based surrogates and models with non-Gaussian predictive distribution. Comparing the

performance of the proposed method with EI revealed that (Gaussian) EI outperforms the empirical one, probably due to the poor mixing of the the BNN surrogates.

- **Evaluation.** We formed a set of six optimization methods using possible combinations of empirical and Gaussian EI with each of the proposed surrogates. We compared these methods with GPO over a set of 24 noise-free benchmark functions. Our studies showed that even optimization methods with small Bayesian Neural network surrogates performed better than, or at least comparably to, GPO on a number of test functions. The performance difference was clearer in higher dimensions and with more complicated functions, where the effect of under-estimated predictive variance in GP was intensified. Moreover, hierarchical Bayesian NN surrogates that were built using the No U-Turn sampler showed the most promising results in comparison with other presented models, whereas ELM-based surrogates were shown not to be competitive with GPO.

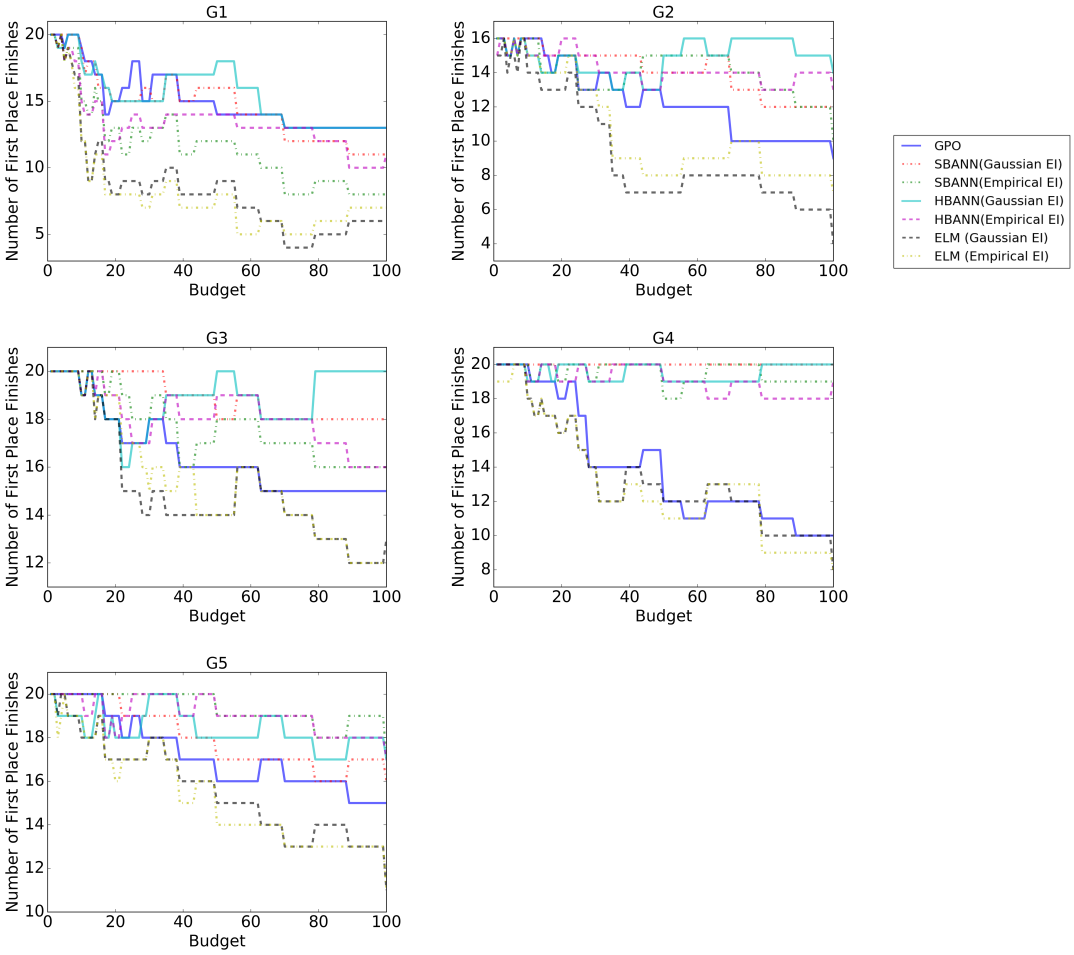


Figure 3.5: Aggregated ranks of different optimization methods over 100 optimization epochs, separated by function category. The figures are germane to the number of first place finishes in partial ranks obtained from hierarchical application of Man-Whitney U test over best function values and AUCs of optimizers in each function group from G1 to G5.

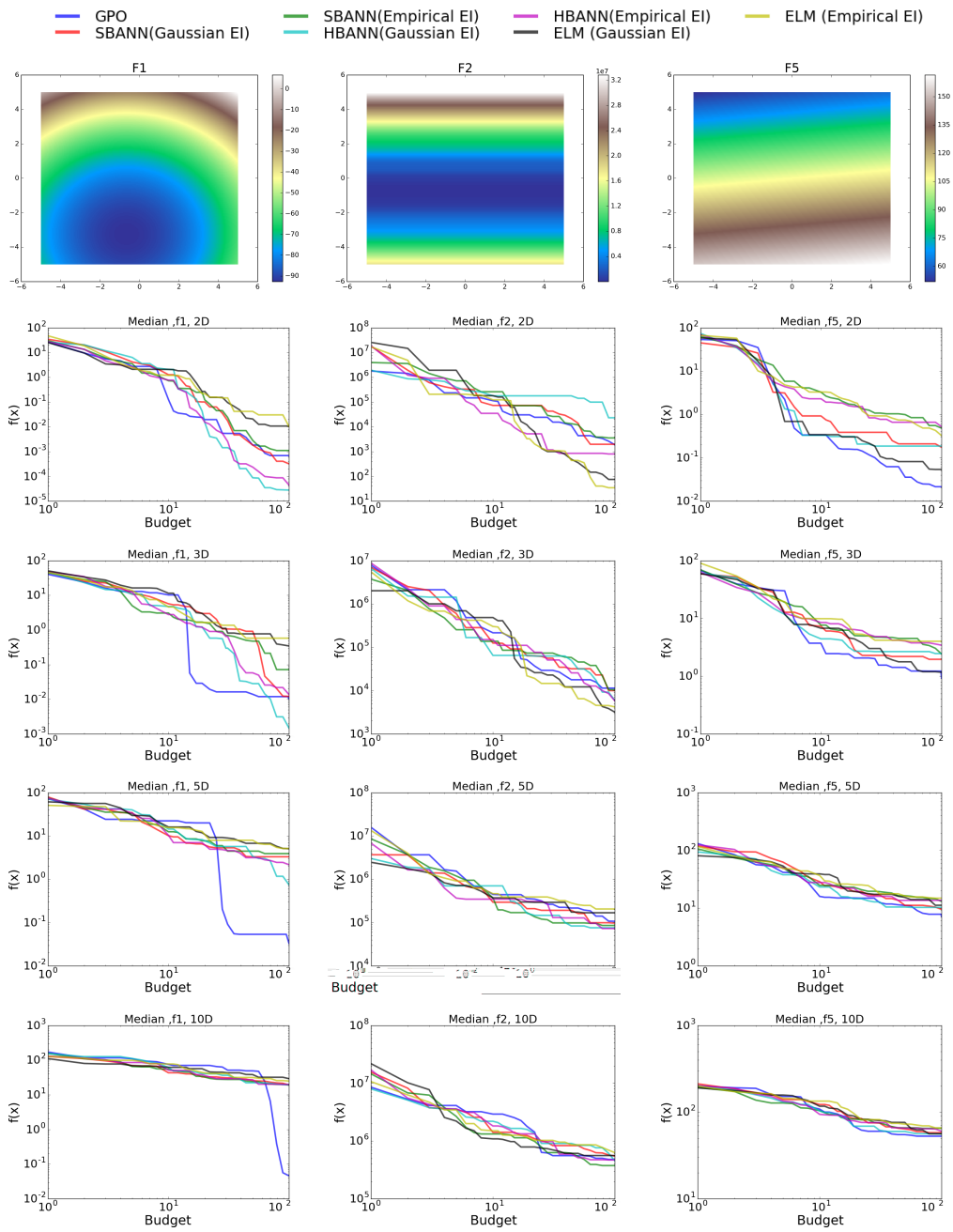


Figure 3.6: Median of function values found during the optimization over the f1, f2 and f5 test functions in 2, 3, 5 and 10 dimensional space. f1 is to compare the best convergence rates, whereas f2 and f5 show the relative ability of optimizers in exploitation of separability and going outside the initial convex hull of solution, respectively. The first (upper) row depicts the functions' structures in 2D space.

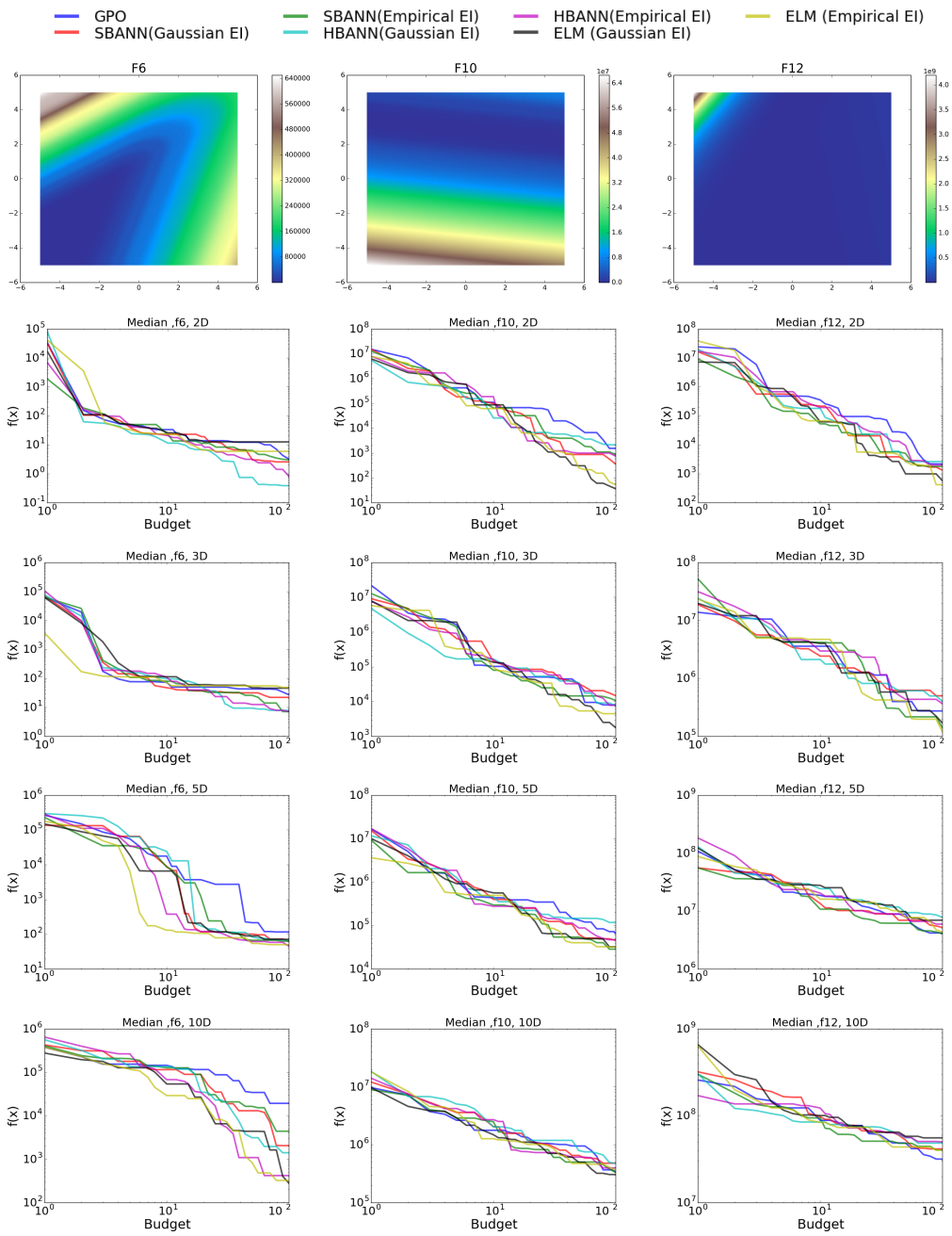


Figure 3.7: Median of function values found during the optimization over the f6, f10 and f12 test functions in 2, 3, 5 and 10 dimensional space. The figures show the effect of increase in the dimensionality on the relative performance of the optimization methods over a highly asymmetric landscape (f6), a non-separable function (f10) and a smooth and narrow ridge (f12). The first (upper) row depicts the functions' structures in 2D space.

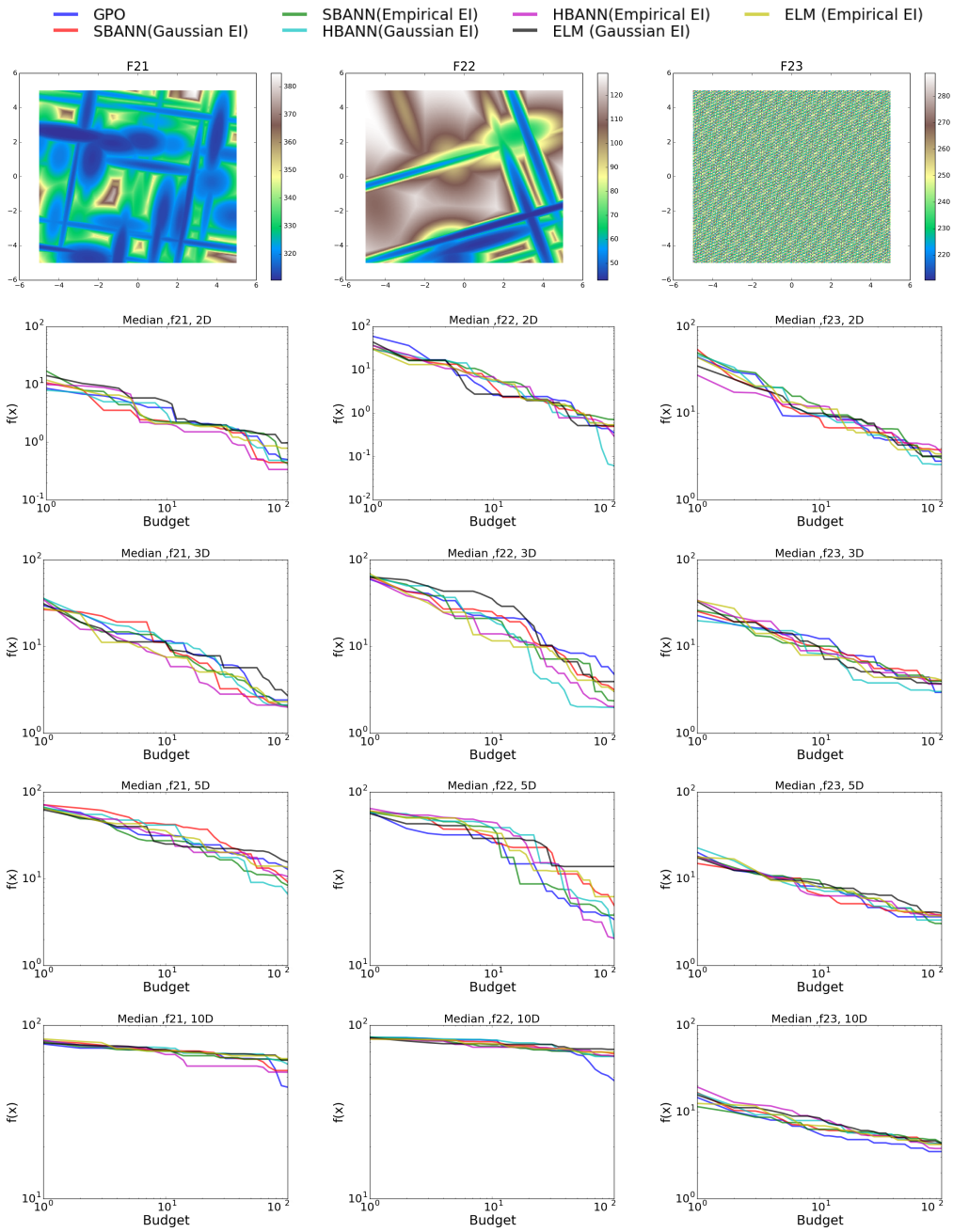


Figure 3.8: Median of function values found during the optimization over the f21, f22 and f23 test functions in 2, 3, 5 and 10 dimensional space. The figures show the effect of increase in the dimensionality on the relative performance of the optimization methods over surfaces that 1) have no global structure (f21), 2) have no global structure but higher condition than f21 (f22), 3) are highly rugged and highly repetitive (f23). The first (upper) row depicts the functions' structures in 2D space.

Chapter 4

Cheap Surrogate Building Using Randomization

4.1 Introduction

This chapter explores a family of methods aimed at getting the benefits of Bayesian NNs while avoiding their high costs.

From the theoretical point of view Bayesian NNs exhibit several advantages over Gaussian Processes, notably:

1. their inference time grows only linearly with the number of observations,
2. by using them one could relax continuity assumptions, and
3. they are inherently multi-output and thus lend themselves for use in multi-objective optimization scenarios (each output is devoted to modelling one of the objective functions).

From a practical perspective, however, Bayesian inference in NNs [158] is not a trivial task. To begin with, the posterior distribution of the model parameters is not analytically available and classic inference methods based on Markov Chain Monte Carlo, i.e Hybrid Monte Carlo [43], are

prohibitively slow. Furthermore, other computationally less expensive approaches such as Bayes by Backprop [13] and MC-Dropout [63] produce overconfident models. Finally, despite the recent advancements in Variational Inference [137, 119, 90, 166], approximating the full posterior distribution over the weights of moderate to large NNs imposes high computational costs.

One solution to the scalability problem of Bayesian NNs is to only perform probabilistic inference in the final layer of the NN and use pointwise estimates to set the rest of the parameters. In this so-called *adaptive basis regression*, first, a deep NN is fully trained on the available observations to get a *maximum a posterior* estimate over its parameters. The training could be performed using one of the many variations of stochastic gradient decent, with (only) the last layer being replaced by a Bayesian linear regressor. Such a model has been successfully used as a surrogate in BO to perform optimization on expensive-to-evaluate functions [190]. Yet, in this model the training is performed on the entire space of the model parameters. Here we take one step further in developing the above idea by using *random basis regression*, i.e linear combination of randomized functions [72]. In this scheme, the training is only performed on a small subset of the model parameters, therefore, the inference process is more efficient than *adaptive basis regression*.

Recent studies show that randomization is an effective alternative to optimization [183, 174, 81, 217, 22]. In fact, it is shown that under certain conditions, a model that consists of weighted sum of random functions is as effective as methods in which the function parameters are fitted [175]. Hence, in many applications the strict necessity of performing the training in the entire parameter space of NNs is under question. In fact, a large class of functions are approximated with high accuracy using models in which most of the parameters are set at random and the training is only performed in the linear final layer. This has given rise to a family of randomized feed-forward networks. As an example of such models, Random

Vector Functional Link (RVFL) [97] is a class of single layer feed-forward NNs (SLFN) in which the parameters of the first (input-to-hidden) layer are constants randomly set from a prespecified distribution, whereas the second (hidden-to-output) layer is a ridge regression model [142].

Ensemble learning is another means to build regression models with the ability to provide uncertainty with their predictions. Recent studies have revealed the existence of strong links between ensembles of NNs and Bayesian inference [24, 121, 138, 162]. It has been shown that an ensemble of MAP solutions independently trained over a noisy loss could be used as an approximation to the posterior [168]. One of the most notable benefits of such an approach is the ability to parallelize the training. In fact, even ensemble models with no Bayesian interpretation can be remarkably successful in BO [100].

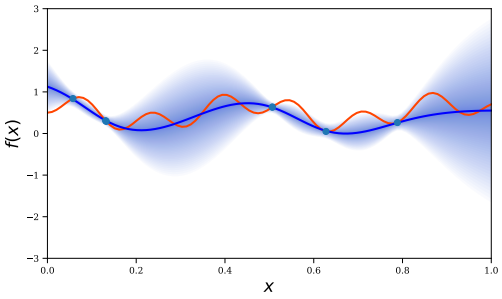
Our contributions in this chapter are as follows:

1. We propose an extremely lightweight surrogate for BO which is based on randomized shallow NNs [187]. Hence, rather than a GP prior the BO uses a prior represented by a *Bayesian* RVFL, i.e. a Random Vector Functional Link mapping in which the second (hidden-to-output) layer is a Bayesian Linear model [186].

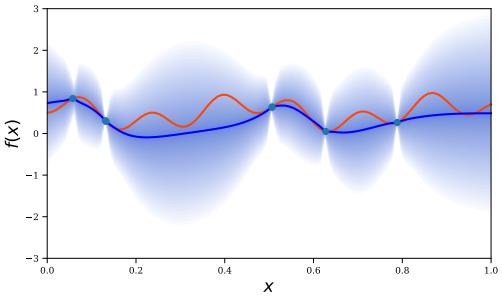
2. We present a simple and effective scheme for initialization of random parameters in an RVFL.

3. We present experiments with a BO where GP is replaced with a prior represented by an ensemble of RVFLs.

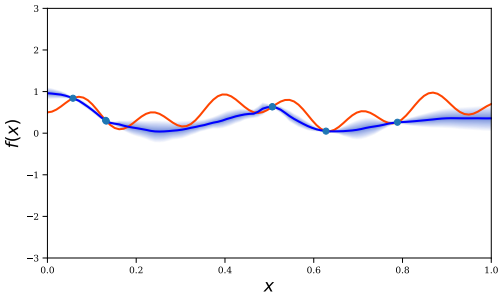
4. We perform extensive evaluation and benchmarking on synthetic functions as well as hyper-parameter optimization of Machine Learning models.



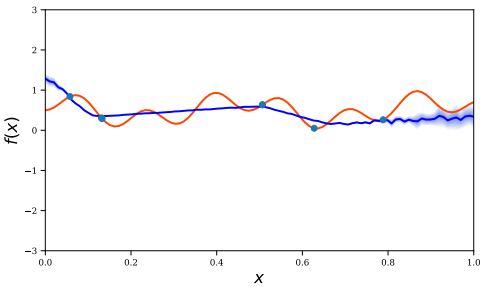
(a) GP



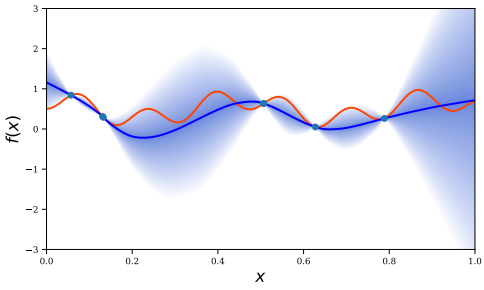
(b) Fully Bayesian GP



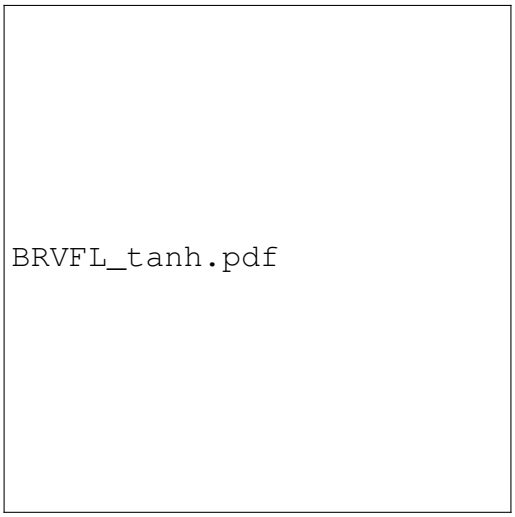
(c) Bayesian NN with HMC



(d) Bayesian NN with Dropout



(e) BRVFL (relu)



(f) BRVFL (tanh)

Figure 4.1: Posterior over function produced by a) GP b) Fully Bayesian GP c) Bayesian NN inferred by Hamiltonian Monte Carlo d) Bayesian NN produced using MC dropout NN e) Bayesian RVFL with *relu* d) Bayesian RVFL with *tanh*

4.2 Random Vector Functional Link Networks

RVFLs are a class of shallow NNs with an extremely fast training algorithm. A RVFL Network with d input nodes, n hidden units and a single scalar output, $g : \mathbb{R}^d \to \mathbb{R}$, is defined as a SLFN of the form:

$$\begin{aligned} g(\boldsymbol{x}) &= \sum_{i=1}^{n} w_i \psi_i(\boldsymbol{x}) \\ \psi_i(\boldsymbol{x}) &= \phi(\boldsymbol{v}_i^\intercal \boldsymbol{x} + b_i) \end{aligned} \tag{4.1}$$

where $\boldsymbol{v}_i \in \mathbb{R}^d$ and $b_i \in \mathbb{R}$ are weights and biases into the i^{th} hidden unit, $\phi(.)$ is a monotonically increasing activation function and $w_i \in \mathbb{R}$ is the output weight of the i^{th} hidden unit. In addition to the architecture, RVFLs also share the same universal approximation properties and error bound with SLFNs [164]. Moreover, it has been shown that adding a direct link from the input to the output of a RVFL greatly boosts its performance [220]. What makes RVFLs different from other types of SLFN is that in RVFLs the input-to-hidden weights and biases are chosen randomly, e.g. from a uniform distribution or a random subset of training points, and set to be constants. Therefore, the training is only performed on the parameters of the final layer. This reduces learning to a convex optimization problem, for which the optimal weight values can be computed in a pre-deterministic number steps using fast quadratic optimization techniques [101, 164]. Given a set of points from a target function with characteristics specified in section 2.2 the maximum likelihood (ML) solution is obtained by taking the gradient

of the log likelihood and setting it to zero [11]:

$$p(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{V}, \boldsymbol{w}, \boldsymbol{b}, \beta) = \prod_{i=1}^{t} \mathcal{N}(y_i \mid \boldsymbol{w}^\intercal \boldsymbol{\psi}(\boldsymbol{x}_i), \boldsymbol{V}, \boldsymbol{b}, \beta)$$

$$\boldsymbol{w}_{ML} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \ \log p(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{w}, \boldsymbol{V}, \boldsymbol{b}, \beta) \quad \propto \quad -\sum_{i=1}^{t} [y_i - \boldsymbol{w}^\intercal \boldsymbol{\psi}(\boldsymbol{x}_i)]^2$$

The gradient of the log likelihood is

$$\nabla \log \ p(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{w}, \boldsymbol{V}, \boldsymbol{b}, \beta) = \sum_{i=1}^{t} [y_i - \boldsymbol{w}^\intercal \boldsymbol{\psi}(\boldsymbol{x}_i)] \boldsymbol{\psi}(\boldsymbol{x}_i)$$

and setting this to zero leads to the ML solution:

$$\boldsymbol{w}_{ML} = \Psi^\dagger \boldsymbol{y}_t \tag{4.2}$$

where $\mathbf{X}_t = [\boldsymbol{x}_1, ..., \boldsymbol{x}_t]$ is the input, $\boldsymbol{V} = [\boldsymbol{v}_1, ..., \boldsymbol{v}_n]$ is the matrix of weights and $\boldsymbol{b}$ is the vector of biases in the first layer, $\boldsymbol{w} = [w_1, ..., w_n]^\intercal$ denotes the vector of the weights in the final layer, $\boldsymbol{\psi}(\boldsymbol{x}) = [\psi_1(\boldsymbol{x}), ..., \psi_n(\boldsymbol{x})]^\intercal$ is the output vector of the first layer and Ψ the design matrix for the final layer:

$$\Psi = \begin{pmatrix} \psi_1(\boldsymbol{x}_1) & \psi_2(\boldsymbol{x}_1) & \cdots & \psi_n(\boldsymbol{x}_1) \\ \psi_1(\boldsymbol{x}_2) & \psi_2(\boldsymbol{x}_2) & \cdots & \psi_n(\boldsymbol{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_1(\boldsymbol{x}_t) & \psi_2(\boldsymbol{x}_t) & \cdots & \psi_n(\boldsymbol{x}_t) \end{pmatrix}$$

$\psi_i(\boldsymbol{x}_j)$ element corresponds to the output of the i^{th} hidden unit for the j^{th} input and $\Psi^\dagger$ is the pseudo-inverse of Ψ which is $(\Psi^\intercal \Psi)^{-1} \Psi$.

4.3 RVFL-based Bayesian Optimization

In this section, we explain our RVFL-based BO. We posit our prior assumption about the target function in the form of a Bayesian RVFL. In Bayesian RVFL the final layer is considered to be a Bayesian regression model, i.e. a

prior distribution (usually a Gaussian) is defined over the parameters of the final layer:

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$$

where $\boldsymbol{\mu}_0 \in \mathbb{R}^n$ is the mean vector and $\boldsymbol{\Sigma}_0 \in \mathbb{R}_+^{n\times n}$ is the symmetric positive semi-definite covariance matrix of the Gaussian prior.

Consequently, the exact posterior could be computed as follows [11] :

$$\begin{aligned} p(\boldsymbol{w} \mid \alpha, \beta, \boldsymbol{V}, \mathcal{D}_t) &= \mathcal{N}(\boldsymbol{\mu},\ \boldsymbol{\Sigma}) \\ \boldsymbol{\mu} &= (\boldsymbol{\Sigma}_0^{-1} + \beta\Psi^\intercal\Psi)^{-1}(\boldsymbol{\Sigma}_0^{-1}\boldsymbol{\mu}_0\ +\ \beta\Psi^\intercal\boldsymbol{y}_t) \\ \boldsymbol{\Sigma}^{-1} &= \boldsymbol{\Sigma}_0^{-1} + \beta\Psi^\intercal\Psi \end{aligned} \tag{4.3}$$

Where $\boldsymbol{\mu} \in \mathbb{R}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}_+^{n\times n}$ are mean and covariance matrix of the posterior probability density function, respectively. As a common practice the prior is usually chosen to be a zero-mean isotropic Gaussian, $p(\boldsymbol{w}) = \mathcal{N}(0, \alpha^{-1}\mathbf{I})$ where α is the precision of the prior, which simplifies the posterior to a Gaussian of the following mean and covariance:

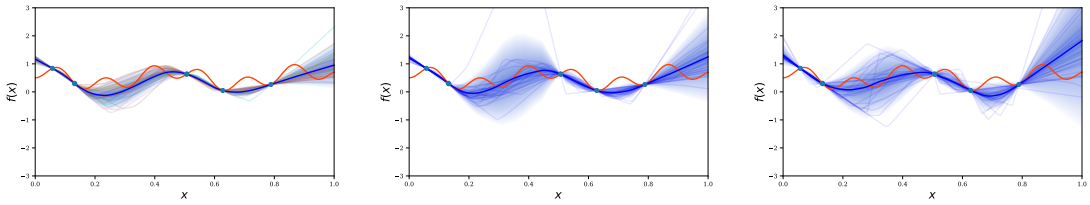
$$\begin{aligned} \boldsymbol{\mu} &= (\frac{\alpha}{\beta}\mathbf{I} + \Psi^\intercal\Psi)^{-1}\Psi^\intercal\boldsymbol{y}_t \\ \boldsymbol{\Sigma}^{-1} &= \alpha\mathbf{I} + \beta\Psi^\intercal\Psi \end{aligned} \tag{4.4}$$

To get the posterior predictive distribution at a new point $\boldsymbol{x}_*$ then we may integrate over $\boldsymbol{w}$:

$$\begin{aligned} p(y_* \mid \boldsymbol{x}_*, \boldsymbol{V}, \mathcal{D}_t) &= \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{w}) p(\boldsymbol{w} \mid \alpha, \beta, \boldsymbol{V}, \mathcal{D}_t) d\boldsymbol{w} \\ &= \mathcal{N}(\ \mu(\boldsymbol{x}_*),\ \sigma^2(\boldsymbol{x}_*)\ ) \\ \mu(\boldsymbol{x}_*) &= \boldsymbol{\mu}^\intercal\Psi(\boldsymbol{x}*) \\ \sigma^2(\boldsymbol{x}_*) &= \frac{1}{\beta} + \Psi(\boldsymbol{x}_*)^\intercal\boldsymbol{\Sigma}\Psi(\boldsymbol{x}_*) \end{aligned} \tag{4.5}$$

It is easy to see that, due to the existence of matrix inversions in equation (4.4) and (4.5), the above solution is computationally constrained by the number of observations as well as hidden units. Moreover, as it does not entail any uncertainty about the values of the random parameters in the

hidden units, we generally expect it to produce overconfident predictions. To understand the importance of the latter, it is rather simple to show that a set of independently trained RVFLs with different random values for the parameters of hidden units is able to quantify uncertainty similar to a Bayesian RVFL (see Figure 4.2) provided that the random generating distribution for these parameters is selected meticulously. If an appropriate random generating distribution is identified in advance, this method is extremely efficient and enjoys a high level of parallelization, however, if it is not set in a principled way, there is no guarantee that the produced uncertainty measure is calibrated or even fruitful in BO task.



(a) (b) (c)

Figure 4.2: Predictive distribution produced by 3 ensembles of RVFLs with similar random generating distribution for input-to-hidden weights $\boldsymbol{w} \sim uniform[-1, 1]$ and biases uniformly generated from a) [-1, 2], b) [-2, 3], c) [-3, 4].

To take advantage of this parallelization and yet retain the benefits of the Bayesian approach, we draw on a rather informal interpretation of the posterior sampling [138] i.e. a set of samples from the model space which comply with both observations and prior beliefs about uncertain quantities of the model. This interpretation is the basis for a set of Bayesian approximate inference methods in which posterior sampling is performed by training models over different noisy versions of a mutual loss [6, 138, 208]. Here we have employed *randomized anchored MAP sampling* [167], a general algorithm based on the above scheme, applicable for NNs and models in which the posterior is dominated by the prior. This algorithm

allows for an efficient, simple and fully parallelizable inference of a *Bayesian ensemble of RVFLs*.

To form a *Bayesian ensemble of RVFLs* using randomized anchored MAP sampling, as in the Bayesian RVFL we define a prior over the weights of the final layer, $p(\boldsymbol{w})$ where each RVFL is trained by:

1. sampling the parameters of the hidden nodes from their corresponding random distributions $\boldsymbol{v}_i \sim p(\boldsymbol{v}), b_i \sim p(b)$;

2. optimizing the per-model log-posterior to get the weight values of the final layer:

$$\boldsymbol{w}_i = \underset{\boldsymbol{w}}{\operatorname{argmax}} \left[\log p(\boldsymbol{y}_t \mid \mathbf{X}_t, \boldsymbol{w}, \boldsymbol{V}_i, \boldsymbol{b}_i, \beta) + \log p_i(\boldsymbol{w})\right] \tag{4.6}$$

   where $p_i(\boldsymbol{w})$ is the per-model prior.

It is proven that [167] in linear models, e.g. RVFLs, with normal likelihood and prior such as in equation (4.3), the correct per-model prior is a normal $p_i(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{\mu}_{0i}, \boldsymbol{\Sigma}_0)$ whose mean is sampled from the following distribution:

$$\boldsymbol{\mu}_{0i} \sim \mathcal{N}(0, \lambda\mathbf{I} + \lambda^2\beta\Psi^\intercal\Psi), \quad \lambda = \frac{\alpha}{\beta} \tag{4.7}$$

Where $\boldsymbol{\Sigma}_0$ is the prior covariance, α is the precision of the prior and β is the precision of the noise. A faster, but coarser, approximation to the posterior could be given by sampling the mean of the per-model prior from $p(\boldsymbol{w})$. However, in this work we rely on equation (4.7) to compute $p_i(\boldsymbol{w})$s.

An important, but largely ignored, question in working with RVFLs is how to randomly generate the parameters of the hidden units in a plausible way. Since symmetry of the random generating distribution is the only criterion to guarantee the universal approximation properties of RVFLs [97], prior work has tended to stick to this recipe. Most work on random shallow networks simply selects the weights and biases of the hidden units uniformly from $[-1, 1]$ and $[0, 1]$ intervals, respectively, i.e. independent of the character of the data distribution. However, this could hardly be

considered an optimal strategy as it might be prone to highly correlated hidden units, e.g. when the activation regions fall far from the input points. This has resulted in a set of works [45, 44, 46, 205] with the goal of devising good initialization strategies for the parameters of NNs with random units.

To circumvent this problem, here we utilize weight normalization [184]. Weight normalisation is a reparametrization method that has been primarily used to accelerate optimization in NNs. In the present case, however, it allows us to decouple the norm of the weight vectors from their direction. In RVFLs the L2 norm of the hidden-unit weights is essentially redundant, since the hidden unit output values are subsequently re-scaled by the hidden-to-output weights during the training. The input-to-hidden mapping merely projects the input onto a line whose direction is uniformly distributed (due to the Gaussian being spherically symmetric). By L2-normalising the input-to-hidden weights, they only vary in terms of directions in input space. The bias is then precisely the distance from the origin in input space. This allows us to select the biases randomly from the input distribution, i.e a zero-mean unit-variance normal when the input has been standardized.

4.4 Experiments

To see how BO methods based on RVFL stack up against other BO methods, we compared them with the existing hyperparameter optimization (HPO) methods from the RoBO library [117]. These were Random Search and four other BOs with different surrogate models, namely GP, Fully Bayesian GP (GP_{mcmc}), Random Forest and Hamiltonian Monte Carlo Neural Net (Bohamiann). The experiments were performed on the set of synthetic functions as well as surrogate benchmarks of HPOlib2 hyper-parameter optimization library [49, 51]. HPOlib2 is a benchmarking library for HPO algorithms and allows us to test the proposed BOs on both synthetic functions and hyper-parameter optimization for machine learning models. Eight vari-

ations of RVFL-based BO were created using Bayesian and ensemble RVFLs. The differences are due to the type of activation function used, and whether or not a skip connection was included between input and output, as shown in Table 4.1.

Table 4.1: Different variations of RVFL-based BO

Index	Type	Activation	Skip Conn.
BRVFL1	Bayesian RVFL	*tanh*	✗
BRVFL2		*tanh*	✓
BRVFL3		*relu*	✗
BRVFL4		*relu*	✓
ERVFL1	Ensemble of RVFLs	*tanh*	✗
ERVFL2		*tanh*	✓
ERVFL3		*relu*	✗
ERVFL4		*relu*	✓

In all experiments we have used RVFLs with 300 hidden units and left the parameters of all HPO methods other than RVFL-based BOs to their default values. The number of hidden units were chosen so that the algorithms could consume their budget (maximum number of optimization steps) before hitting the time limit.

4.4.1 Synthetic Functions

RVFL-based BO with Fixed Hyperparameters

In the first set of experiments on the synthetic benchmark functions precisions of the noise distribution as well as the prior over hidden-to-output weights are chosen to be fixed and their values are selected according to a set of initial observations on a one dimensional toy problem. Since the functions are noise-free, the noise precision is set to a large value,

$\alpha = 1000$. Furthermore, as recommended in [158] the standard deviation of the weights prior is scaled w.r.t the the number of hidden units, $\alpha^{-\frac{1}{2}} = \sigma_{\boldsymbol{w}} = \hat{\sigma_{\boldsymbol{w}}}|\boldsymbol{w}|^{-\frac{1}{2}}$. Hence, we set $\hat{\sigma_{\boldsymbol{w}}} = 0.02$ and $\hat{\sigma_{\boldsymbol{w}}} = 0.001$ for the *relu* and *tanh* networks, respectively. RVFL inputs are standardised and as noise and prior distributions are fixed, the model output is also standardised to prevent unreasonably under(over)-confident models that are disproportionate to the function range.

For each HPO method the experiments consist of 30 independent trials per benchmark function. In each trial, the optimization method is initialised with two random observations. Furthermore, 90 hours of CPU-time is allocated to each trial within which they could perform a maximum of 200 function evaluations (The HPOlib2's default budget for its synthetic benchmark). During the trial, the *absolute regret*, i.e. the absolute difference between the best function value found so far (by the optimization method) and the function value of the optimum point is recorded in each step and used as a means to compare the performance of different HPO methods. The performance of these methods is ranked over each function using the Friedman test[1] [62]. Subsequently, Borda counts are computed from these ranks to get the overall performance of each method over the whole benchmark.

Figure 4.3 contains the Borda scores of the under-experiment HPO methods after 200 function evaluations aggregated over the whole set of benchmark functions. As can be seen, BO with GP_{mcmc} over its hyperparameters has been ranked highest over the synthetic benchmark functions where Bohamiann and GP are the first and second runner-ups, respectively. RVFL-based methods generally showed a poor performance, worse than random forest. The weakest methods amongst all were ERVFL1 and

[1] Mann-Whitney U test, used in the previous chapter, is more suitable for pairwise comparisons, therefore applying it to compare multiple methods requires special care and some adjustments. However, Friedman test is designed for circumstances were multiple method are being compared over several attempts, hence, fits our application better.

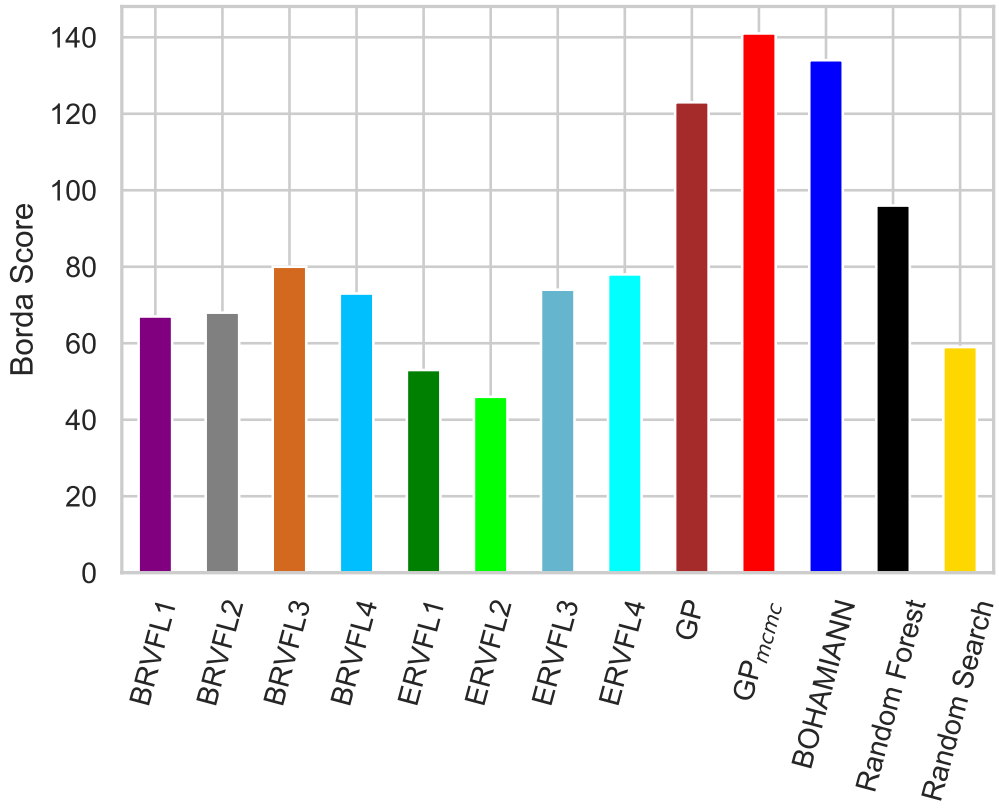


Figure 4.3: Performance evaluation of RVFL-based BOs with fixed hyper-parameters. Borda scores are obtained from applying Friedman test to the final regret values after 200 function evaluations.

ERVFL2 with their overall rank being lower than random search. However, this is not the whole perspective.

Table 4.2: Evaluation of RVFL-based BO (fixed α) over Bohachevsky, Branin and Camelback functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out.

Method	Bohachevsky mean $\pm$ std	Branin mean $\pm$ std	Camelback mean $\pm$ std
BRVFL1	$\underline{7.85 \pm 1.35 \times 10^{1}}$	$2.86 \times 10^{-3} \pm 2.19 \times 10^{-3}$	$7.44 \times 10^{-3} \pm 7.20 \times 10^{-3}$
BRVFL2	$\underline{6.71 \pm 1.28 \times 10^{1}}$	$\underline{2.52 \times 10^{-3} \pm 1.73 \times 10^{-3}}$	$\underline{6.71 \times 10^{-3} \pm 6.29 \times 10^{-3}}$
BRVFL3	$3.89 \times 10^{3} \pm 4.07 \times 10^{3}$	$4.16 \times 10^{-3} \pm 4.02 \times 10^{-3}$	$1.08 \times 10^{-2} \pm 8.10 \times 10^{-3}$
BRVFL4	$3.64 \times 10^{3} \pm 3.70 \times 10^{3}$	$3.70 \times 10^{-3} \pm 2.74 \times 10^{-3}$	$1.60 \times 10^{-2} \pm 1.27 \times 10^{-2}$
ERVFL1	$1.53 \times 10^{2} \pm 1.39 \times 10^{2}$	$9.09 \times 10^{-2} \pm 7.78 \times 10^{-2}$	$3.86 \times 10^{-2} \pm 2.78 \times 10^{-2}$
ERVFL2	$1.84 \times 10^{2} \pm 1.67 \times 10^{2}$	$6.14 \times 10^{-2} \pm 4.72 \times 10^{-2}$	$5.12 \times 10^{-2} \pm 3.65 \times 10^{-2}$
ERVFL3	$4.94 \times 10^{3} \pm 3.22 \times 10^{3}$	$3.90 \times 10^{-2} \pm 3.89 \times 10^{-2}$	$2.76 \times 10^{-2} \pm 2.08 \times 10^{-2}$
ERVFL4	$3.73 \times 10^{3} \pm 3.03 \times 10^{3}$	$3.73 \times 10^{-2} \pm 2.75 \times 10^{-2}$	$2.63 \times 10^{-2} \pm 2.16 \times 10^{-2}$
Bohamiann	$\mathbf{3.31 \times 10^{-1} \pm 2.16 \times 10^{-1}}$	$\underline{1.40 \times 10^{-3} \pm 2.73 \times 10^{-3}}$	$\underline{1.99 \times 10^{-3} \pm 2.36 \times 10^{-3}}$
GP	$1.60 \times 10^{1} \pm 8.72 \times 10^{1}$	$5.16 \times 10^{-2} \pm 2.82 \times 10^{-1}$	$4.42 \times 10^{-2} \pm 2.29 \times 10^{-1}$
GP_{mcmc}	$1.75 \times 10^{2} \pm 9.58 \times 10^{2}$	$\mathbf{2.61 \times 10^{-5} \pm 3.46 \times 10^{-5}}$	$\mathbf{2.78 \times 10^{-5} \pm 3.30 \times 10^{-5}}$
RandomForest	$1.54 \times 10^{1} \pm 1.42 \times 10^{1}$	$4.52 \times 10^{-2} \pm 8.22 \times 10^{-2}$	$3.16 \times 10^{-2} \pm 1.00 \times 10^{-1}$
RandomSearch	$9.68 \times 10^{1} \pm 8.06 \times 10^{1}$	$2.28 \times 10^{-1} \pm 2.16 \times 10^{-1}$	$9.18 \times 10^{-2} \pm 7.62 \times 10^{-2}$

Looking at the mean and variance of the regret values corresponding to the best points found by each optimization method over each function during (Figure 4.4) and at the end of optimization (Table.4.2 to Table.4.5) reveals that use of RVFL-based methods in BO is not bluntly out of question. In fact, considering the mean regret, in 7 out of 12 functions at least one RVFL-based method has appeared among the top-three. Best performance of RVFL-based methods was given in 10 dimensional Levy in which GP_{mcmc}-based BO could not even finish the trials within the time budget, however, BRVLF3, followed by BRVLF4, ranked highest from around 75^{th} step of the optimization onwards and ERVFL4 took the third top position at the end of optimization. On the other hand, in Goldstein-Price the top three positions are taken by GP_{mcmc}, Bohamiann and GP, respectively. Nevertheless, RVFL-

Table 4.3: Evaluation of RVFL-based BO (fixed α) over Goldsteinprice, Hartmann3 and Hartmann6 functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out.

Method	Goldsteinprice mean $\pm$ std	Hartmann3 mean $\pm$ std	Hartmann6 mean $\pm$ std
BRVFL1	$2.79 \times 10^{3} \pm 2.84 \times 10^{3}$	$2.32 \times 10^{-3} \pm 2.04 \times 10^{-3}$	$1.16 \pm 6.11 \times 10^{-1}$
BRVFL2	$3.05 \times 10^{3} \pm 3.02 \times 10^{3}$	$2.14 \times 10^{-3} \pm 1.58 \times 10^{-3}$	$1.05 \pm 3.97 \times 10^{-1}$
BRVFL3	$3.18 \times 10^{3} \pm 3.26 \times 10^{3}$	$\underline{1.76 \times 10^{-3} \pm 9.45 \times 10^{-4}}$	$7.51 \times 10^{-1} \pm 2.94 \times 10^{-1}$
BRVFL4	$3.11 \times 10^{3} \pm 3.27 \times 10^{3}$	$1.91 \times 10^{-3} \pm 1.43 \times 10^{-3}$	$7.83 \times 10^{-1} \pm 3.34 \times 10^{-1}$
ERVFL1	$3.36 \times 10^{2} \pm 4.50 \times 10^{2}$	$2.87 \times 10^{-3} \pm 1.70 \times 10^{-3}$	$1.18 \pm 4.90 \times 10^{-1}$
ERVFL2	$2.81 \times 10^{2} \pm 3.79 \times 10^{2}$	$2.93 \times 10^{-3} \pm 2.48 \times 10^{-3}$	$1.27 \pm 4.34 \times 10^{-1}$
ERVFL3	$1.88 \times 10^{3} \pm 1.77 \times 10^{3}$	$1.95 \times 10^{-3} \pm 1.61 \times 10^{-3}$	$6.92 \times 10^{-1} \pm 2.68 \times 10^{-1}$
ERVFL4	$1.66 \times 10^{3} \pm 1.66 \times 10^{3}$	$1.82 \times 10^{-3} \pm 1.22 \times 10^{-3}$	$7.41 \times 10^{-1} \pm 2.83 \times 10^{-1}$
Bohamiann	$\underline{4.91 \pm 8.20}$	$\underline{5.63 \times 10^{-4} \pm 1.40 \times 10^{-3}}$	$\underline{5.22 \times 10^{-2} \pm 5.56 \times 10^{-2}}$
GP	$\underline{6.34 \pm 1.27 \times 10^{1}}$	$6.20 \times 10^{-2} \pm 8.18 \times 10^{-2}$	$\mathbf{5.17 \times 10^{-2} \pm 6.83 \times 10^{-2}}$
GP_{mcmc}	$\mathbf{2.85 \pm 9.35}$	$\mathbf{5.41 \times 10^{-7} \pm 1.41 \times 10^{-6}}$	$\underline{1.22 \times 10^{-1} \pm 1.50 \times 10^{-1}}$
RandomForest	$3.07 \times 10^{1} \pm 3.39 \times 10^{1}$	$1.97 \times 10^{-1} \pm 7.25 \times 10^{-1}$	$1.48 \times 10^{-1} \pm 8.84 \times 10^{-2}$
RandomSearch	$1.26 \times 10^{1} \pm 1.12 \times 10^{1}$	$1.49 \times 10^{-1} \pm 1.08 \times 10^{-1}$	$1.01 \pm 3.34 \times 10^{-1}$

Table 4.4: Evaluation of RVFL-based BO (fixed α) over Levy (2d and 10d) as well as Hartmann6 functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out.

Method	Levy10d mean ± std	Levy2d mean ± std	Levy5d mean ± std
BRVFL1	$1.21 \times 10^{2} \pm 2.62 \times 10^{1}$	$1.11 \times 10^{-2} \pm 2.34 \times 10^{-2}$	$1.72 \times 10^{1} \pm 7.01$
BRVFL2	$1.28 \times 10^{2} \pm 2.55 \times 10^{1}$	$1.15 \times 10^{-2} \pm 1.60 \times 10^{-2}$	$1.61 \times 10^{1} \pm 6.31$
BRVFL3	$\mathbf{1.04 \pm 3.17 \times 10^{-1}}$	$1.09 \times 10^{-2} \pm 1.12 \times 10^{-2}$	$1.11 \pm 8.30 \times 10^{-1}$
BRVFL4	$\underline{1.21 \pm 3.60 \times 10^{-1}}$	$1.26 \times 10^{-2} \pm 1.36 \times 10^{-2}$	$\underline{9.26 \times 10^{-1} \pm 7.74 \times 10^{-1}}$
ERVFL1	$9.73 \times 10^{1} \pm 2.05 \times 10^{1}$	$7.30 \times 10^{-2} \pm 6.57 \times 10^{-2}$	$1.61 \times 10^{1} \pm 5.36$
ERVFL2	$1.04 \times 10^{2} \pm 1.96 \times 10^{1}$	$9.12 \times 10^{-2} \pm 6.43 \times 10^{-2}$	$1.70 \times 10^{1} \pm 4.18$
ERVFL3	$3.09 \times 10^{1} \pm 1.98 \times 10^{1}$	$3.81 \times 10^{-2} \pm 3.53 \times 10^{-2}$	3.44 ± 2.90
ERVFL4	$\underline{2.54 \times 10^{1} \pm 2.42 \times 10^{1}}$	$4.44 \times 10^{-2} \pm 4.17 \times 10^{-2}$	2.43 ± 2.21
Bohamiann	$2.97 \times 10^{1} \pm 9.37$	$\underline{6.13 \times 10^{-4} \pm 7.26 \times 10^{-4}}$	$\underline{8.44 \times 10^{-1} \pm 4.63 \times 10^{-1}}$
GP	$3.31 \times 10^{1} \pm 4.46 \times 10^{1}$	$\underline{7.13 \times 10^{-3} \pm 3.77 \times 10^{-2}}$	$8.23 \pm 2.36 \times 10^{1}$
GP_{mcmc}	N/A	$\mathbf{3.45 \times 10^{-6} \pm 5.01 \times 10^{-6}}$	$\mathbf{3.44 \times 10^{-1} \pm 3.60 \times 10^{-1}}$
RandomForest	$2.69 \times 10^{1} \pm 1.05 \times 10^{1}$	$1.01 \times 10^{-1} \pm 2.13 \times 10^{-1}$	4.53 ± 3.43
RandomSearch	$4.34 \times 10^{1} \pm 9.55$	$2.07 \times 10^{-1} \pm 1.31 \times 10^{-1}$	7.90 ± 3.82

Table 4.5: Evaluation of RVFL-based BO (fixed α) over Rosenbrock (2d and 5d) as well as 2 dimensional sine functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out.

Method	Rosenbrock2d mean ± std	Rosenbrock5d mean ± std	Sintwo mean ± std
BRVFL1	$1.27 \times 10^3 \pm 1.29 \times 10^3$	$6.79 \times 10^4 \pm 3.19 \times 10^4$	$5.67 \times 10^{-3} \pm 3.98 \times 10^{-3}$
BRVFL2	$1.54 \times 10^3 \pm 1.71 \times 10^3$	$6.55 \times 10^4 \pm 2.94 \times 10^4$	$6.63 \times 10^{-3} \pm 3.37 \times 10^{-3}$
BRVFL3	$1.15 \times 10^3 \pm 1.36 \times 10^3$	$7.75 \times 10^4 \pm 3.58 \times 10^4$	$5.43 \times 10^{-3} \pm 3.52 \times 10^{-3}$
BRVFL4	$1.69 \times 10^3 \pm 2.16 \times 10^3$	$7.14 \times 10^4 \pm 3.28 \times 10^4$	$6.45 \times 10^{-3} \pm 4.08 \times 10^{-3}$
ERVFL1	$1.17 \times 10^2 \pm 1.62 \times 10^2$	$4.51 \times 10^4 \pm 2.05 \times 10^4$	$9.72 \times 10^{-3} \pm 4.65 \times 10^{-3}$
ERVFL2	$1.76 \times 10^2 \pm 3.08 \times 10^2$	$4.06 \times 10^4 \pm 2.01 \times 10^4$	$1.07 \times 10^{-2} \pm 5.88 \times 10^{-3}$
ERVFL3	$9.50 \times 10^2 \pm 1.19 \times 10^3$	$5.54 \times 10^4 \pm 2.55 \times 10^4$	$4.87 \times 10^{-3} \pm 2.98 \times 10^{-3}$
ERVFL4	$7.56 \times 10^2 \pm 1.05 \times 10^3$	$5.66 \times 10^4 \pm 2.45 \times 10^4$	$\underline{4.34 \times 10^{-3} \pm 2.88 \times 10^{-3}}$
Bohamiann	$\underline{8.23 \times 10^{-1} \pm 7.76 \times 10^{-1}}$	$\mathbf{3.01 \times 10^2 \pm 2.47 \times 10^2}$	$\underline{3.13 \times 10^{-3} \pm 4.13 \times 10^{-3}}$
GP	$\mathbf{1.00 \times 10^{-1} \pm 2.38 \times 10^{-1}}$	$5.67 \times 10^3 \pm 1.41 \times 10^4$	$1.36 \times 10^{-2} \pm 4.48 \times 10^{-2}$
GP_{mcmc}	$\underline{3.78 \times 10^{-1} \pm 1.00}$	$\underline{1.65 \times 10^3 \pm 5.75 \times 10^3}$	$\mathbf{4.86 \times 10^{-5} \pm 7.63 \times 10^{-5}}$
RandomForest	4.25 ± 5.14	$\underline{1.76 \times 10^3 \pm 1.36 \times 10^3}$	$4.76 \times 10^{-3} \pm 4.85 \times 10^{-3}$
RandomSearch	3.05 ± 2.89	$3.05 \times 10^3 \pm 3.12 \times 10^3$	$7.47 \times 10^{-3} \pm 4.72 \times 10^{-3}$

based BOs show their worst performance over these functions where none of them could do better than random search.

Due to the varying performance of RVFL-based BOs over different functions and the fact that at least on a number of synthetic functions they actually did as well as or better than the others, we hypothesised that rather than being inherently flawed their poor performance is due to the lack of proper values for their hyper-parameters. When the selected hyper-parameter values are far from their correct values, the predictive uncertainty is either heavily overestimated or underestimated. This creates an imbalance between the local and global elements of the search and shifts the optimization towards either pure exploitation or exploration. One of the quantities that we have set fixed in all previous experiments is the prior distribution over $\boldsymbol{w}$ where its parameters are selected based on a toy problem whose characteristics might be far from the ones we have used to evaluate our methods. A more relevant approach would be to let the hyper-parameter values vary based on the available observations.

RVFL-based BO with Evidence Approximation

In this section we present a new set of experiments over the synthetic functions using RVFL-based BOs whose hyper-parameters are *learned from the observations*. As in the previous experiments, we have set the parameters of the noise distribution to constant values. But, the precision of the prior over $\boldsymbol{w}$, is fine-tuned using *evidence approximation* i.e. by maximising the log marginal likelihood [11]. This is achieved by initialising α with a reasonable value and then iteratively going through equations (4.8) and (4.9):

$$\gamma = \sum \frac{\lambda_i}{\alpha + \lambda_i} \tag{4.8}$$

$$\alpha = \frac{\gamma}{\boldsymbol{\mu}^\intercal \boldsymbol{\mu}} \tag{4.9}$$

where λ_i is the ith eigenvalue of the Hessian matrix of the log marginal likelihood. In the case of ERVFLs, we have performed this procedure for

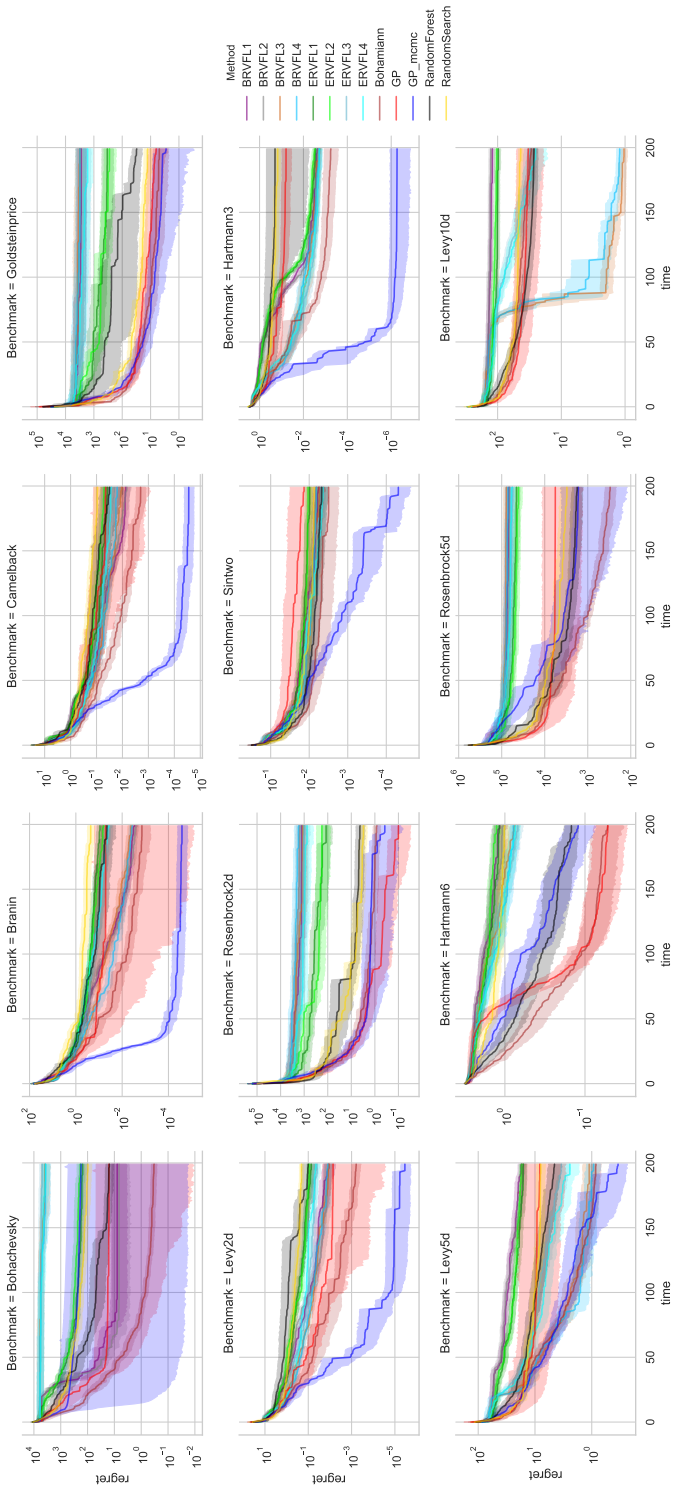


Figure 4.4: Comparing the performance of RVFL-based BOs (fixed α) with other RoBO HPO methods during the optimization over synthetic benchmark functions. Note that y-axis values are log-scaled.

the first member of the ensemble and used the α to set the priors for all other members.

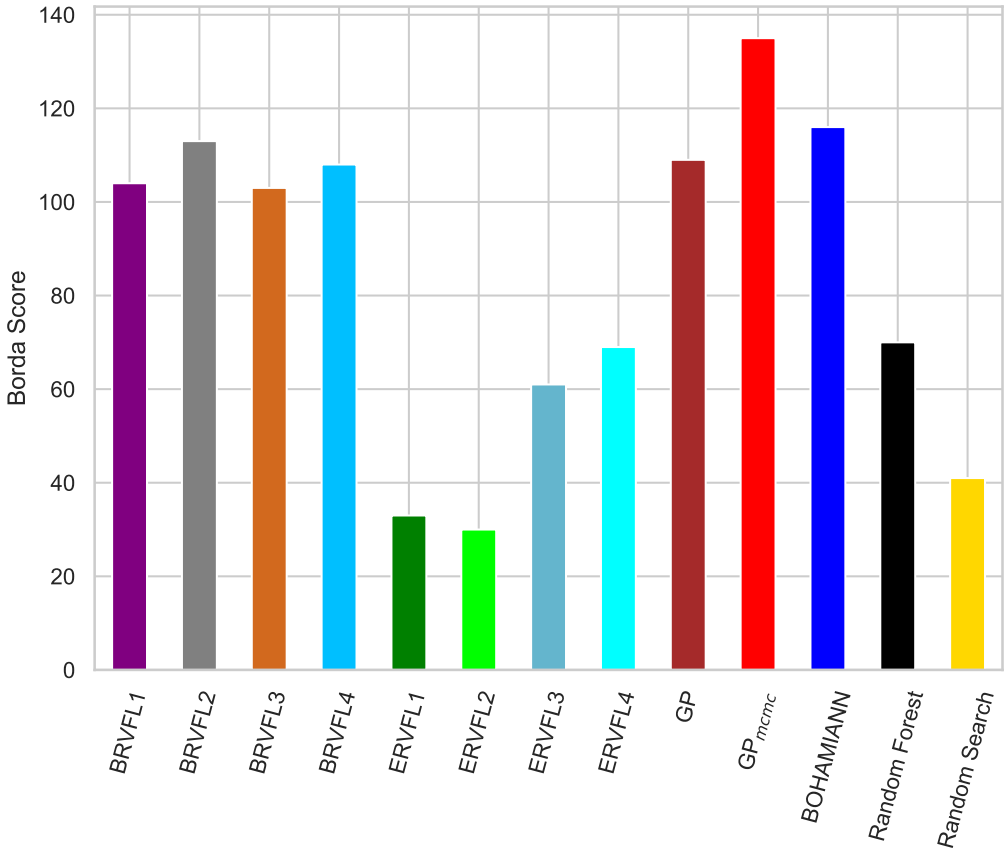


Figure 4.5: Comparing the performance of RVFL-based BOs (learned α) with other RoBO HPO methods during the optimization over synthetic benchmark functions. Note that y-axis is log-scaled.

Figures 4.5 and 4.6 and Tables 4.6 to 4.9 contain the experimental results from comparing GP, GP$_{mcmc}$, Random Forest, Bohamiann and random search with RVFL-based BOs whose prior precision, α, is optimised using evidence approximation. As shown in Figure 4.5 none of the methods were able to beat GP$_{mcmc}$. BRVFL-based methods have enjoyed an elevation in their rank due to the optimization of prior precision. This enhancement

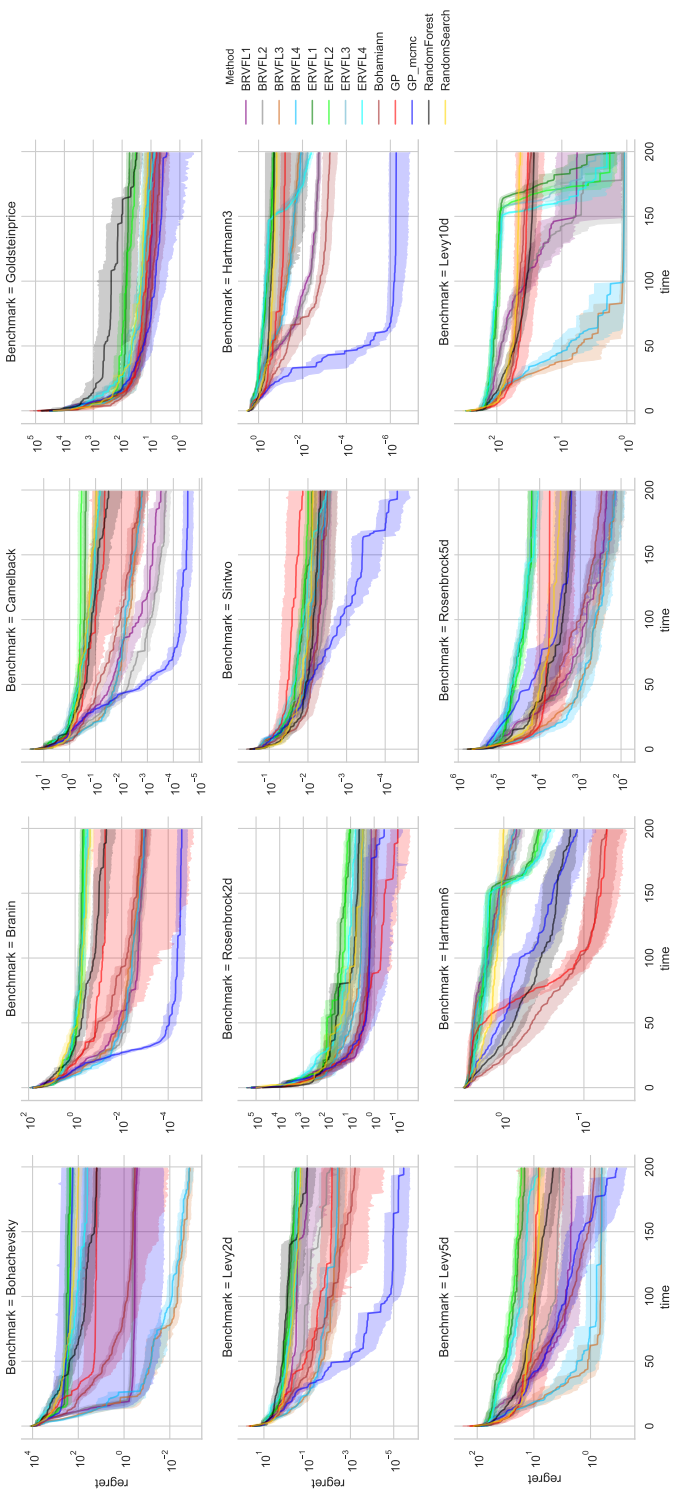


Figure 4.6: Comparing the performance of RVFL-based BOs with other RoBO HPO methods during the optimization over synthetic benchmark functions. Note that y-axis values are log-scaled. Prior precision, α, of BRVFLs derived using evidence approximation.

Table 4.6: Evaluation of RVFL-based BO over Bohachevsky, Branin and Camelback functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function, the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out. Prior precision, α, of BRVFLs derived using evidence approximation.

Method	Bohachevsky mean $\pm$ std	Branin mean $\pm$ std	Camelback mean $\pm$ std
BRVFL1	$\underline{2.71 \times 10^{-1} \pm 1.89 \times 10^{-1}}$	$\underline{9.54 \times 10^{-4} \pm 6.57 \times 10^{-4}}$	$\underline{3.01 \times 10^{-4} \pm 3.78 \times 10^{-4}}$
BRVFL2	$3.88 \times 10^{-1} \pm 2.24 \times 10^{-1}$	$\underline{8.66 \times 10^{-4} \pm 9.85 \times 10^{-4}}$	$\underline{1.92 \times 10^{-4} \pm 2.28 \times 10^{-4}}$
BRVFL3	$\mathbf{1.33 \times 10^{-3} \pm 1.29 \times 10^{-3}}$	$1.29 \times 10^{-3} \pm 9.35 \times 10^{-4}$	$1.60 \times 10^{-3} \pm 1.52 \times 10^{-3}$
BRVFL4	$\underline{1.41 \times 10^{-3} \pm 1.44 \times 10^{-3}}$	$1.11 \times 10^{-3} \pm 6.61 \times 10^{-4}$	$1.61 \times 10^{-3} \pm 1.56 \times 10^{-3}$
ERVFL1	$2.27 \times 10^{2} \pm 2.06 \times 10^{2}$	$4.54 \times 10^{-1} \pm 3.50 \times 10^{-1}$	$2.33 \times 10^{-1} \pm 1.46 \times 10^{-1}$
ERVFL2	$2.30 \times 10^{2} \pm 2.73 \times 10^{2}$	$4.19 \times 10^{-1} \pm 3.35 \times 10^{-1}$	$3.44 \times 10^{-1} \pm 2.68 \times 10^{-1}$
ERVFL3	$4.91 \times 10^{1} \pm 4.39 \times 10^{1}$	$2.86 \times 10^{-1} \pm 2.22 \times 10^{-1}$	$7.27 \times 10^{-2} \pm 5.55 \times 10^{-2}$
ERVFL4	$4.00 \times 10^{1} \pm 4.36 \times 10^{1}$	$3.03 \times 10^{-1} \pm 2.49 \times 10^{-1}$	$6.68 \times 10^{-2} \pm 7.83 \times 10^{-2}$
Bohamiann	$3.31 \times 10^{-1} \pm 2.16 \times 10^{-1}$	$1.40 \times 10^{-3} \pm 2.73 \times 10^{-3}$	$1.99 \times 10^{-3} \pm 2.36 \times 10^{-3}$
GP	$1.60 \times 10^{1} \pm 8.72 \times 10^{1}$	$5.16 \times 10^{-2} \pm 2.82 \times 10^{-1}$	$4.42 \times 10^{-2} \pm 2.29 \times 10^{-1}$
GP_{mcmc}	$1.75 \times 10^{2} \pm 9.58 \times 10^{2}$	$\mathbf{2.61 \times 10^{-5} \pm 3.46 \times 10^{-5}}$	$\mathbf{2.78 \times 10^{-5} \pm 3.30 \times 10^{-5}}$
RandomForest	$1.54 \times 10^{1} \pm 1.42 \times 10^{1}$	$4.52 \times 10^{-2} \pm 8.22 \times 10^{-2}$	$3.16 \times 10^{-2} \pm 1.00 \times 10^{-1}$
RandomSearch	$9.68 \times 10^{1} \pm 8.06 \times 10^{1}$	$2.28 \times 10^{-1} \pm 2.16 \times 10^{-1}$	$9.18 \times 10^{-2} \pm 7.62 \times 10^{-2}$

Table 4.7: Evaluation of RVFL-based BO over Goldsteinprice, Hartmann3 and Hartmann6 functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function, the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out. Prior precision, α, of BRVFLs derived using evidence approximation.

Method	Goldsteinprice mean ± std	Hartmann3 mean ± std	Hartmann6 mean ± std
BRVFL1	$8.64\pm 1.12\times 10^{1}$	$\underline{1.77\times 10^{-3}\pm 1.00\times 10^{-3}}$	$6.97\times 10^{-1}\pm 3.22\times 10^{-1}$
BRVFL2	$1.10\times 10^{1}\pm 1.06\times 10^{1}$	$1.81\times 10^{-3}\pm 9.05\times 10^{-4}$	$6.32\times 10^{-1}\pm 2.32\times 10^{-1}$
BRVFL3	$9.44\pm 1.73\times 10^{1}$	$1.39\times 10^{-2}\pm 9.47\times 10^{-3}$	$7.50\times 10^{-1}\pm 3.30\times 10^{-1}$
BRVFL4	$1.10\times 10^{1}\pm 2.16\times 10^{1}$	$1.17\times 10^{-2}\pm 8.19\times 10^{-3}$	$7.34\times 10^{-1}\pm 3.50\times 10^{-1}$
ERVFL1	$5.53\times 10^{1}\pm 3.84\times 10^{1}$	$1.96\times 10^{-1}\pm 1.33\times 10^{-1}$	$3.66\times 10^{-1}\pm 1.19\times 10^{-1}$
ERVFL2	$3.43\times 10^{1}\pm 2.80\times 10^{1}$	$2.11\times 10^{-1}\pm 1.09\times 10^{-1}$	$3.46\times 10^{-1}\pm 1.35\times 10^{-1}$
ERVFL3	7.34 ± 5.33	$5.80\times 10^{-3}\pm 3.56\times 10^{-3}$	$2.87\times 10^{-1}\pm 8.19\times 10^{-2}$
ERVFL4	7.72 ± 7.28	$4.06\times 10^{-3}\pm 2.68\times 10^{-3}$	$2.61\times 10^{-1}\pm 8.63\times 10^{-2}$
Bohamiann	$\underline{4.91\pm 8.20}$	$\underline{5.63\times 10^{-4}\pm 1.40\times 10^{-3}}$	$\underline{5.22\times 10^{-2}\pm 5.56\times 10^{-2}}$
GP	$\underline{6.34\pm 1.27\times 10^{1}}$	$6.20\times 10^{-2}\pm 8.18\times 10^{-2}$	$\mathbf{5.17\times 10^{-2}\pm 6.83\times 10^{-2}}$
GP_{mcmc}	$\mathbf{2.85\pm 9.35}$	$\mathbf{5.41\times 10^{-7}\pm 1.41\times 10^{-6}}$	$\underline{1.22\times 10^{-1}\pm 1.50\times 10^{-1}}$
RandomForest	$3.07\times 10^{1}\pm 3.39\times 10^{1}$	$1.97\times 10^{-1}\pm 7.25\times 10^{-1}$	$1.48\times 10^{-1}\pm 8.84\times 10^{-2}$
RandomSearch	$1.26\times 10^{1}\pm 1.12\times 10^{1}$	$1.49\times 10^{-1}\pm 1.08\times 10^{-1}$	$1.01\pm 3.34\times 10^{-1}$

Table 4.8: Evaluation of RVFL-based BO over Levy (2d and 10d) as well as Hartmann6 functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out. Prior precision, α, of BRVFLs derived using evidence approximation.

Method	Levy10d mean ± std	Levy2d mean ± std	Levy5d mean ± std
BRVFL1	$5.82\pm 1.55 \times 10^{1}$	$9.77 \times 10^{-2}\pm 2.90 \times 10^{-1}$	2.17 ± 2.51
BRVFL2	$\underline{1.19\pm 2.50 \times 10^{-1}}$	$8.36 \times 10^{-3}\pm 1.90 \times 10^{-2}$	3.53 ± 3.61
BRVFL3	$\underline{1.11\pm 1.96 \times 10^{-1}}$	$\underline{3.34 \times 10^{-3}\pm 6.83 \times 10^{-3}}$	$\underline{6.30 \times 10^{-1}\pm 2.14 \times 10^{-1}}$
BRVFL4	$\mathbf{1.09\pm 2.22 \times 10^{-1}}$	$3.87 \times 10^{-3}\pm 6.92 \times 10^{-3}$	$\underline{6.32 \times 10^{-1}\pm 4.03 \times 10^{-1}}$
ERVFL1	$1.54\pm 9.48 \times 10^{-1}$	$3.29 \times 10^{-1}\pm 3.21 \times 10^{-1}$	$1.46 \times 10^{1}\pm 5.91$
ERVFL2	1.83 ± 1.17	$2.65 \times 10^{-1}\pm 2.03 \times 10^{-1}$	$1.66 \times 10^{1}\pm 6.38$
ERVFL3	2.38 ± 4.86	$3.08 \times 10^{-1}\pm 1.59 \times 10^{-1}$	9.06 ± 5.43
ERVFL4	2.03 ± 1.30	$2.29 \times 10^{-1}\pm 1.80 \times 10^{-1}$	8.21 ± 4.92
Bohamiann	$2.97 \times 10^{1}\pm 9.37$	$\underline{6.13 \times 10^{-4}\pm 7.26 \times 10^{-4}}$	$8.44 \times 10^{-1}\pm 4.63 \times 10^{-1}$
GP	$3.31 \times 10^{1}\pm 4.46 \times 10^{1}$	$7.13 \times 10^{-3}\pm 3.77 \times 10^{-2}$	$8.23\pm 2.36 \times 10^{1}$
GP_{mcmc}	N/A	$\mathbf{3.45 \times 10^{-6}\pm 5.01 \times 10^{-6}}$	$\mathbf{3.44 \times 10^{-1}\pm 3.60 \times 10^{-1}}$
RandomForest	$2.69 \times 10^{1}\pm 1.05 \times 10^{1}$	$1.01 \times 10^{-1}\pm 2.13 \times 10^{-1}$	4.53 ± 3.43
RandomSearch	$4.34 \times 10^{1}\pm 9.55$	$2.07 \times 10^{-1}\pm 1.31 \times 10^{-1}$	7.90 ± 3.82

Table 4.9: Evaluation of RVFL-based BO over Rosenbrock (2d and 5d) as well as 2 dimensional sine functions. Figures represent means and standard deviations of the final regrets after 200 function evaluations aggregated over 30 independent optimization runs. For each function, the best result (bold font) as well as the first and second runner-ups (underlined) are pointed out. Prior precision, α, of BRVFLs derived using evidence approximation.

Method	Rosenbrock2d mean $\pm$ std	Rosenbrock5d mean $\pm$ std	Sintwo mean $\pm$ std
BRVFL1	1.33 ± 1.96	$2.27 \times 10^{2} \pm 3.06 \times 10^{2}$	$3.70 \times 10^{-3} \pm 3.96 \times 10^{-3}$
BRVFL2	1.42 ± 2.00	$\underline{1.42 \times 10^{2} \pm 2.24 \times 10^{2}}$	$\underline{2.69 \times 10^{-3} \pm 2.56 \times 10^{-3}}$
BRVFL3	2.95 ± 4.12	$\underline{1.46 \times 10^{2} \pm 1.80 \times 10^{2}}$	$3.79 \times 10^{-3} \pm 2.49 \times 10^{-3}$
BRVFL4	3.02 ± 4.37	$\mathbf{1.41 \times 10^{2} \pm 1.76 \times 10^{2}}$	$3.29 \times 10^{-3} \pm 2.10 \times 10^{-3}$
ERVFL1	$1.03 \times 10^{1} \pm 1.07 \times 10^{1}$	$1.54 \times 10^{4} \pm 1.02 \times 10^{4}$	$7.69 \times 10^{-3} \pm 4.44 \times 10^{-3}$
ERVFL2	$1.11 \times 10^{1} \pm 1.09 \times 10^{1}$	$1.56 \times 10^{4} \pm 9.85 \times 10^{3}$	$9.92 \times 10^{-3} \pm 4.53 \times 10^{-3}$
ERVFL3	4.38 ± 3.38	$1.45 \times 10^{4} \pm 9.46 \times 10^{3}$	$3.15 \times 10^{-3} \pm 1.78 \times 10^{-3}$
ERVFL4	6.23 ± 7.26	$1.10 \times 10^{4} \pm 7.29 \times 10^{3}$	$3.21 \times 10^{-3} \pm 1.97 \times 10^{-3}$
Bohamiann	$\underline{8.23 \times 10^{-1} \pm 7.76 \times 10^{-1}}$	$3.01 \times 10^{2} \pm 2.47 \times 10^{2}$	$\underline{3.13 \times 10^{-3} \pm 4.13 \times 10^{-3}}$
GP	$\mathbf{1.00 \times 10^{-1} \pm 2.38 \times 10^{-1}}$	$5.67 \times 10^{3} \pm 1.41 \times 10^{4}$	$1.36 \times 10^{-2} \pm 4.48 \times 10^{-2}$
GP_{mcmc}	$\underline{3.78 \times 10^{-1} \pm 1.00}$	$1.65 \times 10^{3} \pm 5.75 \times 10^{3}$	$\mathbf{4.86 \times 10^{-5} \pm 7.63 \times 10^{-5}}$
RandomForest	4.25 ± 5.14	$1.76 \times 10^{3} \pm 1.36 \times 10^{3}$	$4.76 \times 10^{-3} \pm 4.85 \times 10^{-3}$
RandomSearch	3.05 ± 2.89	$3.05 \times 10^{3} \pm 3.12 \times 10^{3}$	$7.47 \times 10^{-3} \pm 4.72 \times 10^{-3}$

in the performance is more notable in BRVFL2 which has ranked third, higher than GP and with a small margin lower than Bohamiann. Moreover, BRVFL4 performance is almost as good as GP with the Borda score of 108 against 109 in GP. BRVFL1 and BRVFL3 have performed very similarly but slightly worse than the ones with skip-connections. The effect of evidence approximation on ERVFL methods is mostly negligible with ERVFL1 and ERVFL2 still standing lower and ERVFL3 and ERVFL4 ranking higher than Random Search. This suggests that the proposed method for setting the prior precision of all ensemble members based on only one member is not effective probably due to the fact that the parameters of prior in ensemble members are not similar.

The regret values during and at the end of optimization over each function also show a significant improvement in BRVFL-based methods compared with their fixed-precision counterparts. In 9 out of 12 functions, the RVFL-based methods have been among the top three. BRVFL3 has been the best once, on Bohachevsky, and BRVFL4 twice similar to GP, on 5d Rosenbrock and 10d Levy, whereas Bohamiann has never obtained the first place.

4.4.2 Hyperparameter Optimization

To test the performance of the proposed RVFL-based BOs on HPO problems we chose to apply them on the set of surrogate benchmark problems available in the HPOlib library [52]. Each problem in the benchmark corresponds to the application of a selected machine learning algorithm on a specific dataset. The surrogate is built using performance data collected from running the algorithm with various possible hyperparmeter configurations on the dataset. Experiments consisted of 30 trials per method per benchmark problem where each trial had a maximum budget of 50 iterations which is the default value of the HPOlib library for HPO problems. Table. 4.10 contains the complete list of model/dataset pairs. Due to the

poor performance of ERVFL methods over the synthetic functions we opt not to include them in the current set of experiments.

Table 4.10: Overview of models and datasets used for HPO benchmarking.

Model	No. Parameteres	Dataset
SVM	2	MNIST
Wide Resnet 2d	2	CIFAR10
CNN	5	CIFAR10
Paramnet	8	Adult
		Higgs
		Letter
		MNIST
		Optical Digists
		Poker

The overall performance of BO methods is presented in Figure 4.7 in the form of aggregated Borda scores. To compute these scores each experiment is performed for 30 independent runs. Each run is started from a different random initial design with an allocated budget of 50 function evaluations. At the end of the run the best validation error obtained during the optimization is recorded and used to rank the methods.

As it is evident in Figure 4.7 Random Forest has been ranked higher than all other methods except for GP_{mcmc}, which has shown to have the best overall performance in the HPO problems. On the other hand, BO with Baysian NN surrogate has been ranked worst of all. Amid RVFL-based methods BRVFL4, i.e. Bayesian RVFL with relu nonlinearity and skip connection, has demonstrated a fairly good overall performance by standing in the third place, higher than GP, whereas BRVFL1 has been the least effective method. The results indicate that models with *relu* activation functions are superior to the ones using *tanh*. On the other hand, presence of *skip connection* also contributes to the performance of RVFL-based BOs.

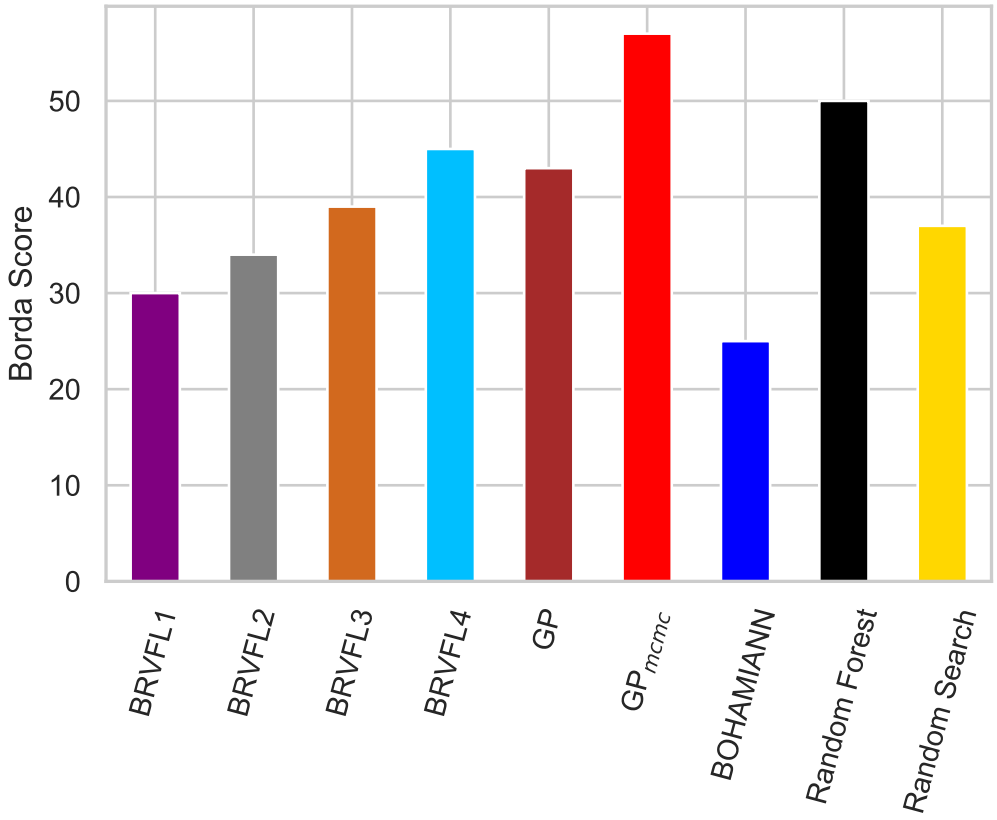


Figure 4.7: Aggregated Borda score of BOs method over the whole set of benchmark problems as measure of their performance.

Table 4.11: Validation errors induced by the best hyperparameter configurations found by BO methods for SVM, Wide Resnet and CNN models over MNIST, and CIFAR10 datasets.

Method	svm-MNIST mean $\pm$ std	wideresnet2d mean $\pm$ std	cnn-cifar10 mean $\pm$ std
BRVFL1	$2.0287 \times 10^{-2} \pm 8.6405 \times 10^{-3}$	$5.9018 \times 10^{-2} \pm 7.7218 \times 10^{-3}$	$1.6597 \times 10^{-1} \pm 1.0822 \times 10^{-2}$
BRVFL2	$1.9607 \times 10^{-2} \pm 8.1780 \times 10^{-3}$	$5.8747 \times 10^{-2} \pm 7.6693 \times 10^{-3}$	$1.7345 \times 10^{-1} \pm 2.4775 \times 10^{-2}$
BRVFL3	$3.2476 \times 10^{-2} \pm 1.2293 \times 10^{-2}$	$5.2432 \times 10^{-2} \pm 5.5952 \times 10^{-5}$	$1.6608 \times 10^{-1} \pm 1.5268 \times 10^{-2}$
BRVFL4	$3.2638 \times 10^{-2} \pm 1.2371 \times 10^{-2}$	$5.2441 \times 10^{-2} \pm 6.5810 \times 10^{-5}$	$\underline{1.6572 \times 10^{-1} \pm 1.0659 \times 10^{-2}}$
BOHAMIANN	$1.8896 \times 10^{-2} \pm 7.6396 \times 10^{-3}$	$5.3460 \times 10^{-2} \pm 9.8911 \times 10^{-4}$	$1.7832 \times 10^{-1} \pm 1.4038 \times 10^{-2}$
GP	$\underline{1.5349 \times 10^{-2} \pm 1.2611 \times 10^{-3}}$	$\mathbf{5.2382 \times 10^{-2}}$	$1.6707 \times 10^{-1} \pm 5.8834 \times 10^{-3}$
GP_{mcmc}	$\mathbf{1.5127 \times 10^{-2} \pm 4.4707 \times 10^{-4}}$	$\underline{5.2382 \times 10^{-2} \pm 5.0742 \times 10^{-7}}$	$\mathbf{1.6194 \times 10^{-1} \pm 6.9072 \times 10^{-3}}$
Random Forest	$4.4614 \times 10^{-2} \pm 1.6035 \times 10^{-1}$	$5.2912 \times 10^{-2} \pm 2.2700 \times 10^{-3}$	$\underline{1.6203 \times 10^{-1} \pm 5.4784 \times 10^{-3}}$
Random Search	$\underline{1.5754 \times 10^{-2} \pm 1.4833 \times 10^{-3}}$	$\underline{5.2382 \times 10^{-2} \pm 6.9149 \times 10^{-7}}$	$1.6928 \times 10^{-1} \pm 9.1168 \times 10^{-3}$

Table 4.12: Validation errors induced by the best hyperparameter configurations found by BO methods for Paramnet model over Adult, Higgs and Letter datasets.

Method	paramnet-adult mean $\pm$ std	paramnet-higgs mean $\pm$ std	paramnet-letter mean $\pm$ std
BRVFL1	$1.4896 \times 10^{-1} \pm 6.2342 \times 10^{-4}$	$2.8692 \times 10^{-1} \pm 2.9132 \times 10^{-3}$	$\underline{4.3942 \times 10^{-2} \pm 6.5744 \times 10^{-3}}$
BRVFL2	$1.4911 \times 10^{-1} \pm 1.1029 \times 10^{-3}$	$2.8671 \times 10^{-1} \pm 3.0866 \times 10^{-3}$	$\underline{4.3070 \times 10^{-2} \pm 6.2588 \times 10^{-3}}$
BRVFL3	$\underline{1.4882 \times 10^{-1} \pm 9.1561 \times 10^{-4}}$	$\underline{2.8354 \times 10^{-1} \pm 1.6313 \times 10^{-3}}$	$\mathbf{4.2939 \times 10^{-2} \pm 5.7651 \times 10^{-3}}$
BRVFL4	$\mathbf{1.4852 \times 10^{-1} \pm 1.0830 \times 10^{-3}}$	$\underline{2.8386 \times 10^{-1} \pm 2.1194 \times 10^{-3}}$	$4.8070 \times 10^{-2} \pm 2.3188 \times 10^{-2}$
BOHAMIANN	$1.4961 \times 10^{-1} \pm 7.8405 \times 10^{-4}$	$2.8608 \times 10^{-1} \pm 2.1377 \times 10^{-3}$	$5.5965 \times 10^{-2} \pm 1.1644 \times 10^{-2}$
GP	$1.4921 \times 10^{-1} \pm 8.6069 \times 10^{-4}$	$2.8734 \times 10^{-1} \pm 3.0183 \times 10^{-3}$	$4.9495 \times 10^{-2} \pm 1.0114 \times 10^{-2}$
GP_{mcmc}	$1.4911 \times 10^{-1} \pm 6.4692 \times 10^{-4}$	$2.8508 \times 10^{-1} \pm 2.8941 \times 10^{-3}$	$4.4366 \times 10^{-2} \pm 6.9121 \times 10^{-3}$
Random Forest	$\underline{1.4883 \times 10^{-1} \pm 1.1818 \times 10^{-3}}$	$\mathbf{2.8352 \times 10^{-1} \pm 2.3782 \times 10^{-3}}$	$4.7895 \times 10^{-2} \pm 1.0485 \times 10^{-2}$
Random Search	$1.4927 \times 10^{-1} \pm 9.0122 \times 10^{-4}$	$2.8505 \times 10^{-1} \pm 2.1943 \times 10^{-3}$	$6.6122 \times 10^{-2} \pm 2.4308 \times 10^{-2}$

Table 4.13: Validation errors induced by the best hyperparameter configurations found by BO methods for Paramnet model over MNIST, Optical Digists and Poker datasets.

Method	paramnet-MNIST mean ± std	paramnet-optdigits mean ± std	paramnet-poker mean ± std
BRVFL1	$\underline{1.6938 \times 10^{-2} \pm 7.6159 \times 10^{-4}}$	$\mathbf{1.7729 \times 10^{-2} \pm 1.0232 \times 10^{-3}}$	$8.3931 \times 10^{-3} \pm 1.5176 \times 10^{-2}$
BRVFL2	$\underline{1.6960 \times 10^{-2} \pm 6.4255 \times 10^{-4}}$	$\underline{1.7908 \times 10^{-2} \pm 1.2213 \times 10^{-3}}$	$\underline{4.5599 \times 10^{-3} \pm 9.1412 \times 10^{-3}}$
BRVFL3	$\mathbf{1.6876 \times 10^{-2} \pm 6.1069 \times 10^{-4}}$	$1.8445 \times 10^{-2} \pm 2.0566 \times 10^{-3}$	$\mathbf{1.5174 \times 10^{-3} \pm 1.4230 \times 10^{-3}}$
BRVFL4	$1.8994 \times 10^{-2} \pm 8.0493 \times 10^{-3}$	$1.8396 \times 10^{-2} \pm 3.9608 \times 10^{-3}$	$\underline{1.5971 \times 10^{-3} \pm 1.3639 \times 10^{-3}}$
BOHAMIANN	$1.8313 \times 10^{-2} \pm 1.5784 \times 10^{-3}$	$2.0971 \times 10^{-2} \pm 2.9924 \times 10^{-3}$	$1.5116 \times 10^{-2} \pm 1.8334 \times 10^{-2}$
GP	$1.7609 \times 10^{-2} \pm 1.1851 \times 10^{-3}$	$1.8099 \times 10^{-2} \pm 1.2732 \times 10^{-3}$	$2.4774 \times 10^{-2} \pm 3.1478 \times 10^{-2}$
GP_{mcmc}	$1.7380 \times 10^{-2} \pm 1.2299 \times 10^{-3}$	$\underline{1.7973 \times 10^{-2} \pm 1.5034 \times 10^{-3}}$	$7.8420 \times 10^{-3} \pm 1.3995 \times 10^{-2}$
Random Forest	$1.9257 \times 10^{-2} \pm 4.7268 \times 10^{-3}$	$1.8276 \times 10^{-2} \pm 1.9074 \times 10^{-3}$	$1.4926 \times 10^{-2} \pm 3.2396 \times 10^{-2}$
Random Search	$1.7830 \times 10^{-2} \pm 1.5579 \times 10^{-3}$	$2.0373 \times 10^{-2} \pm 2.5240 \times 10^{-3}$	$1.2285 \times 10^{-2} \pm 1.4672 \times 10^{-2}$

Tables 4.11 to 4.13 contain the performance data at a finer resolution, i.e. the mean and standard deviations of the final validation errors obtained by each BO method over each problem. As it could be seen, GP, GP_{mcmc} and Random Search are the three methods which could find configurations with lowest expected validation error in two-dimensional problems (Table. 4.11). GP_{mcmc} has been the best on SVM/MNIST and with a very small margin the second best after GP on Wide Resnet/CIFAR10. In both cases, Random Search could find configurations with third best expected validation error. In 5 dimensional CNN/CIFAR10 again GP_{mcmc} has dominated the rest, however, on this model/dataset pair Random Forest and BRVFL4 have produced the second and third best results, respectively. Yet, experiments over Paramnet model (Table. 4.12 and Table. 4.13) show the domination of BRVFL methods over the rest. In all mode/dataset pairs but Paramnet/Higgs where Random Forest has got the least mean validation error, the best results are produced by one of BRVFL methods. In this set of 8 dimensional problems, BRVFL3 has shown to be the most successful BO method in terms of mean validation error.

4.5 Discussion

The class of priors represented by Neural Networks have favourable characteristics which make them promising to be used as surrogate models in BO. However, Bayesian NN, as the most commonly used member of this class in the context of BO, has some practical limitations whose sources are rooted in the fact that most of inference algorithms are troubled when it comes to models with high number of parameters. A solution to this problem would be to only perform inference on a selected subset of the NN parameters. Our experiments show that Bayesian RVFLs as surrogate models in BO could perform comparably to GPs and Bayesian NNs in the optimization of Hyper parameters as well as synthetic functions. This is significant since similar results, in terms of quality, could be obtained using a surrogate with much cheaper inference procedure.

Our experiments show an absolute superiority of RVFLs with learned hyperparameters over the ones in which they are fixed. Moreover, among the RVFL-base methods the best overall results were given by models with skip-connection, i.e BRVFL2 and BRVFL4 in the synthetic functions and HPO problems, respectively. This is in line with the findings of [220] about the advantages of adding skip-connection to RVFLs. We also found that Bayesian RVFLs performed relatively well on higher dimensional problems. However, the reasons of this phenomenon need to be studied.

The overall performance of the *relu* networks was mostly better than *tanh* ones. This is interesting since in Bayesian RVFLs with *relu* nonlinearities by going further from the observed points the predictive variance gets larger and, unlike GPs and models with *tanh* activation, does not level-off. We expect this to exacerbate the *boundary over-exploration effect* [46], which already exists in GP-based BOs, by promoting the BO even more to sample close to the boundaries. Nevertheless, zero-mean GP and *tanh* Bayesian RVFLs create large plateaus in the regions of the search space that are far from current sample points. Whereas, these plateaus do not

exist in relu networks. Having said that, we could think of two probable reasons for the superiority of *relu* networks. On one hand, it is possible that the none-existence of plateaued regions might have overshadowed the downsides of the boundary over-exploration effect. On the other, it might be the result of our random parameter assignment method. While our weight normalization and bias assignment scheme works perfectly well with *relu* nonlinearities, it might not work as well if *tanh*s are used. In fact, in models with *sigmoid* activation functions we might have been better off with a random weight-generating methods such as the one presented in [46] which are specifically tailored for those nonlinearities.

4.6 Conclusion

This chapter addressed the scalability problem and high computational cost of using Monte Carlo-inferred BNNs in BO. It studied a set of BOs in which RVFLs are utilized as surrogates with efficient inference processes and introduced a novel Bayesian RVFL-based BO whose performance is better than or on par with the state-of-the-art BO methods on a set of synthetic functions and HPO problems. Furthermore, this work presented a simple scheme for initialization of the random parameters involved in RVFL networks.

This study examined the use of Bayesian RVFLs and (Bayesian) ensemble of *MAP* RVFLs as surrogates models in BO. Through the experiments it was shown that RVFL-based BOs whose hyper-parameter values were specified *a priori* had a poor overall performance, compared with BO methods that utilized Bayesian NN, GP or Random Forest surrogates. This included both Bayesian RVFL with constant prior precision and Bayesian ensemble of *MAP* RVFLs. However, in Bayesian RVFL this problem was resolved by inferring the hyperparameters from the observations. It was shown that BO with a Bayesian RVFL surrogate and evidence approximation over the hyperparameters outperforms the ones with Random Forest and GP (with

ML over its hyperparameters) over synthetic functions. Furthermore, the overall performance of such a BO was better than Bayesian NN-based BO over the set of HPO problems.

In this study we used low dimensional benchmark problem. It would be insightful to see how they could be compared with the existing methods over higher dimensional HPO problems. Moreover, we found that (*relu*) Bayesian RVFLs and ensemble of *MAP* RVFLs (with hyperparameters set fixed) have comparable performance. Although in this work we did not come up with a principled method for refining the hyperparameters of Bayesian ensemble of *MAP* RVFLs, the simplicity and parallelizablility of their training encourages further research in this direction. In a possible future work we are interested in observing the possible enhancement in the performance of the BO if a hyperparamter inference method is used along with the ensemble method. Another extension to the current work would be to apply a fully Bayesian method to infer the posterior over the parameters of the final layer.

Chapter 5

A Conditional Likelihood Acquisition Function

5.1 Introduction

In principle, if Bayesian optimization uses the Expected Improvement infill criterion and GP prior (Vanilla BO) under certain assumptions it will find the global optimum in continuous non-convex problems in an asymptotic regime (see M. Locatelli's seminal work on this [134] and also [201]). That is, in a noiseless environment, regardless of the initial design, it can eventually find the global optimum of a function with 100% certainty. However, in scenarios where the budget is limited, the convergence is not guaranteed. Therefore, performance of the BO is strongly contingent on having a good initial design. In other words, if the initial points are not representative of the properties of the underlying function, BO may fail to produce good results. Consequently, we might face circumstances in which BO could even be beaten by simple methods, such as trivial random search with only twice the budget [116, 129, 105]. The potential for deception by misleading initial points is a fundamental flaw of Vanilla BO and is germane to the two-stage process of its Optimization steps, in which the errors propagated from one stage to the next one remain

undealt. To circumvent this problem, in this chapter we study a new type of acquisition function for Bayesian Optimization which employs NNs to cope with inaccuracies that are propagated forward from the surrogate building process.

- We apply the notion of conditional maximum likelihood as an infill criterion to find new points in BO.
- We investigate the composition of the new acquisition function with different surrogate models and compare it with EI over a set of HPO benchmark problems.

5.2 Conditional Maximum Likelihood in GP

The source of the susceptibility of Vanilla BO to deception is said to be [18] that the hyperparameters are inferred solely from the available observations. A possible solution is therefore to somehow include a new (unobserved!) point into the hyperparameter estimation process involved in each optimization step. One way to achieve this, known as the *single stage* optimization method, has been suggested as a replacement for vanilla Bayesian optimization in [106]. If the function value for the global optimum , y^*, is known in advance, we could use the conditional likelihood as a credibility measure, and by maximizing it, simultaneously compute the hyperparameter values as well as the location of the next point to be evaluated. To this end, given a dataset of observations, $\mathcal{D} = (\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^n$ and the function value of the optimum point, y^*, we assume that the underlying function passes through the previous n points as well as a new point $(\hat{\boldsymbol{x}}, y^*)$ with unknown location $\hat{\boldsymbol{x}}$ which at this stage we expect to be the global optimum. The location of $\hat{\boldsymbol{x}}$ and the model hyperparameter values are inferred by optimizing their joint conditional likelihood:

$$\hat{\boldsymbol{x}}, \hat{\boldsymbol{\theta}} = \underset{\boldsymbol{x}, \boldsymbol{\theta}}{\operatorname{argmax}}\, p(\boldsymbol{x}, \boldsymbol{\theta} \mid \mathcal{D}, y^*) \tag{5.1}$$

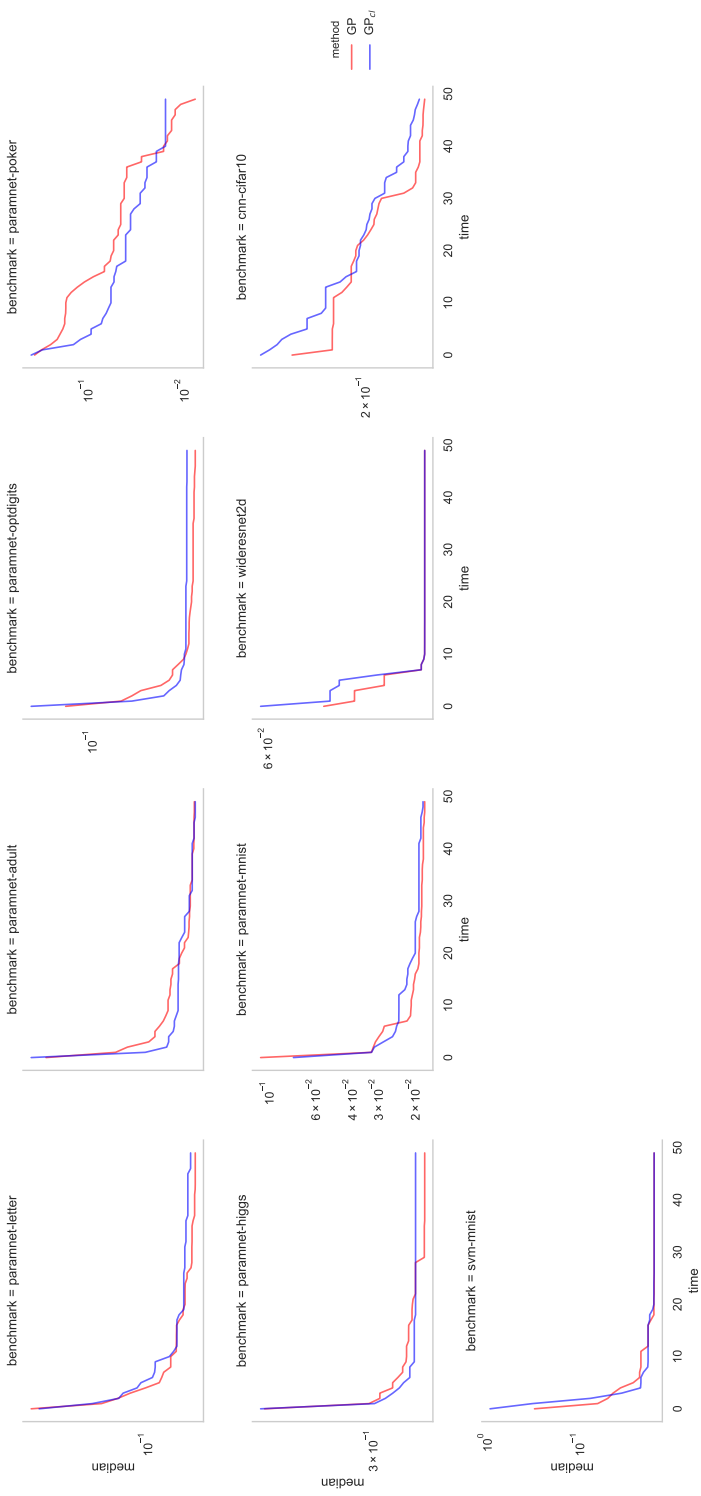


Figure 5.1: Comparsion of the median of validation errors induced from using the hyperparameter configurations proposed by GP with EI (red), and GP with the new acquisition function (GP_{cl}) (blue).

In a zero-mean GP this conditional likelihood could be computed analytically as follows[173]:

$$p(\boldsymbol{x}, \boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}, y^*, \sigma_n^2) = \frac{1}{(2\pi)^{\frac{n}{2}} \mid \mathbf{C} + \sigma_n^2 \mathbf{I} \mid^{\frac{1}{2}}} \exp(-\frac{1}{2}\bar{\boldsymbol{y}}^\intercal (\mathbf{C} + \sigma_n^2 \mathbf{I})^{-1} \bar{\boldsymbol{y}}) \quad (5.2)$$

$$\mathbf{C} = \mathbf{K} - \hat{\mathbf{k}}\hat{\mathbf{k}}^\intercal \ , \quad \bar{\boldsymbol{y}} = \boldsymbol{y} - \hat{\mathbf{k}}.y^*$$

where $\mathbf{K}$ and $\hat{\mathbf{k}}$ are defined as in section 2.3.2. For numerical reasons, maximizing the *conditional log-likelihood* is preferred to a direct use of the equation (5.1):

$$\log\ p(\boldsymbol{x}, \boldsymbol{\theta} \mid \boldsymbol{X}, \boldsymbol{y}, y^*, \sigma_n^2) = -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|) - \frac{1}{2}\bar{\boldsymbol{y}}^\intercal (\mathbf{C} + \sigma_n^2 \mathbf{I})^{-1} \bar{\boldsymbol{y}} \quad (5.3)$$

where $\hat{\mu}$ and $\hat{\sigma}^2$ are computed as follows:

$$\bar{\hat{\mathbf{k}}} = \mathbf{1} - \hat{\mathbf{k}}$$

$$\hat{\mu} = \frac{\bar{\hat{\mathbf{k}}}^\intercal \mathbf{C}^{-1} \bar{\boldsymbol{y}}}{\bar{\hat{\mathbf{k}}}^\intercal \mathbf{C}^{-1} \bar{\hat{\mathbf{k}}}} \quad (5.4)$$

$$\hat{\sigma}^2 = \frac{1}{n}(\bar{\boldsymbol{y}}^\intercal - \bar{\hat{\mathbf{k}}}\hat{\mu})\mathbf{C}^{-1}(\bar{\boldsymbol{y}} - \bar{\hat{\mathbf{k}}}\hat{\mu}) \quad (5.5)$$

In practice, (5.3) could be simplified further to get the concentrated form [173]:

$$\mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{\theta}) = -\frac{n}{2}\log(\hat{\sigma}^2) - \frac{1}{2}\log(|\mathbf{C}|), \quad \boldsymbol{\theta} \in \mathbb{R} \quad (5.6)$$

In an ideal scenario, where y^* is given, each optimization step would consists of *a)* going through equations (5.4),(5.5) and (5.6) until convergence condition is satisfied; *b)* evaluating the computed $\hat{\boldsymbol{x}}$ using the target function. Nevertheless, the assumption of having access to y^* is far fetched in many cases, therefore, in [106] it is suggested that in each optimization step the above process in performed multiple times, and each time y^* is

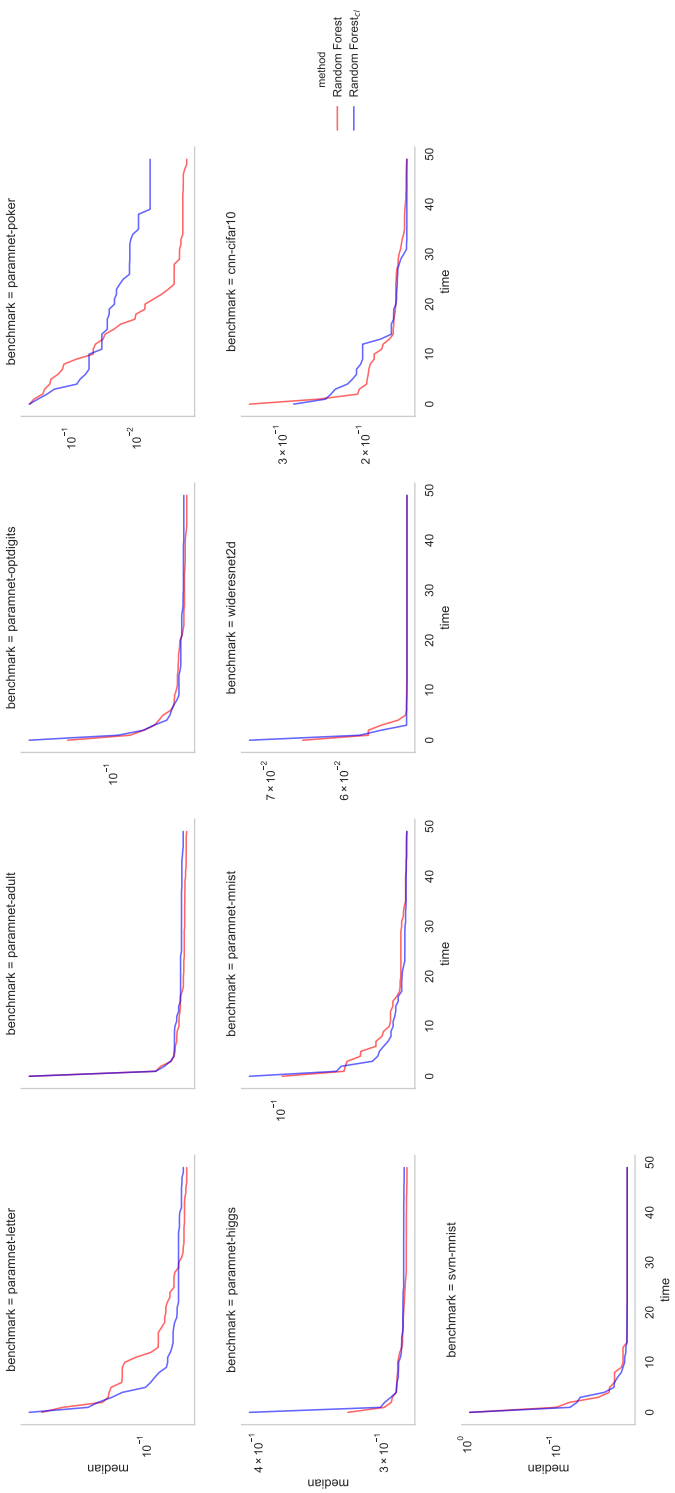


Figure 5.2: Comparison of the median of validation errors induced from using the hyperparameter configurations proposed by Random Forests with EI as well as Random Forests with the new acquisition function (Random Forests$_{cl}$).

subsumed by an arbitrary value that is smaller than the function value of the best point found so far, $y < y_{min}$.

This is a somewhat tedious process since in each optimization step the conditional maximum likelihood solution should be computed several times. Moreover, as reported in [173] it is prone to numerical problems. The problem is germane to GP with hyperparameters set via Maximum Likelihood, as having sample points close to the current set of observations might create an ill-conditioned covariance matrix. Interactions between $\hat{\boldsymbol{x}}$ and $\boldsymbol{\theta}$ further intensify the risk of ill-conditioning when using the conditional maximum likelihood approach.

In the following we will explain our proposed infill criterion, which also entails the solution to the above problems.

5.3 Conditional *Maximum a Posteriori* in Neural Networks

In the optimization problem of section 5.2, it was seen that Neural Networks have at least two advantages over Gaussian Processes. Firstly, using them we could easily sample from any location in the search space without worrying about numerical issues. Secondly, optimization of the loss function could be performed in a neat way using gradient descent via backpropagation. Here we present a method based on NNs for the same task. To this end, given a dataset of previously observed points $\mathcal{D}$ and the function value for the optimum point, y^*, we could find the next evaluation point by maximizing the joint posterior over the parameters of the Neural net and $\hat{\boldsymbol{x}}$, the point corresponding to y^*, where a flat (uniform) prior is defined over $\hat{\boldsymbol{x}}$ and NN's trainable parameters have a Gaussian or *elastic net* joint prior. This is equal to minimizing the regularized least square loss

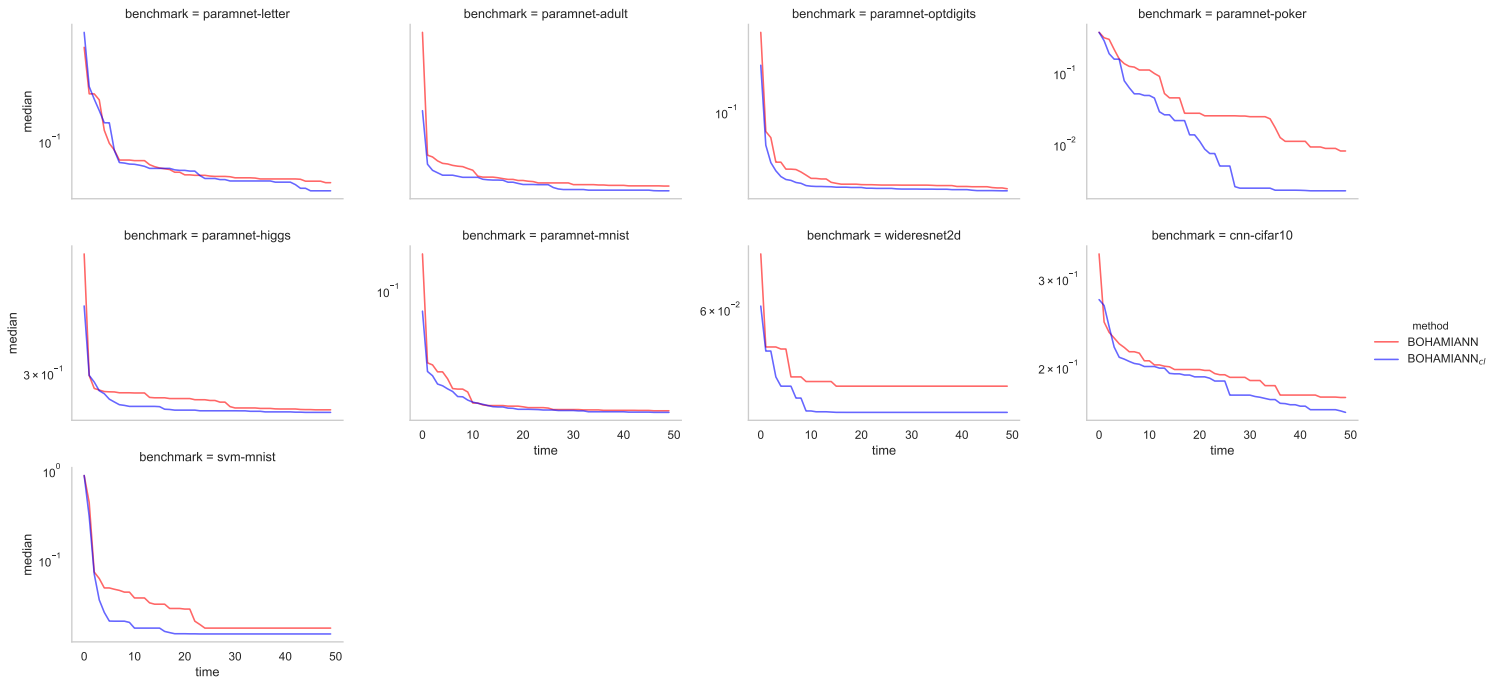


Figure 5.3: Comparison of the median of validation errors induced from using the hyperparameter configurations proposed by BOHAMIANN with EI as well as BOHAMIANN with the new acquisition function (BOHAMIANN$_{cl}$).

as follows:

$$\mathcal{L}(\hat{\boldsymbol{x}}, \boldsymbol{w}) = -\log p(\hat{\boldsymbol{x}}, \boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}, y^*) \tag{5.7}$$

$$\propto \frac{1}{n+1}\big[\sum_{i=1}^{n}(f_{\boldsymbol{w}}(\boldsymbol{x}_i) - y_i)^2 + (f_{\boldsymbol{w}}(\hat{\boldsymbol{x}}) - y^*)^2\big] + \lambda_0 \sum_{i=1}^{m} w_i^2 + \lambda_1 \sum_{i=1}^{m} |w_i| \tag{5.8}$$

where $\boldsymbol{w}$ is the vector of trainable parameters of NN, $f_{\boldsymbol{w}}(.)$ denotes the NN's function and λ_0 and λ_1 are tuning parameters. The advantage of using *elastic net* over *l2* regularization is that it encourages sparsity. This is specially desirable when the dimensionality of the input is higher than the number of observations [222].

If the value of $y*$ is unknown however (as is often the case), we would like to know how one might estimate it in a sensible way. Fortunately, vanilla BO gives us the means to compute the expected value of y^*:

$$\mathbb{E}[y^*] = y_{min} - \max_{\boldsymbol{x}\in\chi} \mathrm{EI}(\boldsymbol{x}) \tag{5.9}$$

where y_{min} denotes the best function value so far and EI(.) is the expected improvement function. $\max_{\boldsymbol{x}\in\chi} \mathrm{EI}(\boldsymbol{x})$ is obtained by performing a local search using EI, as the second stage of the optimization in Vanilla BO. Replacing $y*$ in (5.8) with $E(y^*)$ gives rise to a new acquisition function which could enjoy the advantages of both single and double stage optimization. Algorithm 4 shows the complete pseudo-code for a BO with the new acquisition function.

5.4 Experiments and Results

In this section we explain our experimental setup and results. The experiments are performed using HPOlib2 [50] on the set of hyperparameter optimization problems which are provided as surrogates of the corresponding (real) HPO problem. We have compared the new infill criterion with EI in combination with three state-of-the-art surrogates, namely Bayesian

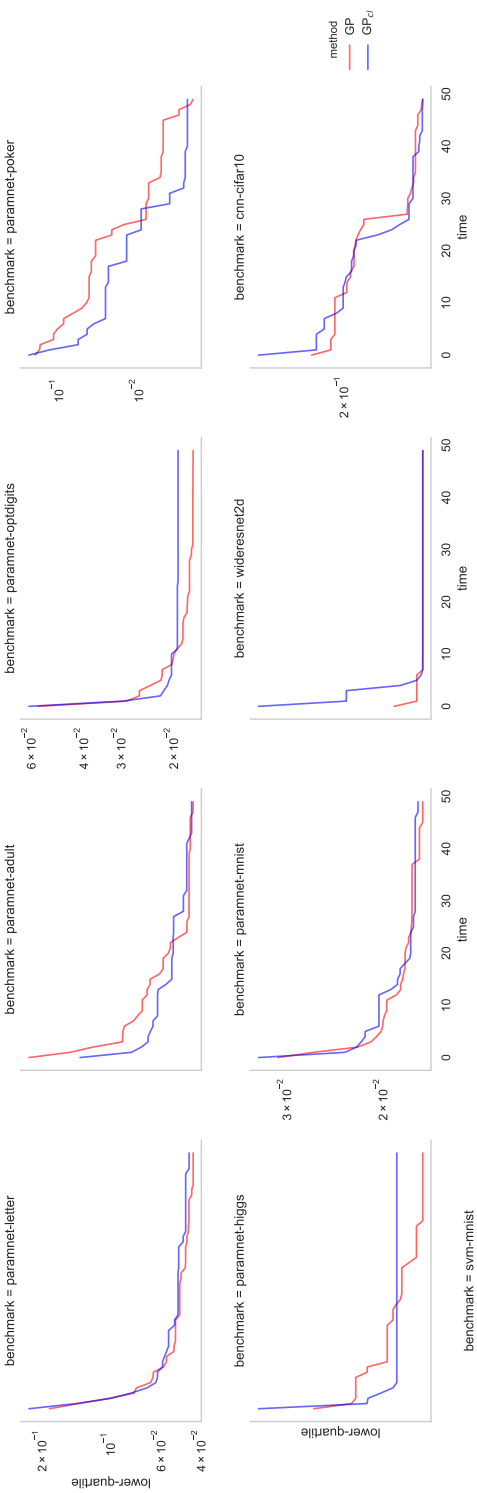


Figure 5.4: Comparison of the lower quartile of validation errors induced from using the hyperparameter configurations proposed by GP with EI as well as GP with the new acquisition function (GP_{cl}).

NNs with Hamiltonian Monte Carlo as inference method (BOHAMIANN), GP with MLE, and Random Forests. To compute the conditional *maximum a posteriori* estimates we have used a feed-forward NN with 2 hidden layers each having 50 $tanh$ units. Both λ_0 and λ_1 are set to be 5.0×10^{-5}. All methods use random search to optimize EI, and the rest of the parameters are left to their default values. Performance statistics for each experiment are gathered from 30 independent runs, where each run consists of 50 optimization steps.

Table 5.1 contains the median of final validation errors resulting from the combination of surrogates with each infill criterion. Methods that use the new infill criterion are specified by cl subscript. As Table 5.1 shows, the effectiveness of the new infill criteria changes from one surrogate mode to another. In BO with Bayesian NN surrogate, the new infill criterion has absolutely outperformed EI. In all benchmark problems BOHAMIANN$_{cl}$ has resulted in validation errors which are lower than those produced by BOHAMIANN. In the case of GP and Random Forests, however, this superiority could not be seen. For example in GP-based BOs, in all problems except for `Paramnet-adult`, either the original EI could reach to a better result or both infill criteria performed equally well. Likewise, in Random Forests, optimization of the hyperparameters of Convolutional NNs over CIFAR10 dataset (`cnn-cifar10`) has given the lowest median validation error, whereas in other problems EI has acted better than or as good as the new infill criterion. The performance statistics of these methods during the optimization process, presented in Figures 5.1 to 5.9, is almost in line with the results in Table 5.1.

It is interesting to speculate whether better performance when using the new acquisition function might be connected to the similarity between the model used in the acquisition function with that of the surrogate model, both of which are Bayesian NN in the successful cases. Another reason might be the lower performance of BOHAMIANN with EI infill criteria in comparison with the other BO methods.

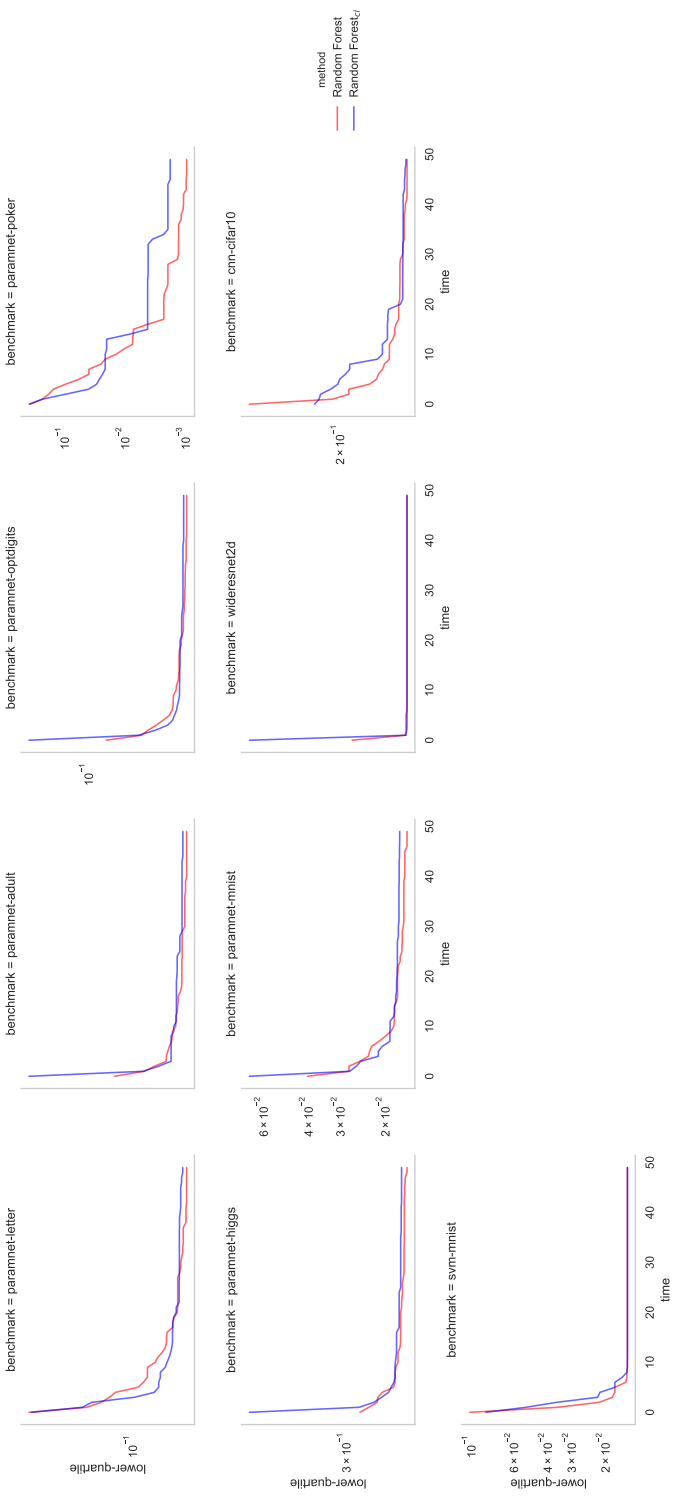


Figure 5.5: Comparison of the lower quartile of validation errors induced from using the hyperparameter configurations proposed by Random Forests with EI as well as Random Forests with the new acquisition function (Random Forests$_{cl}$).

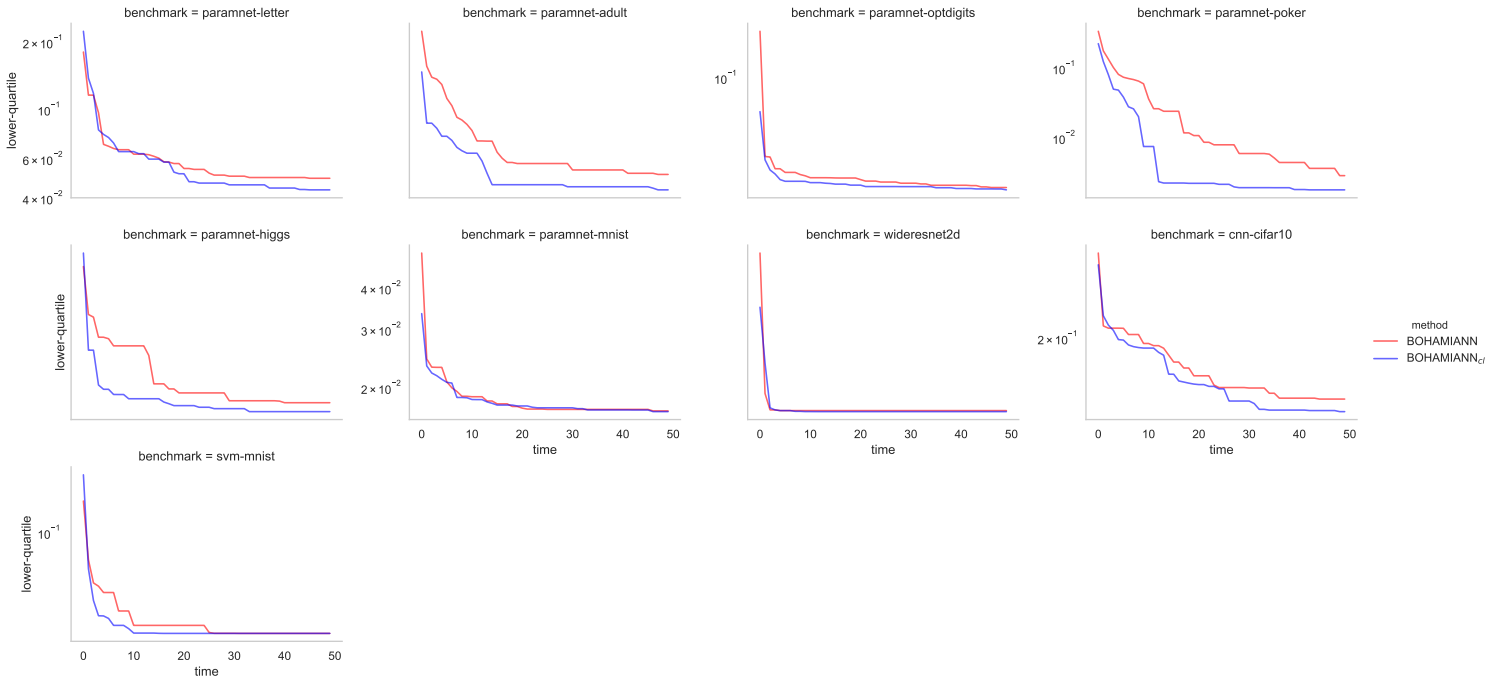


Figure 5.6: Comparison of the lower quartile of validation errors induced from using the hyperparameter configurations proposed by BOHAMIANN with EI as well as BOHAMIANN with the new acquisition function (BOHAMIANN$_{cl}$).

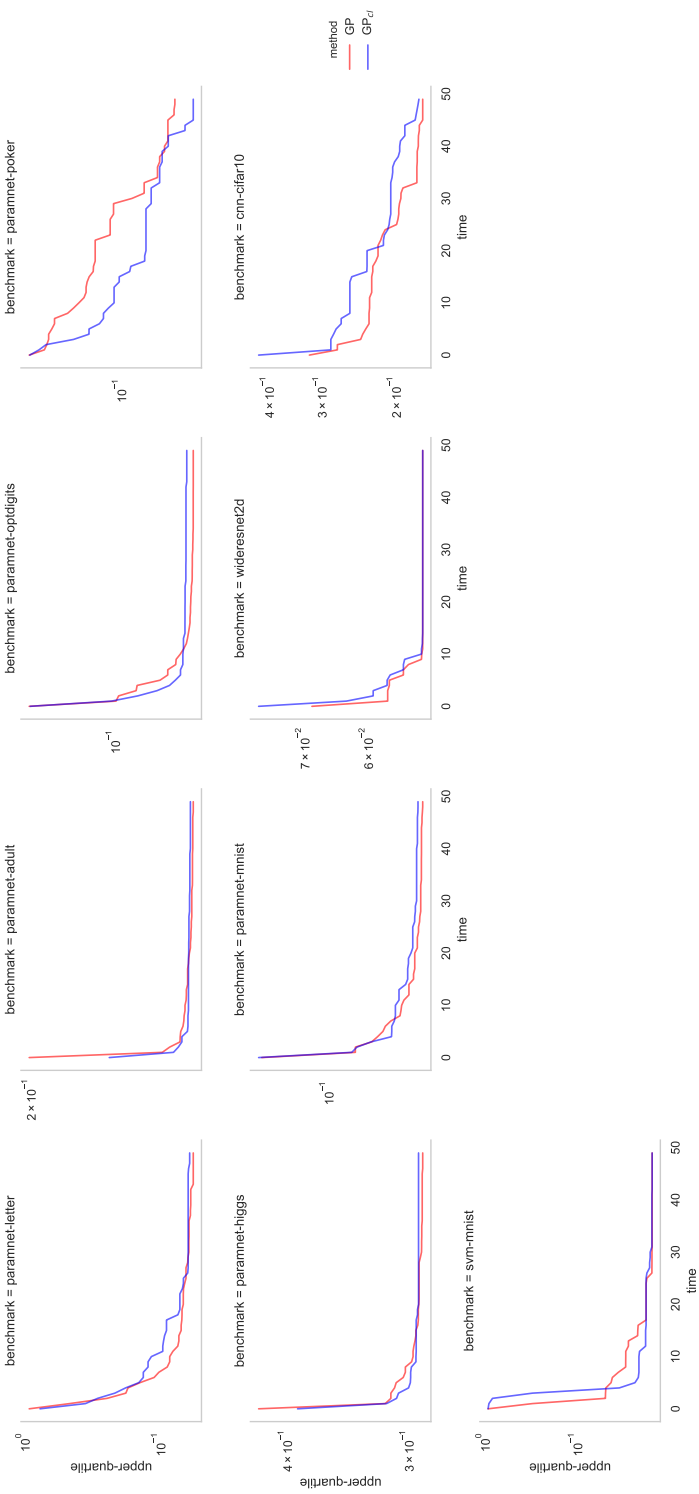


Figure 5.7: Comparison of the upper quartile of validation errors induced from using the hyperparameter configurations proposed by GP with EI as well as GP with the new acquisition function (GP_{cl}).

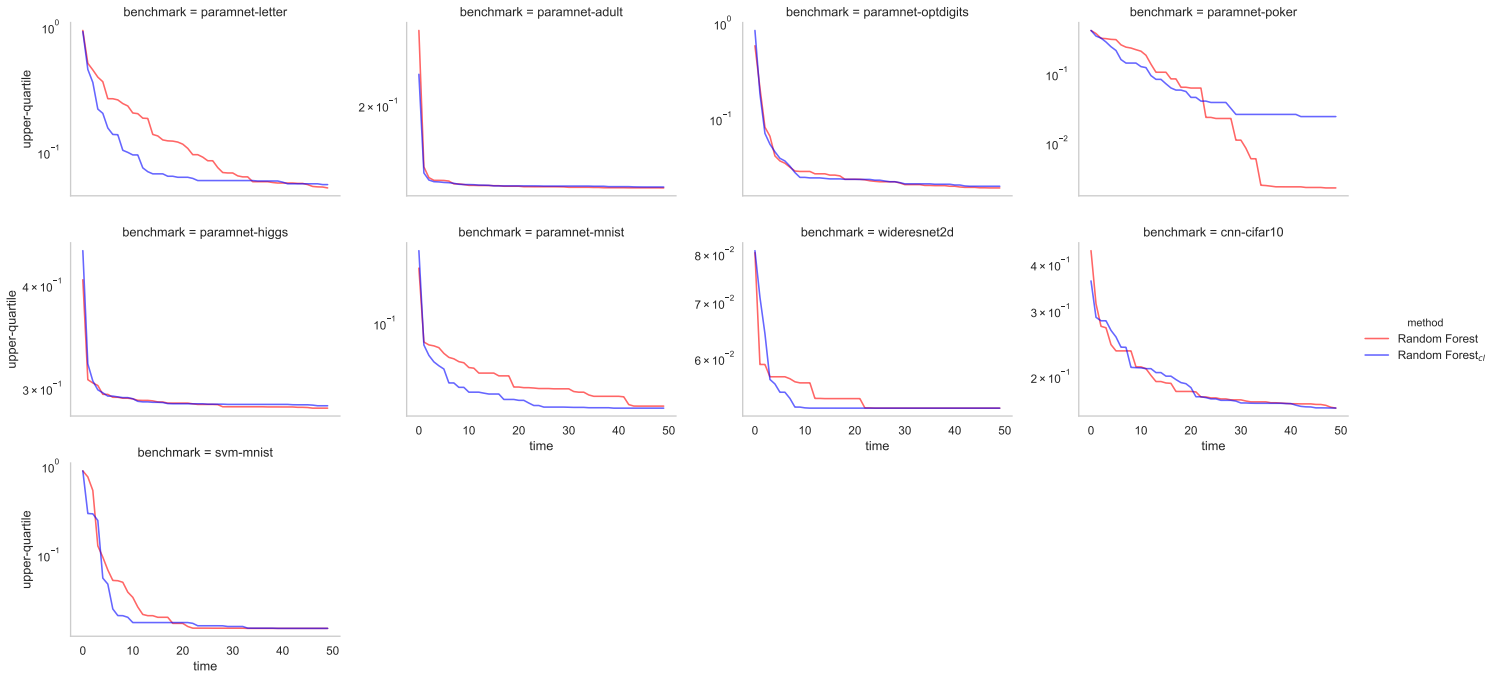


Figure 5.8: Comparison of the upper quartile of validation errors induced from using the hyperparameter configurations proposed by Random Forests with EI as well as Random Forests with the new acquisition function (Random Forests$_{cl}$).

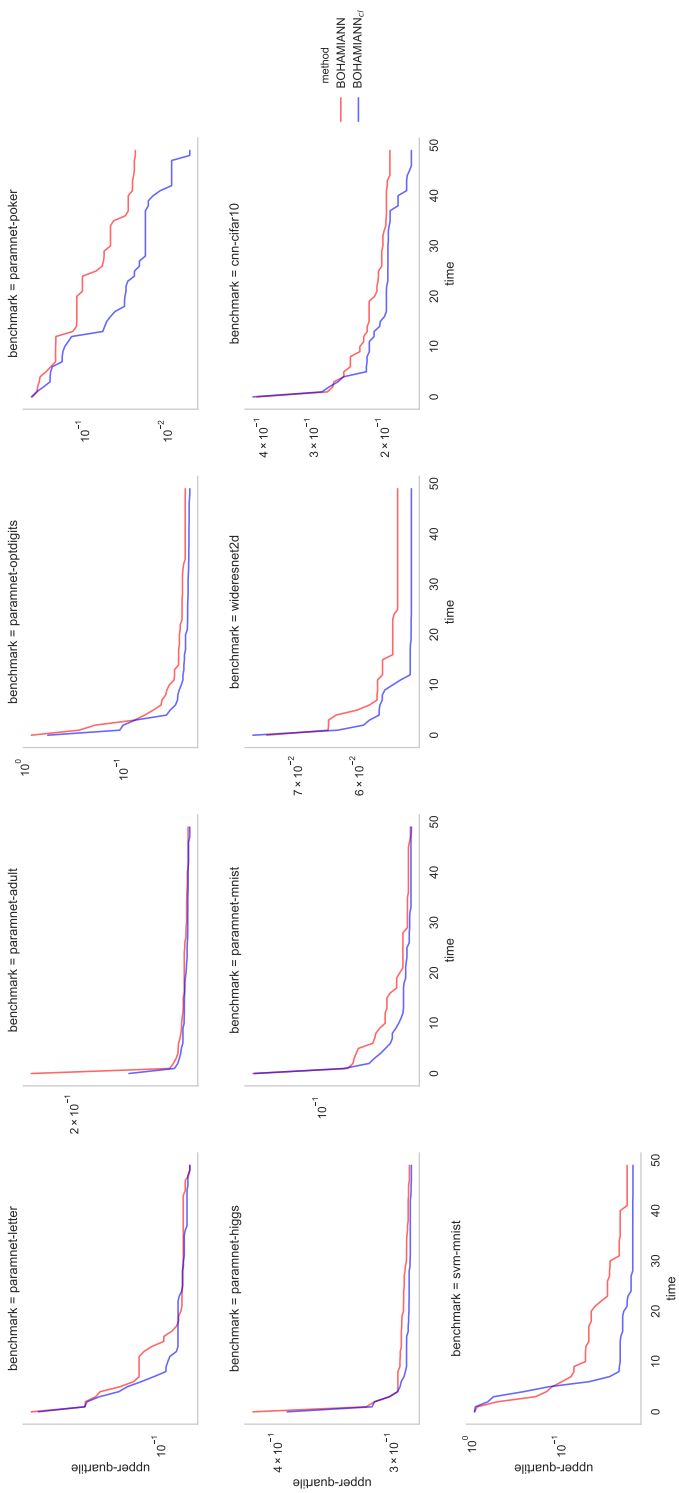


Figure 5.9: Comparison of the upper quartile of validation errors induced from using the hyperparameter configurations proposed by BOHAMIANN with EI as well as BOHAMIANN with the new acquisition function (BOHAMIANN$_{cl}$).

5.5 Conclusion

This chapter presented a novel infill criterion, building on the idea of *conditional maximum likelihood* suggested in Jones' *single-stage* optimization method. As opposed to Vanilla BO in which it is assumed that the surrogate model is an accurate model of the underlying function, the new infill criterion complements EI with a method that utilizes NNs to cope with possible inaccuracies in the surrogate model. In order to get a more precise estimate for the location of the best infill point to try, the new infill criterion uses the value of expected improvement to compute the conditional *max a posteriori* solution over the join space of the location of the optimum and parameters of a NN model.

To evaluate the proposed infill criteria, it was compared to EI on a set of 9 benchmark problems each of which includes hyperparameter optimization of a machine learning model on a benchmark dataset. The surrogate models used in the experiments consisted of Bayesian NNs (inferred by Hamiltonian Monte Carlo), GP (with Maximum likelihood estimate over its parameters) and Random Forests. Our studies showed that when used with a Bayesian NN surrogate, the new infill criterion performed better than EI. However, if used with GP or random-forest, the new infill criterion does not provide any advantage over EI, probably because the surrogate model and the NN used in the infill criteria represent different assumptions about the smoothness characteristics of the underlying functions. This study established a solution to utilize the idea of single-stage optimization in a new acquisition function to cope with inaccuracies that are propagated forward from the surrogate building process. Furthermore, it provided a strait-forward way to accomplish this task using NNs. Although the experiments did not indicate a decisive dominance of the proposed infill criterion over EI, its limited success in BOs with NN-based surrogates encourages further research in this direction, for example by trying different regularization methods for the infill criterion's NN.

Algorithm 4: Pseudo-code for BO with the new acquisition function

Input: $f(.)$: the black-box function
$\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^{n_0}$: initial design
n: Number of optimization steps
$p_0(f)$: a prior over f
EI$(.)$: Expected Improvement function
$\boldsymbol{w}$: set of parameters of a NN

Output: $\boldsymbol{x}_{min}$: next point to sample from

$i \leftarrow 0$;
$y_{min} \leftarrow \min(y_0, .., y_{n_0})$;
while $i < n$ **do**
  Compute $p_i(f \mid \mathcal{D})$ given $\mathcal{D}$ and $p_0(f)$;
  $\text{EI}_{max} \leftarrow \max \text{EI}(f, p_i(f \mid \mathcal{D}))$;
  $\mathbb{E}[y^*] \leftarrow y_{min} - \text{EI}_{max}$;
  $\boldsymbol{x}_{n_0+i}, \_ \leftarrow \text{argmax}_{\boldsymbol{x},\boldsymbol{w}} \log p(\boldsymbol{x}, \boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}, \mathbb{E}[y^*])$;
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{x}_{n_0+i}, f(\boldsymbol{x}_{n_0+i}))\}$;
  if $f(\boldsymbol{x}_{n_0+i}) < y_{min}$ **then**
    $y_{min} \leftarrow f(\boldsymbol{x}_{n_0+i})$;
    $\boldsymbol{x}_{min} \leftarrow \boldsymbol{x}_{n_0+i}$;
  end
  $i \leftarrow i + 1$
end

Table 5.1: Comparison of performance of the new infill criterion with EI's in three different BO methods. Figures show the median of validation errors from 30 independent runs. The best result in each row is depicted in bold.

method benchmark	BOHAMIANN	BOHAMIANN$_{cl}$	GP	GP$_{cl}$	Random Forest	Random Forest$_{cl}$
cnn-cifar10	0.174359	0.162855	0.166174	0.168873	0.161916	**0.161756**
paramnet-adult	0.149523	0.148949	0.149375	0.149323	**0.148769**	0.149287
paramnet-higgs	0.285667	0.284689	0.287129	0.289287	**0.283275**	0.285095
paramnet-letter	0.0548847	0.0487449	0.0479195	0.0513795	**0.0459634**	0.0489597
paramnet-mnist	0.018125	0.017715	**0.01766**	0.01799	0.01772	0.017765
paramnet-optdigits	0.0205953	0.0197506	**0.0179002**	0.0205149	0.018222	0.019469
paramnet-poker	0.00813595	0.00225279	0.00735737	0.0149929	**0.0016057**	0.00583595
svm-mnist	0.0174945	0.0150458	0.0150458	0.0150458	0.0150458	0.0150458
wideresnet2d	0.054226	0.052382	0.052382	0.052382	0.052382	0.052382

Chapter 6

Implicit models and order preservation

6.1 Introduction

This chapter develops a novel and efficient way of building Bayesian neural network surrogates in Bayesian Optimization. In this study, we propose a new approach to reduce the computational cost of building Bayesian Neural Network surrogates. To this end, rather than running an expensive Markov Chain to infer the parameters of a Bayesian neural network, we make use of the variational optimization and employ a highly expressive approximate distribution from the family of deep generative models, also known as an "implicit" model. The contributions of this chapter are two-fold. First, we present the notion of *order-preserving* implicit models, a deep generative model in which the order of the level-sets in the distribution defined over the latent space is kept consistent with the ones in the target distribution - a desirable property for our application. Then, we use order-preserving models under variational Bayesian framework to infer the parameters of a Bayesian neural network surrogate. This new inference scheme falls into the category of *Hypernets*, neural networks that generate neural networks. We refer to the Bayesian NNs inferred using

this scheme as *Implicit Variational Bayesian neural network*s and show that they usually out-perform MCMC Bayesian NNs in Bayesian optimization of hyperparamters. To make this study more comparable with other works in the literature of implicit models, we decided to adopt the naming convention which is more commonly used in this context, rather than keeping the notation consistent with the previous chapters.

6.2 Deep Generative Models

Variational inference (VI) is an instrumental part of the Bayesian inference toolkit and a computationally efficient alternative to Markov Chain Monte Carlo methods. This efficiency comes from the fact that in VI the inference is performed by means of optimization rather than sampling, the approach taken by MCMC. However, the decision on picking one over another is usually a compromise on efficiency over accuracy. On one hand, MCMC methods are able to produce an asymptotically exact estimation of a target (posterior) distribution, but are short on scalability. On the other hand, although VI is scalable and fast, compared to MCMC, it usually results in biased estimates. A combination of two factors, and their interaction, contribute to the sub-optimal behaviour of VI. (1) The application of strong over-simplified assumptions on the characteristics of the true posterior. (2) The use of KL-divergence to measure the closeness of the variational distribution to the true posterior.

Recalling from 2.3.1, in VI, assumptions about the characteristics of the true posterior are expressed by selecting a parametric form for the approximating distribution. Then, inference is performed by fine-tuning the parameters of this distribution in order to minimize the KL-divergence between the approximating and the true distributions. Traditionally, to reduce the complexity of the variational optimization, the approximating distribution is chosen from a factorized family. This prohibits them from being able to recover complicated posteriors due to their limited expres-

sive power. Furthermore, the KL-divergence encourages the variational distribution to prioritize high probability regions of the true posterior. In other words, the representational capacity of the variational distribution is devoted to recovering as much probability mass (from the posterior) as possible. If the gap between the complexity of the true posterior and the capacity of the variational distribution is wide, variational optimization leads to the so-called 'mode-seeking' behaviour [179, 148, 218], meaning that the limited capacity of the approximating distribution is only enough to encompass the probability mass around the posterior's modes. Depending on the complexity of the true posterior, the effectiveness of VI is thus greatly dependent on the representational power of its approximating distribution.

Amongst many approximating models that have been studied in VI, deep neural density estimators [41, 165, 90, 34] have the advantage of being both scalable and yet highly expressive (they are able to provide complex representations of data or distributions). Therefore, they have attracted considerable attention and have become a means to construct arbitrarily complex approximating distributions that could asymptotically recover the target distribution. This high level of complexity is usually achieved by squashing (transforming) a simple probability distribution (such as an isotropic Gaussian) through a parametric transformation of the form $\mathcal{G}_{\boldsymbol{\beta}} : \mathbb{R}^m \rightarrow \mathbb{R}^d$, specified by a deep NN:

$$\boldsymbol{\theta} = \mathcal{G}_{\boldsymbol{\beta}}(\boldsymbol{z}), \quad \boldsymbol{z} \sim p(\boldsymbol{z}) \tag{6.1}$$

$\boldsymbol{z} \in \mathbb{R}^m$, is a random variable from a simple distribution ($p(\boldsymbol{z})$ could be treated as either noise or latent variable [198]), transformed to a variable from the target domain, $\boldsymbol{\theta} \in \mathbb{R}^d$, through a nonlinear *generator* function, $\mathcal{G}_{\boldsymbol{\beta}}$, with $\boldsymbol{\beta}$ being its vector of parameters. Subsequently, the corresponding transformed density $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ is defined as the derivative of the cumulative distribution function [153]:

$$q_{\boldsymbol{\beta}}(\boldsymbol{\theta}) = \frac{\partial}{\partial \theta_1} \ldots \frac{\partial}{\partial \theta_d} \int_{\{\mathcal{G}_{\boldsymbol{\beta}}(\boldsymbol{z}) \leq \boldsymbol{\theta}\}} p(\boldsymbol{z}) \, d\boldsymbol{z} \tag{6.2}$$

	mean-field (Gaussian)	flow-based	implicit
Representational Power	✓	✓✓	✓✓✓
Tractable $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$	✓	✓	✗
Tractable $p(\boldsymbol{\theta})$.	✓✓	✓✓	✓
Intractable $p(\boldsymbol{\theta})$.	✗	✗	✓

Table 6.1: Comparison of mean-field, flow-based and implicit variational inference. The representational power of implicit models is deemed to be higher than flow-based models as they are able to encompass information from a high dimensional space into a low dimensional input space. However, implicit models are only able to deal with tractable target distribution, if only they have a tractable sampling process.

in which θ_i is the i^{th} element of $\boldsymbol{\theta}$.

Based on the specifications of $\mathcal{G}_{\boldsymbol{\beta}}(.)$, deep generative models could be divided into two main classes: *prescribed* and *implicit* models [40]. *Prescribed* models, also known as *flow-based generative models* [84], have tractable probability distribution functions, as $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ is analytically computed from equation (6.2). This tractability is attained when the transformation satisfies a set of conditions: (1) it has an equidimensional input and output, $m = d$, (2) it is invertible and smooth, and (3) the determinant of the Jacobian is cheap to compute [180]. The application of flow-based models in VI is relatively straightforward - during the variational optimization, expectations are taken under the approximating distribution and evaluation is performed using both the approximating as well as the target distributions. In this inference scheme, the density of the target distribution is available up to a normalizing constant, but it does not need to be an easy-to-sample distribution.

Implicit models are alternatives to flow-based models, with the well-known Generative Adversarial Network (GAN) [71] being the most famous

member of the family. The transformations used in implicit models do not have the restrictions imposed on flow-based models: $\mathcal{G}_{\boldsymbol{\beta}}(.)$ does not need to be invertible, and the dimensionality of the random variable $\boldsymbol{z}$ at the "input" is usually much lower than dimensionality of the transformed one: $m << d$. As a consequence however, the specifications of the approximating distribution are not explicitly parameterised, and its density function $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ is intractable.

6.3 Likelihood-free Inference

Implicit models are primarily used in *likelihood-free inference* problems [153, 198, 14] where the goal is to approximate intractable or unknown *likelihood function*s. This is a challenging scenario since both the approximating and the target distributions have intractable densities. This is despite there being a tractable data-generating process, meaning that samples could be drawn from both distributions. However, a direct evaluation of the density for samples (from either density) is not possible.

To this end, in an iterative two-step process, first an adversarial model is trained using samples drawn from both distributions. This model, namely *discriminator* or *critic*, is usually a feed-forward NN and acts as an approximation to the divergence between the two distributions. Subsequently, training the generator/implicit model using the critic reduces the divergence measure between the approximating and the target distribution. Common forms of loss for the critic and the implicit model are presented in equations (6.3) and (6.4), respectively.

$$\mathcal{L}(\mathcal{C};\mathcal{G}) = \mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\log \sigma(\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta}))] + \mathbb{E}_{p(\boldsymbol{\theta})}[\log 1 - \sigma(\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta}))] \tag{6.3}$$

$$\mathcal{L}(\mathcal{G};\mathcal{C}) = -\mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta})] \tag{6.4}$$

in which $\mathcal{C}$ denotes a critic parameterised by $\boldsymbol{\gamma}$, $\mathcal{G}$ is the implicit model, $\mathcal{L}(\mathcal{C};\mathcal{G})$ denotes the loss of the critic given the generator function (implicit

model) and its parameters, $p(\boldsymbol{\theta})$ is the target distribution and $\sigma(.)$ is a transfer function.

Two of the most broadly used divergence measures in this learning scheme are *Jensen–Shannon* and *Kullback–Leibler* divergences. The former is attained by employing a logistic regression model as critic and setting $\sigma(.)$ to be the identity function, $\sigma(\boldsymbol{x}) = \boldsymbol{x}$, whereas, for the latter $\mathcal{C}$ and $\sigma(.)$ are required to be a regression model and sigmoid function, respectively. Along with these two, any statistical divergence from the class of *f-divergence* is available to the adversarial training strategy as well [159].

One aspect of this GAN-style adversarial training is that the implicit distribution's objective function depends solely on the critic's output [193], which does not put any restriction on how the implicit model's input is mapped into its output space [26]. For example, there is no guarantee that a sample located near the mode of the implicit model's input distribution, $p(\boldsymbol{z})$, is mapped to a high probability region of the target distribution. Besides, several input locations might be mapped to the same output location. This is the root to some of the most well known pathological behaviors of implicit models such as *mode collapse*, in which the implicit model only recovers some of the modes of the target distribution, and *highly entangled* input space, where there is no semantically meaningful correspondence between information in each dimension of the input space and characteristics of the target distribution.

6.4 Order-preserving Implicit Models

For some problems, in addition to the generated samples, there is more information about the target distribution. This extra information could be available through pairwise preferences [111], also known as *implicit feedback*s. This suggests that even though direct access to the target distributions density is impossible, a coarser level of information could be attained from a noisy pairwise comparison function which assigns binary

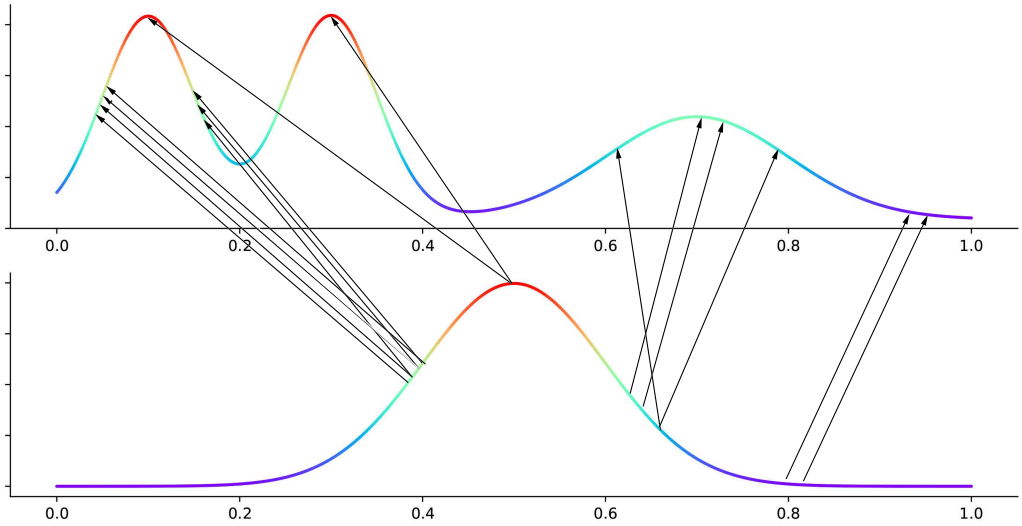


Figure 6.1: Using a *order-preserving* implicit model, points under a simple input distribution $p(\boldsymbol{z})$ (bottom) are mapped to the ones under a more complex transformed distribution $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ (top) in a manner that the order between level sets is not changed under the transformation (level sets are demonstrated using colors). As it can be seen, samples around the mode of $p(\boldsymbol{z})$ are mapped to points close to the modes of the transformed distribution. Likewise, low probability regions of the input distribution are also mapped to a low probability region under the target density .

preferences $\{0, 1\}$, termed *duels* [2, 70], to pairs of competing samples $[\boldsymbol{\theta}, \boldsymbol{\theta}']$ generated by the implicit model.

These pairwise preferences could be used in building an *order-preserving* implicit model where, under an asymptotic regime, the order of the density level-sets is kept intact under the transformation of $p(\boldsymbol{z})$ to $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ through $\mathcal{G}_{\boldsymbol{\beta}}(.)$. In such a model, mode-collapse is prevented to a high degree and since the relationship between distributions in the input and output of the generator function is fully specified, sampling can be performed in a controlled manner.

For a probability density function, $p(\boldsymbol{x})$, defined over a continuous

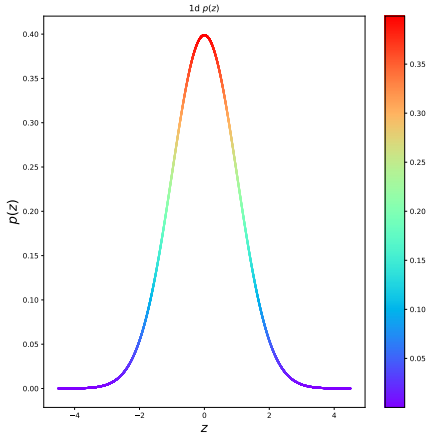


Figure 6.2: In the examples, $p(\boldsymbol{z})$ is chosen to be a unidimensional standard Gaussian. Consequently, each level set in $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ is (at most) covered by two $\boldsymbol{z}$s.

domain, $\boldsymbol{x} \in \boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^k$, a λ-level set is the set of all point whose probabilities are equal to λ [27]:

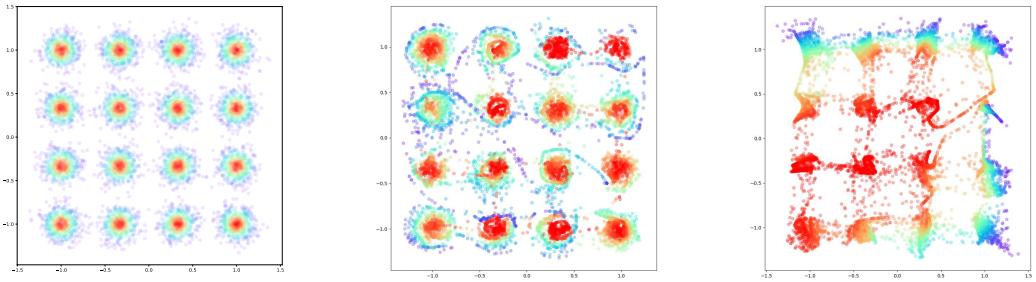
$$G(\lambda; p(\boldsymbol{x})) = \{\boldsymbol{x} \in \boldsymbol{\mathcal{X}} \ : \ p(\boldsymbol{x}) = \lambda\}, \quad \lambda \geq 0 \tag{6.5}$$

Subsequently we give our definition of the *order-preserving* implicit model as a generative neural sampler whose generator function satisfies the following conditions:

$$\begin{aligned} G(\nu; p(\boldsymbol{\theta})) \ &= \ \mathcal{G}_{\boldsymbol{\beta}}(G(\lambda; p(\boldsymbol{z}))) \\ \forall \lambda_i, \lambda_j \geq 0: \quad \lambda_i \geq \lambda_j \ &\Rightarrow \ \nu_i \geq \nu_j \end{aligned} \tag{6.6}$$

in which $G(\nu; p(\boldsymbol{\theta}))$ denotes the ν-level set resulted from projecting members of the λ-level set into the $\boldsymbol{\theta}$ space using the generator function $\mathcal{G}_{\boldsymbol{\beta}}(.)$.

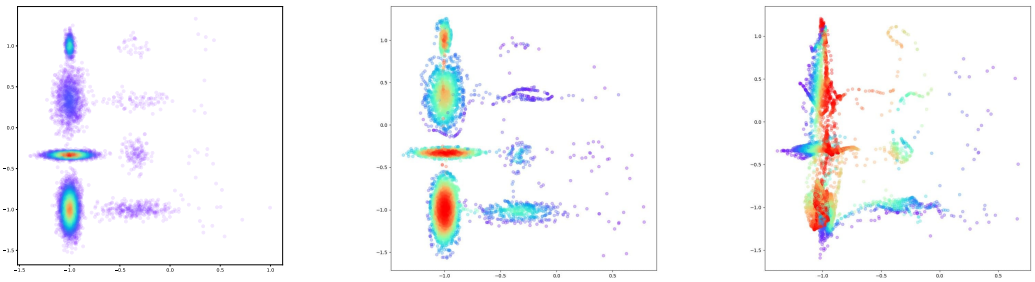
To get more insight into the behaviour of order-preserving implicit models, lets assume we are approximating a multi-modal target distribution using a model with a unidimensional Gaussian input (see Figure 6.1). At the end of the training, one might wish to sample from the modes of the



(a) (b) (c)

Figure 6.3: Comparison of order-preserving (center) and a simple (right) implicit distributions on a grid shaped target distribution, $p(\boldsymbol{\theta})$. (a) Ground truth. (b) Samples from an order-preserving implicit distribution. (c) Samples from a simple implicit distribution trained over $p(\boldsymbol{\theta})$ using KL-divergence. Colors indicate the probability of the generated samples under $p(\boldsymbol{z})$.

approximating distribution. Under an order-preserving model, mode of the input distribution, $p(\boldsymbol{z})$, is likely to be mapped into the modes of the target distribution. Hence, repeatedly transforming $\operatorname{argmax} p(\boldsymbol{z})$ through $\mathcal{G}_{\boldsymbol{\beta}}(.)$ should result in a mode-hopping effect under $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$. With some abuse of notation, these are the points $\boldsymbol{\theta} \sim \mathcal{G}_{\boldsymbol{\beta}}(G(\lambda; p(\boldsymbol{z})))$ where $\lambda = \max p(\boldsymbol{z})$. Clearly, in this example (and other similar circumstances) $q_{\boldsymbol{\beta}}(\boldsymbol{\theta} \mid \boldsymbol{z})$ is not a point mass, consequently $\mathcal{G}_{\boldsymbol{\beta}}(.)$ is required to be an stochastic function. An alternative approach is to use a deterministic generator but inject the stochasticity into its input [26]. In such a case, the input space could be decomposed into a latent factor, $\boldsymbol{z} \sim p(\boldsymbol{z})$, and a noise part, $\boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$, which results in a generator function of the form $\mathcal{G}_{\boldsymbol{\beta}}(\boldsymbol{z}, \boldsymbol{\epsilon})$. Figures 6.3, 6.4, 6.5, and 6.6 compare such an order-preserving model and simple implicit ones in approximating three target distributions with different structures. Models in all three examples use $2d$ inputs where the values of the first and second input dimension are sampled from a standard Gaussian (see Figure 6.2) and



(a) (b) (c)

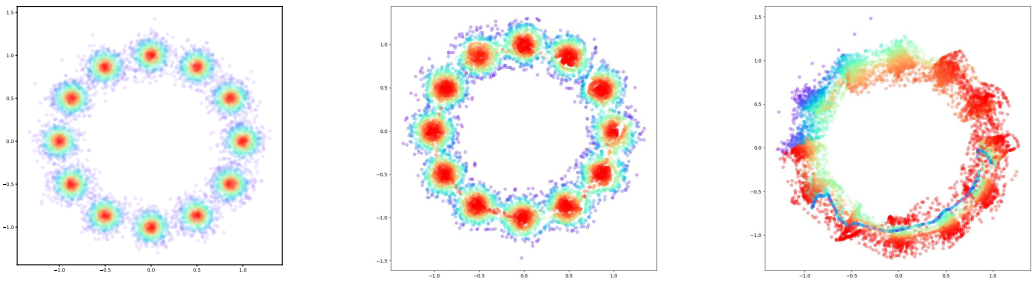
Figure 6.4: Comparison of order-preserving (center) and a simple (right) implicit distributions on a $p(\boldsymbol{\theta})$ which is a mixture of *2d* Gaussians with various standard deviations. (a) Ground truth. (b) Samples from an order-preserving implicit distribution. (c) Samples from a simple implicit distribution trained over $p(\boldsymbol{\theta})$ using KL-divergence. Colors indicate the probability of the generated samples under $p(\boldsymbol{z})$.

uniform distribution in $[0, 2\pi]$, respectively. In the order-preserving models (Figures 6.3-b, 6.4-b and 6.5-b), the first input dimension is considered to contain the latent code, however, in the simple implicit model both inputs are treated the same.

6.4.1 Modeling Pairwise Preferences

One way to model pairwise preferences is through using a utility function which induces a complete order over its input space [96]. This function is used to assign a utility to each member of a duel [70]. These utilities could be arbitrarily interpreted as the level of desirability of samples with respect to some criteria. Using utilities associated with each member of a duel, an unobservable joint utility is produced for the pair. Here we assume this quantity to be the difference between individual utilities of the pair:

$$r(\boldsymbol{\theta}, \boldsymbol{\theta}') = u(\boldsymbol{\theta}') - u(\boldsymbol{\theta}) \tag{6.7}$$



(a) (b) (c)

Figure 6.5: Comparison of order-preserving (center) and a simple (right) implicit distributions on a mixture of *2d* Gaussians positioned on the perimeter of a circle. (a) Ground truth. (b) Samples from an order-preserving implicit distribution. (c) Samples from a simple implicit distribution trained over $p(\boldsymbol{\theta})$ using KL-divergence. Colors indicate the probability of the generated samples under $p(\boldsymbol{z})$.

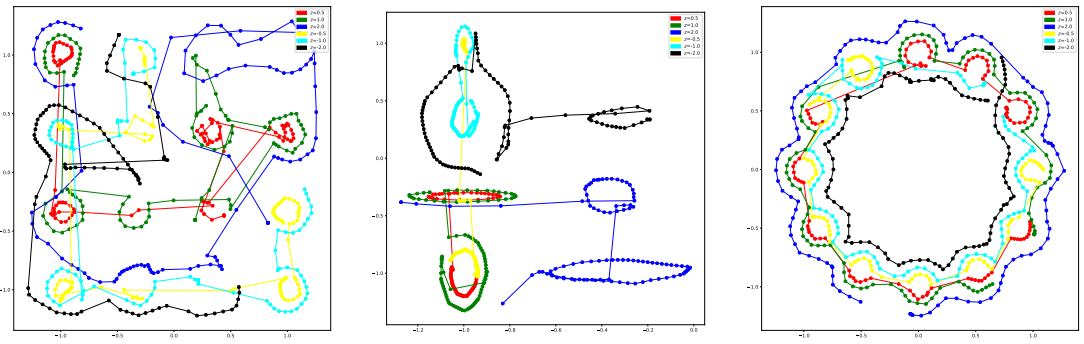
where $u : \mathbb{R}^d \to \mathbb{R}_+$ denotes a utility function and $r : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is the joint utility. We assume that the value of the joint utility is unobserved and the utility system is susceptible to noise. Therefore, the response from the joint utility function is modelled as a Bernoulli random variable with the following distribution:

$$\begin{aligned} p(b = 1 \mid [\boldsymbol{\theta}, \boldsymbol{\theta}']) &= \rho([\boldsymbol{\theta}, \boldsymbol{\theta}']) \\ p(b = 0 \mid [\boldsymbol{\theta}, \boldsymbol{\theta}']) &= \rho([\boldsymbol{\theta}, \boldsymbol{\theta}']) \end{aligned} \tag{6.8}$$

where $\rho([\boldsymbol{\theta}, \boldsymbol{\theta}'])$ is an inverse link function and specifies the probability that $\boldsymbol{\theta}$ has a higher preference than $\boldsymbol{\theta}'$. Clearly, this function satisfies $\rho([\boldsymbol{\theta}, \boldsymbol{\theta}']) = 1 - \rho([\boldsymbol{\theta}', \boldsymbol{\theta}])$. Therefore, we could assume it to be a logistic function:

$$\rho([\boldsymbol{\theta}, \boldsymbol{\theta}']) = \frac{1}{1 + e^{-r([\boldsymbol{\theta}, \boldsymbol{\theta}'])}} \tag{6.9}$$

Since the output of $\rho(\,.\,)$ could be interpreted as a degree of confidence in preferring the first member of a duel to the second, it is also referred to as



(a) (b) (c)

Figure 6.6: Level sets of order-preserving approximating distributions that are obtained by setting $\boldsymbol{z}$ to fixed values and moving around in the input noise space, $\boldsymbol{\epsilon}$. Here, per $\boldsymbol{z}$ value, 100 samples are taken from $\boldsymbol{\epsilon}$ on equal intervals in range of $[0, 2\pi]$. Level sets shown in (a), (b) and (c) correspond to the order-preserving models of Figure 6.3-b, Figure 6.4-b and Figure 6.5-b, respectively.

preference function. This method of modeling pairwise preferences allows us to reformulate the problem of order-preservation in an implicit model as a binary classification [89, 96, 54].

6.4.2 Order-Preserving Loss

To enforce order preservation, we assign a binary label to each pair of samples from an implicit model, $[\boldsymbol{\theta}_i, \boldsymbol{\theta}_j],\ i \neq j$, by comparing the probabilities of their corresponding latent code through an indicator function $\mathbb{1}[p(\boldsymbol{z}_i) \geq p(\boldsymbol{z}_j)]$. This label denotes the desired order between $p(\boldsymbol{\theta}_i)$ and $p(\boldsymbol{\theta}_i)$ under the target distribution. We assume that the preference function could evaluate the existence of such an order in a probabilistic manner. Subsequently, the disagreement between the order of the level sets in the latent space and the output of the implicit model could be expressed using binary cross entropy.

Based on this principle, we add a new term to (6.4) to penalize the discrepancy between the order of the level sets in the input, and output of generator function. For a set of M samples $\boldsymbol{\theta}_i = \mathcal{G}_{\boldsymbol{\beta}}(\boldsymbol{z}_i, \boldsymbol{\epsilon}_i), i = 1, \dots, M$ in a batch, a set of $M/2$ pairs $(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j), i \neq j$ are formed and the loss term is computed as follow:

$$\mathcal{L}_o(\mathcal{G}_{\boldsymbol{\beta}}) = -\frac{2}{M}\sum_{1}^{M/2}\mathbb{1}[p(\boldsymbol{z}_i) \geq p(\boldsymbol{z}_j)] \log \rho([\boldsymbol{\theta}_i, \boldsymbol{\theta}_j]) + \\ (1 - \mathbb{1}[p(\boldsymbol{z}_i) \geq p(\boldsymbol{z}_j)]) \log(1 - \rho([\boldsymbol{\theta}_i, \boldsymbol{\theta}_j])) \tag{6.10}$$

in which $\mathbb{1}[.]$ is the indicator function.

Using (6.10) the new generator loss (6.4) becomes:

$$\mathcal{L}(\mathcal{G}; \mathcal{C}) = -\mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta})] + \kappa.\mathcal{L}_o(\mathcal{G}_{\boldsymbol{\beta}}), \quad \kappa \geq 0 \tag{6.11}$$

where κ is a hyperparameter.

Input latent code $\boldsymbol{z}$ could be chosen from any arbitrary distribution and no restriction is put on its dimensionality. A reasonable choice, however, would be a one-dimensional truncated Normal. By doing so, after performing inference in an ideal scenario, each point in the $\boldsymbol{z}$ space is expectedly mapped into a level set in the output space. This suggests that $\boldsymbol{\epsilon}$ is uniformly sampled from a bounded range so that no preference is given to a specific region on a level set. Then, for any two points $\boldsymbol{\theta}$ and $\boldsymbol{\theta}'$:

$$\boldsymbol{\theta} \sim p(\boldsymbol{\theta} \mid \boldsymbol{z}_i) \ \wedge \ \boldsymbol{\theta}' \sim p(\boldsymbol{\theta} \mid \boldsymbol{z}_i) \quad \Longrightarrow \quad q_{\boldsymbol{\beta}}(\boldsymbol{\theta}) = q_{\boldsymbol{\beta}}(\boldsymbol{\theta}')$$

If $\boldsymbol{\epsilon} \sim \mathcal{U}(0, 2\pi.\mathbf{I})$ then the combination of $\boldsymbol{z}$ and $\boldsymbol{\epsilon}$ resembles a polar coordinate in which $\boldsymbol{z}$ represents distance from the origin and $\boldsymbol{\epsilon}$ is the angle. Consequently, generating points on a λ-level set in the $\boldsymbol{\theta}$ space corresponds to sampling from the surface of a hyper-sphere in the generator function's input. In practice, we found that using a $1d$ Normal (rather than a truncated distribution) results in a faster convergence and better recovery of the level sets.

6.5 Implicit Variational Bayesian Neural Networks

A major field of application for implicit models is *implicit variational inference* (IVI) where the target distribution has a tractable density while having an intractable data-generative process, for example when it is the posterior over a set of global (or local) latent variables. IVI uses a two-step adversarial training, similar to the likelihood-free inference [98], however, due to the intractability of the sampling, it is necessary to use KL-divergence (2.23) as the only available criterion from the set of f-divergances [198]. If the target distribution is the posterior probability distribution over the weight space of another neural network, then the idea of deep implicit models coincides with the concept of what are called *HyperNetwork*s.

A *Hypernetwork* [76, 16, 219, 135] is a neural network trained to generate learnable parameters of another neural network, termed the *main network*. Traditionally, Hypernetworks are meant to produce point estimates, that is a single best setup for the parameters of the main-network under ML or MAP regime. Nevertheless, using a generative model (whether implicit [206, 37, 177] or flow-based [119]) as a Hypernetwork upgrades it to a powerful inference schema for Variational Bayesian Neural Networks (VBNN)s whose performance is on a par with BNNs inferred using slow MCMC methods [119, 166]. Next, we explain *Implicit Variational Bayesian Neural Network*s (IVBNN)s, i.e. Bayesian neural networks Inferred using IVI. We utilize them in a BO context for hyperparameter optimization of ML models.

Inference is performed by optimization of the variational objective aided by a GAN-style adversarial training. To this end, given a set of noisy observations $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\}$ from a continuous function (see 2.3) the ELBO (2.27) for the IVBN is rewritten as follows:

$$\mathcal{L}(\boldsymbol{\beta}, \mathcal{D}) = \mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n)] - \mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\log \frac{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}] \tag{6.12}$$

where σ_n denotes the standard deviation of the noise, $\boldsymbol{\theta}$ is the vector of

trainable parameters of the *main network* and $p(\boldsymbol{\theta})$ is an (implicit or explicit) prior defined over these parameters. The first term on the RHS is simply the expected log-likelihood and the second term in (6.12) is the *prior-contrastive* term, that is the KL-distance between the approximating and prior distributions. Intractability of the KL term suggests the use of an adversarial optimization scheme. Hence, a critic is trained with the loss in (6.3) and used as an approximation to the log ratio between the approximating and the prior distribution. Subsequently, the ELBO for the implicit model (6.12) can be maximized by minimization of the following loss over the parameters of the generator function:

$$\mathcal{L}(\mathcal{G};\mathcal{C},\mathcal{D}) = -\mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta})] - \mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n)] \tag{6.13}$$

While this loss, which represents the KL-distance between the approximating and the posterior distributions, could serve for building an IVBNN, the estimation of ratios using adversarial units is prone to instability in high dimensional spaces [198], such as the space of the weights of a neural network. However, using this loss, we could also assign a noisy value to each sample generated by $q_{\boldsymbol{\beta}}(\boldsymbol{\theta})$ which represents the divergence of the approximating distribution from the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{y}, \boldsymbol{X})$ from that sample's point of view. *These* values could be used as utilities to form a preference function:

$$u(\boldsymbol{\theta}) = -\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta}) - \log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n) \tag{6.14}$$

Subsequently, an order-preserving term is added to the generator's loss:

$$\mathcal{L}(\mathcal{G};\mathcal{C},\mathcal{D}) = -\mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\mathcal{C}_{\boldsymbol{\gamma}}(\boldsymbol{\theta})] - \mathbb{E}_{q_{\boldsymbol{\beta}}(\boldsymbol{\theta})}[\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}, \sigma_n)] + \kappa.\mathcal{L}_o(\mathcal{G}_{\boldsymbol{\beta}}), \quad \kappa \geq 0 \tag{6.15}$$

For the IVBNN we have chosen the *hypernetwork* from the class of *deep implicit model*s (DIM) [181, 198, 85]. DIM is a rich class of generative models which entails the advantages of both *probabilistic graphical model*s and *generative adversarial networks*. What makes DIM different from the most commonly used "simple" implicit model is that in the latter the noise

only appears in the input of the generator and therefore the generator function is deterministic. In the DIM version, noise is injected into every second layer of the generative network, creating an alternating sequence of deterministic and stochastic layers. As a result, their training is easier than simple implicit models [198]. The general structure of a DIM is as follows [198]:

$$\boldsymbol{z}^{(L)} = g_L(\boldsymbol{\epsilon}^{(L)} \mid \boldsymbol{\beta}_L) \tag{6.16}$$

$$\boldsymbol{z}^{(L-i)} = g_{L-i}(\boldsymbol{\epsilon}^{(L-i)} \mid \boldsymbol{z}^{(L-i+1)}, \boldsymbol{\beta}_{L-i}), \quad i = 1, \ldots, L-1 \tag{6.17}$$

$$\boldsymbol{\theta} = g_0(\boldsymbol{z}^{(1)} \mid \boldsymbol{\beta}_0), \qquad \boldsymbol{\epsilon}^{(\ell)} \sim p(\boldsymbol{\epsilon}^{(\ell)}) \tag{6.18}$$

where g_L is a nonlinear transformation, $\boldsymbol{\beta}_\ell$ is the vector of its parameters, $\epsilon^{(\ell)}$ and $\boldsymbol{z}^{(\ell)}$ denote noise and latent variable corresponding to the ℓ^{th} transformation, respectively. While g_L and g_0 are deterministic transformations, g_{L-i} denotes a deterministic transformation of $\boldsymbol{z}^{(L-i+1)}$ followed by a stochastic one in which $\boldsymbol{\epsilon}^{(L-i)}$ is used as the source of variation.

Since in our case an order-preserving loss is to be used, the input to the first layer of the DIM is divided into a noise part and a latent code. Consequently, (6.16) is replaced by the following transformation:

$$\boldsymbol{z}^{(L)} = g_L(\boldsymbol{\epsilon}^{(L)}, \boldsymbol{z}^{(o)} \mid \boldsymbol{\beta}_L) \tag{6.19}$$

in which $\boldsymbol{z}^{(o)}$ denotes a latent code.

Using a set of latent variables $\boldsymbol{z}^1, \ldots, \boldsymbol{z}^L$, DIM hyper-networks are able to encompass interactions between parameters in the highly correlated weight space of neural networks. However, as it can be seen in (6.18), regardless of the placement of a weight in the main network, values of the weights in the weight vectors generated by DIM are conditionally dependent on these latent variables, and only marginally independent. Contrast this with the case of a regular (feed forward) neural network, in which a direct conditional dependence exists between the weights of successive layers. Hence, a more natural way to represent these conditional dependencies is to include them into the architecture of the hyper-network,

essentially as prior knowledge. Accordingly, rather than using a single output layer, we modify the DIM to have a *sequence of output layers*, each of which is responsible for generating the weights of a single layer in the main network, as follows:

$$\boldsymbol{\theta}^{(i)} = \begin{cases} g_{0,i}(\boldsymbol{z}^{(i)} \mid \boldsymbol{\beta}_{0,i}), & i = 1 \\ g_{0,i}(\boldsymbol{\theta}^{(i-1)} \mid \boldsymbol{\beta}_{0,i}), & i > 1, \end{cases} \tag{6.20}$$

where $\boldsymbol{\theta}^{(i)}$ is the weight vector of the i^{th} layer of the main network, and $g_{0,i}$ denotes the i^{th} output layer parameterized by $\boldsymbol{\beta}_{0,i}$. Subsequently, using the chain rule, the approximating distribution can be written as:

$$q_{\boldsymbol{\beta}}(\boldsymbol{\theta}) = q(\boldsymbol{\theta}^{(1)} \mid \boldsymbol{z}^{(1)}) \prod_{i=1}^{N_l} q(\boldsymbol{\theta}^{(i)} \mid \boldsymbol{\theta}^{(i-1)}), \qquad \boldsymbol{\theta} = \cup_{i=1}^{N_l} \{\boldsymbol{\theta}^{(i)}\} \tag{6.21}$$

in which N_l is the number of the layers of the *main network*.

Figure 6.7 shows a schematic of the complete architecture: an *o*rder-preserving IVBNN with an deep implicit hypernetwork.

6.6 Bayesian Optimization using IVBNN

Despite theoretical advantages of Bayesian neural networks over GPs, the slow convergence rate of MCMC methods has limited their usage in BO. Methods that deal with this issue usually make some simplifying assumptions about the characteristics of the model. As an almost inevitable consequence of these simplifications, recovery of the true posterior becomes out of reach. Although in many applications, some deviation from the true posterior is tolerable, the whole machinery of BO depends on the preciseness of the inferred posterior. Using IVBNN,there is no need to compromise the quality of the posterior for the speed. Here, we use IVBNN as a surrogate model in BO.

Figure 6.8 gives an illustrative example of an IVBNN generating a predictive distribution given 4 data points in 1 dimension.

To the best of our knowledge, all previous NN-based BOs maintain a model with a fixed number of parameters during the optimization. The structure of the model is selected (in advance) in the hope that it holds enough capacity to derive the optimization for some finite number of steps. For example, suppose that at the onset of the optimization just two initial observations are available. In this case, a Bayesian linear model has sufficient representational capacity to accommodate the whole information provided by these points, whereas a higher capacity model might represent a misleading perspective. However, after a few optimization steps, new observations are added to the available data and their modeling might be beyond the capacity of a linear model.

Following this line of thinking, we argue that maintaining a model with a fixed number of parameters is absolutely unnecessary in BO, and suggest instead that the model capacity be adjusted in proportion to the amount of information available about the underlying function.

6.7 Experimental Setup

The IVBNN used for BO has a single-hidden layer *main network* with *tanh* activation functions. To make a more efficient use of computational resources, unlike the previous chapters, here we do not utilize a model whose structure is fixed during the whole optimization. Instead, we gradually increase the complexity of the surrogate model according to the number of observations. Therefore, at each optimization step a width of twice the number of observations is considers for the hidden layer. Data appearing in the input as well as output of the *main network* is standardized. The *Hypernetwork* is a three-layer DIM whose widths are equal to the number of hidden units in the *main network*; the input latent variable to the implicit model $\boldsymbol{z}^{(o)}$ is a five-dimensional standard Normal and a unidimensional Gaussian noise is injected to the input as well as every other layer of the DIM. This architecture was set based on a set of initial trials on a toy prob-

lem. The *Hypernetwork* weights are initialized using a variance scaling initializer with the standard deviation equal to 2.0. The critic also has three hidden layers whose widths are equal to the number of the parameters in the *main network*. The prior defined over the parameters of the main network is a mixture of two zero-centered isotropic Gaussians: the first component has a mixing probability of 0.2 and standard deviation of 1.0, whereas the second one has a standard deviation equal to 5.0. In other words, although we prefer the weights to have small values (generated from standard Gaussians), we still do not reject the possibility of larger values. The likelihood noise σ_n starts at 1.0 and is annealed over the course of training to a lower bound of 0.002. We found this method to be more effective than fixing the noise standard deviation to a fixed value. During training, samples are generated in batches of 100. The BNN used in BO is formed from 30 individual NNs. $\kappa = 2.0$, the learning rate for the generator, and critic is set to $3e-5$ and in each full training step of the implicit model, the critic is trained for 40 iterations.

Experiments are performed using the Surrogate Benchmark for hyperparameter optimization [51] which is a part of HPOlib Hyperparameter Optimization Benchmark [49]. As before, for each algorithm 30 independent runs are done on each benchmark problem where each run has a budget of 50 evaluations, the default maximum budget of HPO problems in HPOlib. A run begins with an initial design of size two. Since we have limited the number of initial points to such a small number, the results also reflect the sensitivity the models to the location of initial points.

6.8 Results and Discussion

The main comparison we wish to make is between the IVBNN and BOHAMIANN, since they use similar surrogate models (a Bayesian neural network) but a different inference method (stochastic gradient Hamiltonian Monte Carlo [192]). Figure 6.9 shows this for the 9 hyperparameter

optimization problems in `HPOlib` (other algorithms are omitted from 6.9 to avoid clutter but are shown in later tables).

As the figure shows, performance of our implicit variational surrogate is considerably better than Monte Carlo running on the same underlying architecture in all cases except one. In no case does BOHAMIANN dominate IVBNN in the early stages. In some cases in [`CNN(CIFAR10)`, `Paramnet(MNIST)` and `Paramnet(Letter)`] performance is about the same early on, but after 10-20 samples IVBNN comes to the fore. In almost all cases IVBNN is exploring better points in the search space over the majority of the trajectory.

The sole exception is `SVM(MNIST)`, in which IVBNN never moves far from its initial condition: with this setup, the implicit model's NN is never generating samples that are suitably dense in the high probability regions of the posterior (whereas the MCMC approach is).

Summary results for IVBNN against a variety of opposition are shown in Tables 6.2, 6.3 and 6.4, for the position after 50 samples. At that point, the main competition is (predictably) GP_{mcmc}. Most importantly, and as is seen in Figure 6.9, the new method is better than BOHAMIANN in all cases except one. One might reasonably exclude `Paramnet (Adult)`, `Paramnet (Higgs)` and `Wide Resnet (2d)`, as results are very close and even the random search algorithm is competitive after 50 samples. Of the remainder, IVBNN is always among the top 3 (usually second behind GP_{mcmc}). While these results do not indicate that one should abandon (say) GP_{mcmc} for the new algorithm, they do show that a completely orthogonal approach to the use of neural nets in BO is feasible and can reliably outperform the existing one.

6.9 Conclusion

We introduced *order-preserving* implicit models which are useful in scenarios where in addition to the generated samples, there is more information

Table 6.2: Validation errors (averaged over 50 trials) induced by using six different BO methods to find the best set of hyperparameter configurations for CNN and Paramnet models. CNN is trained on CIFAR10 and the Paramnet is trained on Adult and Higgs datasets. In each column, the best result is shown in bold and the first and second runner-ups are underlined.

Benchmark Method	CNN (CIFAR10)	Paramnet (Adult)	Paramnet (Higgs)
IVBNN	$\underline{1.66 \times 10^{-1} \pm 1.38 \times 10^{-2}}$	$\mathbf{1.49 \times 10^{-1} \pm 9.74 \times 10^{-4}}$	$\underline{2.85 \times 10^{-1} \pm 2.98 \times 10^{-3}}$
Bohamiann	$1.75 \times 10^{-1} \pm 1.40 \times 10^{-2}$	$1.49 \times 10^{-1} \pm 9.79 \times 10^{-4}$	$2.86 \times 10^{-1} \pm 2.69 \times 10^{-3}$
GP	$\underline{1.65 \times 10^{-1} \pm 5.82 \times 10^{-3}}$	$1.49 \times 10^{-1} \pm 5.52 \times 10^{-4}$	$2.87 \times 10^{-1} \pm 3.59 \times 10^{-3}$
GP_{mcmc}	$\mathbf{1.63 \times 10^{-1} \pm 5.53 \times 10^{-3}}$	$\underline{1.49 \times 10^{-1} \pm 9.22 \times 10^{-4}}$	$\underline{2.85 \times 10^{-1} \pm 2.73 \times 10^{-3}}$
RandomForest	$1.70 \times 10^{-1} \pm 1.66 \times 10^{-2}$	$\underline{1.49 \times 10^{-1} \pm 9.69 \times 10^{-4}}$	$\mathbf{2.84 \times 10^{-1} \pm 3.32 \times 10^{-3}}$
RandomSearch	$1.68 \times 10^{-1} \pm 5.20 \times 10^{-3}$	$1.49 \times 10^{-1} \pm 9.93 \times 10^{-4}$	$2.85 \times 10^{-1} \pm 1.67 \times 10^{-3}$

about the target density that can only be obtained in the form of inaccurate pairwise preferences. In such a case, the proposed model can utilize this information to impose the order over the density level sets from the distribution defined over its input latent space to the approximating distribution. A further advantage of the order-preserving models over the simple implicit models is that in the former mode collapse is prevented to a high degree. We used these new models as hypernetworks in a variational Bayesian inference scheme to infer the posterior over the weights of a BNN. We applied this implicit variational BNN, namely IVBNN, as the surrogate in a BO scenario. Preliminary evaluations over a set of benchmark hyperparameter optimization problems show dominance of IVBNN over BNNs inferred using stochastic Hamiltonin Monte Carlo in BO.

Table 6.3: Validation errors (averaged over 50 trials) induced by using six different BO methods to find the best set of hyperparameter configurations for Paramnet model trained over Letter, Mnist and Optical Digits datasets. In each column, the best result is shown in bold and the first and second runner-ups are underlined.

Benchmark Method	Paramnet (Letter)	Paramnet (Mnist)	Paramnet (Optical Digits)
IVBNN	$\mathbf{4.44 \times 10^{-2} \pm 1.12 \times 10^{-2}}$	$\underline{1.75 \times 10^{-2} \pm 3.43 \times 10^{-3}}$	$\underline{1.83 \times 10^{-2} \pm 1.98 \times 10^{-3}}$
Bohamiann	$5.51 \times 10^{-2} \pm 1.43 \times 10^{-2}$	$1.83 \times 10^{-2} \pm 1.67 \times 10^{-3}$	$1.98 \times 10^{-2} \pm 1.52 \times 10^{-3}$
GP	$\underline{4.79 \times 10^{-2} \pm 6.12 \times 10^{-3}}$	$\underline{1.76 \times 10^{-2} \pm 1.41 \times 10^{-3}}$	$\mathbf{1.82 \times 10^{-2} \pm 1.40 \times 10^{-3}}$
GP_{mcmc}	$\underline{4.55 \times 10^{-2} \pm 7.18 \times 10^{-3}}$	$\mathbf{1.71 \times 10^{-2} \pm 7.78 \times 10^{-4}}$	$\underline{1.85 \times 10^{-2} \pm 1.35 \times 10^{-3}}$
RandomForest	$4.89 \times 10^{-2} \pm 1.26 \times 10^{-2}$	$1.80 \times 10^{-2} \pm 3.80 \times 10^{-3}$	$1.93 \times 10^{-2} \pm 6.35 \times 10^{-3}$
RandomSearch	$6.16 \times 10^{-2} \pm 1.79 \times 10^{-2}$	$1.77 \times 10^{-2} \pm 1.21 \times 10^{-3}$	$2.01 \times 10^{-2} \pm 2.12 \times 10^{-3}$

Table 6.4: Validation errors (averaged over 50 trials) induced by using six different BO methods to find the best set of hyperparameter configurations for Paramnet model (over Poker dataset), SVM (over Mnist) and Wide Resnet. In each column, the best result is shown in bold and the first and second runner-ups are underlined.

Benchmark Method	Paramnet (Poker)	SVM (Mnist)	Wide Resnet (2d)
IVBNN	$\underline{1.37 \times 10^{-2} \pm 4.83 \times 10^{-2}}$	$1.32 \times 10^{-1} \pm 3.04 \times 10^{-1}$	$5.24 \times 10^{-2} \pm 7.66 \times 10^{-6}$
Bohamiann	$2.00 \times 10^{-2} \pm 2.76 \times 10^{-2}$	$2.13 \times 10^{-2} \pm 8.91 \times 10^{-3}$	$5.39 \times 10^{-2} \pm 1.14 \times 10^{-3}$
GP	$2.06 \times 10^{-2} \pm 2.24 \times 10^{-2}$	$\underline{1.52 \times 10^{-2} \pm 6.20 \times 10^{-4}}$	$\mathbf{5.24 \times 10^{-2} \pm 0.00}$
GP_{mcmc}	$\mathbf{8.50 \times 10^{-3} \pm 1.16 \times 10^{-2}}$	$\mathbf{1.51 \times 10^{-2} \pm 4.47 \times 10^{-4}}$	$5.24 \times 10^{-2} \pm 6.91 \times 10^{-7}$
RandomForest	$\underline{1.67 \times 10^{-2} \pm 4.51 \times 10^{-2}}$	$\underline{1.56 \times 10^{-2} \pm 1.57 \times 10^{-3}}$	$\underline{5.24 \times 10^{-2} \pm 0.00}$
RandomSearch	$1.73 \times 10^{-2} \pm 2.48 \times 10^{-2}$	$1.63 \times 10^{-2} \pm 1.78 \times 10^{-3}$	$\underline{5.24 \times 10^{-2} \pm 5.07 \times 10^{-7}}$

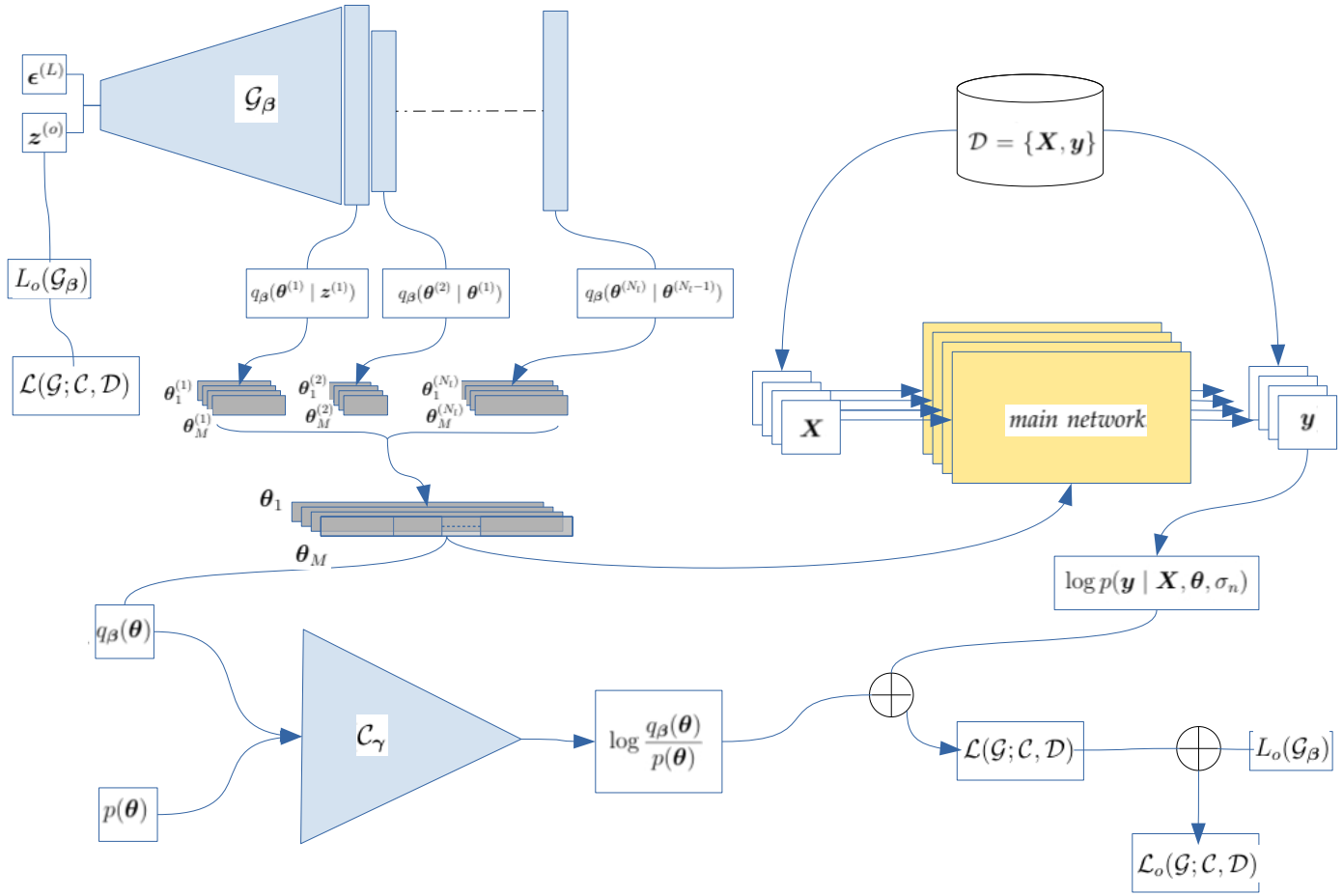


Figure 6.7: Schematic of an *order-preserving* IVBNN with an L-layer *deep implicit hypernetwork*. The generator's input is divided to noise part, ϵ^L (which contributes to the stochasticity) and a latent code z^o which is used along with the per-sample log-ratio to form the order-preserving loss term. The *main network* has N_l layers of varying sizes. The posterior over weights of each layer is conditionally dependent on the previous layer. This conditional dependence is reflected in the structure of the hypernetwork. Each batch of samples (main networks) generated by the hypernetwork is evaluated against the prior (using the critic) and the log likelihood (using available noisy observations) to form the negative ELBO $\mathcal{L}(\mathcal{G};\mathcal{C},\mathcal{D})$. To get the final loss, the order-preserving loss term is added to the equation.

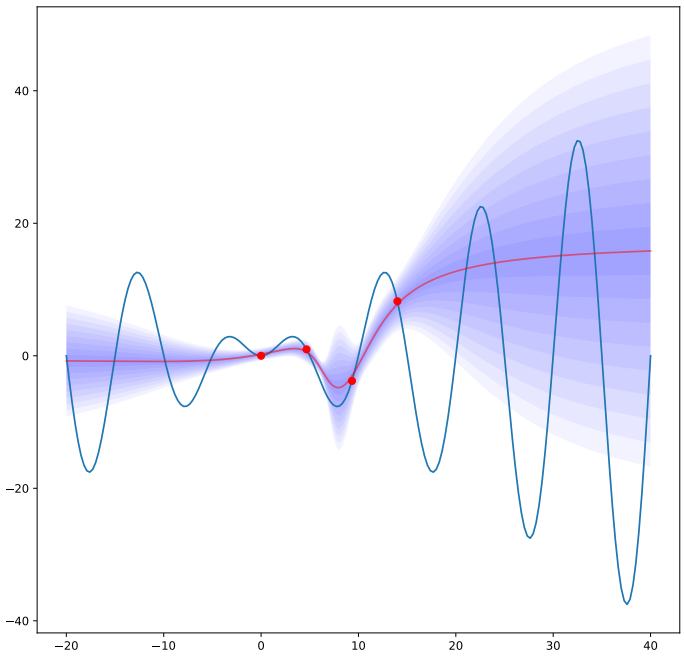


Figure 6.8: IVBNN trained using 4 observations (from a 1-dimensional sinusoid, shown in blue). Note the well-behaved "error bars" as the predictor moves away from data points. In this case, those errors grow more rapidly on the right than the left as a consequence of the slope between two leftmost points being gentler than the one between the two on the right. (Note this would not happen with a vanilla Gaussian Process).

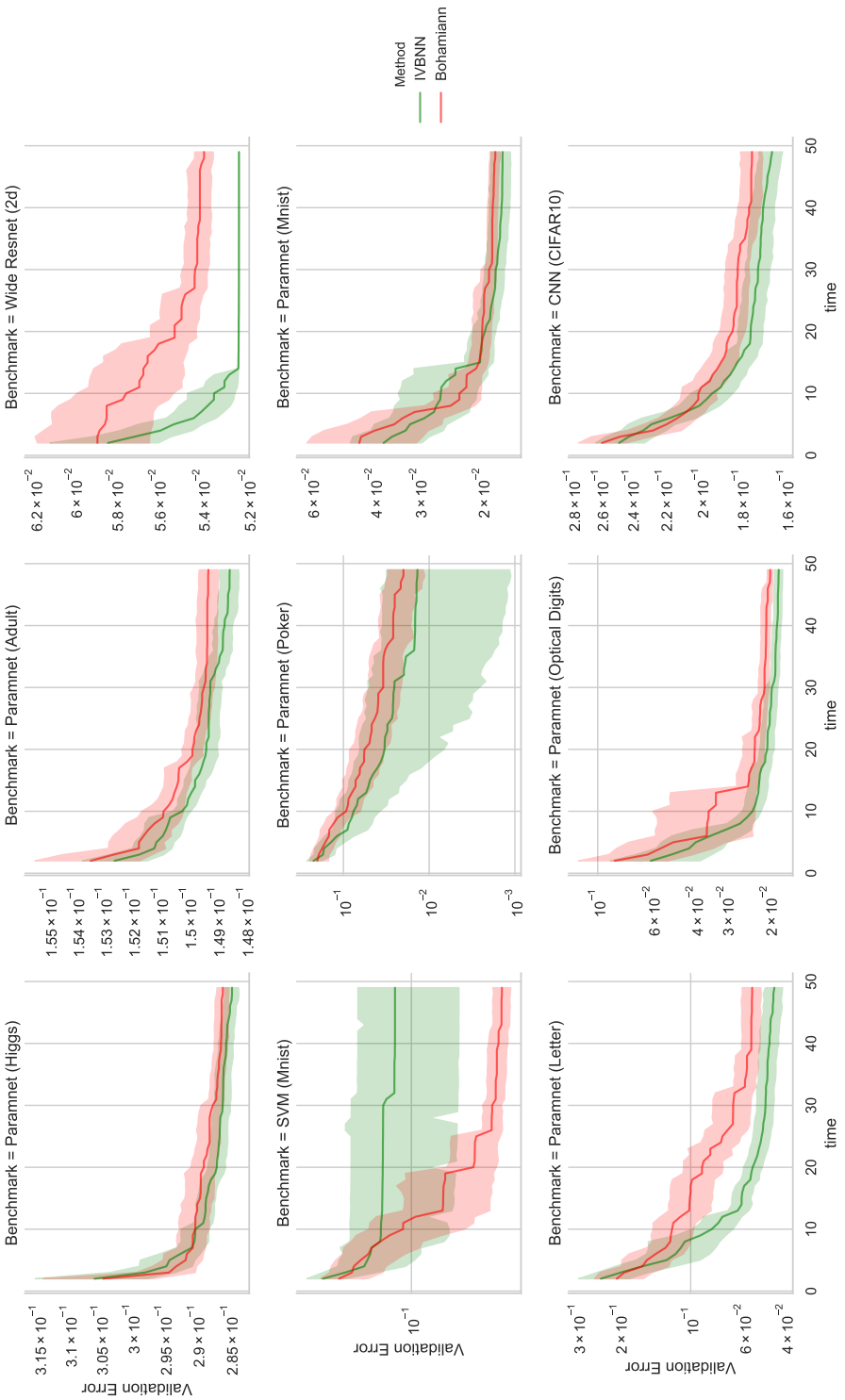


Figure 6.9: Timecourse comparison of IVBNN with BOHAMIANN, for 9 problems.

Chapter 7

Conclusion

This thesis introduced novel methods in which Neural Networks are used to enhance Bayesian Optimization. In this respect, several such algorithms were developed, tested and compared to the state-of-the-art on benchmark problems. Bayesian Optimization algorithms essentially consist of two elements: a surrogate model capable of representing uncertainty, and an acquisition function that uses the surrogate to suggest new sample points. The work covered here has introduced improved alternatives for both elements. The proposed methods were evaluated against currently available Bayesian optimization algorithms on sets of benchmark problems, including standard numerical tests and genuinely expensive hyperparameter optimization tasks.

We now revisit the contributions outlined in Chapter One and summarize each contribution.

7.1 Accomplished Objectives

The following objectives were accomplished in this thesis:

- We developed a new Bayesian Optimization algorithm in which *Bayesian Neural Network*s were used as a replacement for Gaussian

Process models. The computational complexity of the model building process in the new BO is lower than that classical BO's. Both simple and fully Bayesian Neural Networks were evaluated against the Gaussian Process using the commonly used squared exponential covariance function. The Bayesian Neural Network models were inferred using an adaptive version of Hamiltonian Monte Carlo in which the need for setting the number of leap frog steps is eliminated. Extensive benchmarks of different variations of the proposed method against Gaussian Process based Bayesian Optimization showed that they could perform on par with or better than classical BO in the optimization of a variety of continuous functions with different levels of difficulty as well as dimensionality. We also developed a new BO where an ensemble of random Neural Networks, namely Extreme Learning Machines, was used as surrogate. This BO method, however, demonstrated a poor performance compared to the other available BO methods. Last but not least, we introduced a new *empirical* Expected Improvement infill criteria that can be used with any ensemble-based surrogate model and in cases where sampling is computationally cheaper than computing the posterior.

- We introduced a fast and simple Bayesian Optimization algorithm using Bayesian Random Vector Functional-link Networks. The motivation behind the use of these models was their small number of trainable parameters and fast inference process compared with Bayesian Neural Networks. We also evaluated an ensemble of MAP RVFLs. Moreover, a simple and efficient random initialization method was proposed that interoperates well with *Relu* RVFLs. We performed extensive evaluation of the proposed methods using various configurations on both a set of benchmark synthetic functions as well as machine learning hyperparameter optimization problems. This showed that Bayesian Optimization using Bayesian RVFLs surrogates equipped with evidence approximation for hyperparameter

tuning performs on par with many state of art Bayesian Optimization algorithms, while having the advantage of fast inference.

- We introduced a new infill criterion based on the notion of *maximum a posteriori* estimation. Motivated by the weakness of the double stage optimization approach in which infill points are selected based on a sequence of error-prone assumptions, the new model-based infill criteria uses a Neural Network model to correct possible inaccuracies of the expected improvement acquisition function. To this end, the proposed infill criterion utilizes a "single-stage optimization" based on a conditional maximum likelihood method. We evaluated a combination of different surrogate models and the proposed infill criteria and compared it against expected improvement on a number of machine learning hyperparameter optimization problems. The evaluations showed advantages of using this acquisition function with Neural network-based surrogates over Expected improvement infill criterion.

- We developed order-preserving implicit models as an alternative to simple implicit models, such as Generative Adversarial Networks (GAN)s, where along with the tractable sampling process some (noisy) information about the density of the target distribution can be obtained in the form of inaccurate binary preferences. As opposed to other GAN-style generative models, the new model can use this preference information to impose an order over the density level sets from its input latent distribution to the approximating distribution. Moreover, in the presence of the "preference" information, the order-preserving model can prevent mode-collapse to a high degree.

  We also introduced an approach to Bayesian optimization using *implicit variational Bayesian neural networks* (IVBNN). In this approach a rich class of implicit distributions are used as *Hypernetwork* to approximate the posterior over the weights of a Bayesian Neural Networks, which is then utilized as a surrogate in the BO context. The new

model enjoys the advantages of asymptotically exact Monte Carlo-based sampling methods as well as fast optimization-based variational inference techniques. Preliminary evaluations on a set of hyperparameter optimization benchmark problems show the dominance of IVBNN over Bayesian Neural Networks inferred by Hamiltonian Monte Carlo.

In conclusion, and as a high-level summary, this thesis has explored the application of neural networks to the Bayesian Optimization problem. Several new algorithms were introduced, with promising results. We believe this thesis can provide both a useful guide for those who wish to apply neural networks to difficult black-box optimization problems, as well as a starting point for future research in this important area.

Bibliography

[1] ABDEL-HAMID, O., MOHAMED, A.-R., JIANG, H., DENG, L., PENN, G., AND YU, D. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing 22*, 10 (2014), 1533–1545.

[2] AILON, N., KARNIN, Z., AND JOACHIMS, T. Reducing dueling bandits to cardinal bandits. In *International Conference on Machine Learning* (2014), pp. 856–864.

[3] AKRITIDIS, L., KATSAROS, D., AND BOZANIS, P. Effective rank aggregation for metasearching. *Journal of Systems and Software 84*, 1 (2011), 130–143.

[4] ASHMAIG, O., CONNOLLY, M., GROSS, R. E., AND MAHMOUDI, B. Bayesian optimization of asynchronous distributed microelectrode theta stimulation and spatial memory. In *2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)* (2018), IEEE, pp. 2683–2686.

[5] BAKER, J. *Large-scale Bayesian computation using Stochastic Gradient Markov Chain Monte Carlo*. PhD thesis, Lancaster University, 2019.

[6] BARDSLEY, J. M., SOLONEN, A., HAARIO, H., AND LAINE, M. Randomize-then-optimize: A method for sampling from posterior distributions in nonlinear inverse problems. *SIAM Journal on Scientific Computing 36*, 4 (2014), A1895–A1910.

[7] BARTON, R. R. Metamodeling: a state of the art review. In *Proceedings of the 26th conference on Winter simulation* (1994), Society for Computer Simulation International, pp. 237–244.

[8] BERG, R. V. D., HASENCLEVER, L., TOMCZAK, J. M., AND WELLING, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649* (2018).

[9] BETANCOURT, M. The fundamental incompatibility of scalable hamiltonian monte carlo and naive data subsampling. In *International Conference on Machine Learning* (2015), pp. 533–540.

[10] BETANCOURT, M. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434* (2017).

[11] BISHOP, C. M. *Pattern recognition and machine learning*. springer, 2006.

[12] BLEI, D. M., KUCUKELBIR, A., AND MCAULIFFE, J. D. Variational inference: A review for statisticians. *Journal of the American Statistical Association 112*, 518 (2017), 859–877.

[13] BLUNDELL, C., CORNEBISE, J., KAVUKCUOGLU, K., AND WIERSTRA, D. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424* (2015).

[14] BREHMER, J., LOUPPE, G., PAVEZ, J., AND CRANMER, K. Mining gold from implicit models to improve likelihood-free inference. *arXiv preprint arXiv:1805.12244* (2018).

[15] BROCHU, E., CORA, V. M., AND DE FREITAS, N. 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).

[16] BROCK, A., LIM, T., RITCHIE, J. M., AND WESTON, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).

[17] BUHMANN, M. A new class of radial basis functions with compact support. *Mathematics of Computation 70*, 233 (2001), 307–318.

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

[19] BYRD, R. H., LU, P., NOCEDAL, J., AND ZHU, C. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing 16*, 5 (1995), 1190–1208.

[20] CALANDRA, R., GOPALAN, N., SEYFARTH, A., PETERS, J., AND DEISENROTH, M. P. Bayesian gait optimization for bipedal locomotion. In *Proceedings of Learning and Intelligent OptimizatioN Conference (LION8)* (2014).

[21] CARPENTER, B., GELMAN, A., HOFFMAN, M., LEE, D., GOODRICH, B., BETANCOURT, M., BRUBAKER, M. A., GUO, J., LI, P., AND RIDDELL, A. Stan: A probabilistic programming language. *J Stat Softw* (2016).

[22] CECOTTI, H. Deep random vector functional link network for handwritten character recognition. In *2016 International Joint Conference on Neural Networks (IJCNN)* (2016), IEEE, pp. 3628–3633.

[23] CHAU, H. N., AND RASONYI, M. Stochastic gradient hamiltonian monte carlo for non-convex learning in the big data regime. *arXiv preprint arXiv:1903.10328* (2019).

[24] CHEN, H., AND YAO, X. Regularized negative correlation learning for neural network ensembles. *IEEE Transactions on Neural Networks 20*, 12 (2009), 1962–1979.

[25] CHEN, T., FOX, E., AND GUESTRIN, C. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning* (2014), pp. 1683–1691.

[26] CHEN, X., DUAN, Y., HOUTHOOFT, R., SCHULMAN, J., SUTSKEVER, I., AND ABBEEL, P. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems* (2016), pp. 2172–2180.

[27] CHEN, Y.-C., GENOVESE, C. R., AND WASSERMAN, L. Density level sets: Asymptotics, inference, and visualization. *Journal of the American Statistical Association 112*, 520 (2017), 1684–1696.

[28] CHUNG, J., GULCEHRE, C., CHO, K., AND BENGIO, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[29] CORDUNEANU, A., AND BISHOP, C. M. Variational bayesian model selection for mixture distributions. In *Artificial intelligence and Statistics* (2001), vol. 2001, Morgan Kaufmann Waltham, MA, pp. 27–34.

[30] CRESSIE, N. Statistics for spatial data. *Terra Nova 4*, 5 (1992), 613–617.

[31] CRIMINISI, A., SHOTTON, J., KONUKOGLU, E., ET AL. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends® in Computer Graphics and Vision 7*, 2–3 (2012), 81–227.

[32] CSÁJI, B. C. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary 24* (2001), 48.

[33] CUTAJAR, K., BONILLA, E. V., MICHIARDI, P., AND FILIPPONE, M. Random feature expansions for deep gaussian processes. In

Proceedings of the 34th International Conference on Machine Learning-Volume 70 (2017), JMLR. org, pp. 884–893.

[34] DE CAO, N., TITOV, I., AND AZIZ, W. Block neural autoregressive flow. *arXiv preprint arXiv:1904.04676* (2019).

[35] DEISENROTH, M. P., AND NG, J. W. Distributed gaussian processes. *arXiv preprint arXiv:1502.02843* (2015).

[36] DESARKAR, M. S., SARKAR, S., AND MITRA, P. Preference relations based unsupervised rank aggregation for metasearch. *Expert Systems with Applications 49* (2016), 86–98.

[37] DEUTSCH, L. Generating neural networks with neural networks. *arXiv preprint arXiv:1801.01952* (2018).

[38] DEWANCKER, I., MCCOURT, M., CLARK, S., HAYES, P., JOHNSON, A., AND KE, G. A Stratified Analysis of Bayesian Optimization Methods. *CoRR abs/1603.09441* (2016).

[39] DIEHL, F., AND JAUCH, A. apsis - Framework for Automated Optimization of Machine Learning Hyper Parameters. *CoRR abs/1503.02946* (2015).

[40] DIGGLE, P. J., AND GRATTON, R. J. Monte carlo methods of inference for implicit statistical models. *Journal of the Royal Statistical Society: Series B (Methodological) 46*, 2 (1984), 193–212.

[41] DINH, L., SOHL-DICKSTEIN, J., AND BENGIO, S. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803* (2016).

[42] DRINEAS, P., AND MAHONEY, M. W. On the nyström method for approximating a gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research 6* (2005), 2153–2175.

[43] DUANE, S., KENNEDY, A. D., PENDLETON, B. J., AND ROWETH, D. Hybrid monte carlo. *Physics letters B 195*, 2 (1987), 216–222.

[44] DUDEK, G. Extreme learning machine for function approximation-interval problem of input weights and biases. In *2015 IEEE 2nd International Conference on Cybernetics (CYBCONF)* (2015), IEEE, pp. 62–67.

[45] DUDEK, G. Extreme learning machine as a function approximator: initialization of input weights and biases. In *Proceedings of the 9th International Conference on Computer Recognition Systems CORES 2015* (2016), Springer, pp. 59–69.

[46] DUDEK, G. A method of generating random weights and biases in feedforward neural networks with random hidden nodes. *arXiv preprint arXiv:1710.04874* (2017).

[47] DUFOUR, R., DE MUELENAERE, J., AND ELHAM, A. Trajectory driven multidisciplinary design optimization of a sub-orbital spaceplane using non-stationary gaussian process. *Structural and Multidisciplinary Optimization* (2015), 1–17.

[48] DURAND, T., MEHRASA, N., AND MORI, G. Learning a deep convnet for multi-label classification with partial labels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 647–657.

[49] EGGENSPERGER, K., FEURER, M., HUTTER, F., BERGSTRA, J., SNOEK, J., HOOS, H., AND LEYTON-BROWN, K. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice* (2013), pp. 1–5.

[50] EGGENSPERGER, K., FEURER, M., HUTTER, F., BERGSTRA, J., SNOEK, J., HOOS, H., AND LEYTON-BROWN, K. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In

NeurIPS workshop on Bayesian Optimization in Theory and Practice (Dec. 2013).

[51] EGGENSPERGER, K., HUTTER, F., HOOS, H. H., AND LEYTON-BROWN, K. Surrogate benchmarks for hyperparameter optimization. In *MetaSel@ ECAI* (2014), pp. 24–31.

[52] EGGENSPERGER, K., HUTTER, F., HOOS, H. H., AND LEYTON-BROWN, K. Surrogate benchmarks for hyperparameter optimization. In *MetaSel@ECAI* (2014), J. Vanschoren, P. Brazdil, C. Soares, and L. Kotthoff, Eds., vol. 1201 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 24–31.

[53] EVANS, T. W., AND NAIR, P. B. Scalable gaussian processes with grid-structured eigenfunctions (gp-grief). *arXiv preprint arXiv:1807.02125* (2018).

[54] FAHANDAR, M. A., AND HÜLLERMEIER, E. Analogy-based preference learning with kernels. *arXiv preprint arXiv:1901.02001* (2019).

[55] FALKNER, S., KLEIN, A., AND HUTTER, F. Practical hyperparameter optimization for deep learning.

[56] FANG, H., AND HORSTEMEYER, M. F. Global response approximation with radial basis functions. *Engineering Optimization 38*, 04 (2006), 407–424.

[57] FINCK, S., HANSEN, N., ROS, R., AND AUGER, A. Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Tech. rep., Citeseer, 2010.

[58] FORRESTER, A., SOBESTER, A., AND KEANE, A. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[59] FORRESTER, A. I., AND KEANE, A. J. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences 45*, 1 (2009), 50–79.

[60] FOX, C. W., AND ROBERTS, S. J. A tutorial on variational bayesian inference. *Artificial intelligence review 38*, 2 (2012), 85–95.

[61] FREAN, M., AND BOYLE, P. Using Gaussian processes to optimize expensive functions. In *AI 2008: Advances in Artificial Intelligence*. Springer, 2008, pp. 258–267.

[62] FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association 32*, 200 (1937), 675–701.

[63] GAL, Y., AND GHAHRAMANI, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016* (2016), pp. 1050–1059.

[64] GARDNER, J. R., PLEISS, G., WU, R., WEINBERGER, K. Q., AND WILSON, A. G. Product kernel interpolation for scalable gaussian processes. *arXiv preprint arXiv:1802.08903* (2018).

[65] GERSHMAN, S., HOFFMAN, M., AND BLEI, D. Nonparametric variational inference. *arXiv preprint arXiv:1206.4665* (2012).

[66] GEURTS, P., ERNST, D., AND WEHENKEL, L. Extremely randomized trees. *Machine learning 63*, 1 (2006), 3–42.

[67] GIROLAMI, M., AND CALDERHEAD, B. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 73*, 2 (2011), 123–214.

[68] GIROLAMI, M., CALDERHEAD, B., AND CHIN, S. A. Riemannian manifold hamiltonian monte carlo. *arXiv preprint arXiv:0907.1100* (2009).

[69] GIRSHICK, R. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision* (2015), pp. 1440–1448.

[70] GONZÁLEZ, J., DAI, Z., DAMIANOU, A., AND LAWRENCE, N. D. Preferential bayesian optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1282–1291.

[71] GOODFELLOW, I., POUGET-ABADIE, J., MIRZA, M., XU, B., WARDE-FARLEY, D., OZAIR, S., COURVILLE, A., AND BENGIO, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[72] GORBAN, A. N., TYUKIN, I. Y., PROKHOROV, D. V., AND SOFEIKOV, K. I. Approximation with random bases: Pro et contra. *Information Sciences 364* (2016), 129–145.

[73] GRAMACY, R. B., AND LEE, H. K. Bayesian treed gaussian process models with an application to computer modeling. *Journal of the American Statistical Association 103*, 483 (2008).

[74] GRAVES, A., AND JAITLY, N. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning* (2014), pp. 1764–1772.

[75] GRAVES, A., MOHAMED, A.-R., AND HINTON, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (2013), IEEE, pp. 6645–6649.

[76] HA, D., DAI, A., AND LE, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106* (2016).

[77] HANSEN, N., AUGER, A., MERSMANN, O., TUSAR, T., AND BROCKHOFF, D. COCO: A Platform for Comparing Continuous Optimizers in a Black-Box Setting. *CoRR abs/1603.08785* (2016).

[78] HE, K., GKIOXARI, G., DOLLÁR, P., AND GIRSHICK, R. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision* (2017), pp. 2961–2969.

[79] HE, K., ZHANG, X., REN, S., AND SUN, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778.

[80] HENNIG, P., AND SCHULER, C. J. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research 13*, Jun (2012), 1809–1837.

[81] HENRIQUEZ, P. A., AND RUZ, G. A. An empirical study of the hidden matrix rank for neural networks with random weights. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2017), IEEE, pp. 883–888.

[82] HENSMAN, J., FUSI, N., AND LAWRENCE, N. D. Gaussian processes for big data. *arXiv preprint arXiv:1309.6835* (2013).

[83] HERNÁNDEZ-LOBATO, J. M., HOFFMAN, M. W., AND GHAHRAMANI, Z. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems* (2014), pp. 918–926.

[84] HO, J., CHEN, X., SRINIVAS, A., DUAN, Y., AND ABBEEL, P. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv preprint arXiv:1902.00275* (2019).

[85] HOFFMAN, M. D. Learning deep latent gaussian models with markov chain monte carlo. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 1510–1519.

[86] HOMAN, M. D., AND GELMAN, A. The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *The Journal of Machine Learning Research 15*, 1 (2014), 1593–1623.

[87] HONG, X., GAO, J., JIANG, X., AND HARRIS, C. J. Fast identification algorithms for gaussian process model. *Neurocomputing 133* (2014), 25–31.

[88] HORNIK, K. Approximation capabilities of multilayer feedforward networks. *Neural networks 4*, 2 (1991), 251–257.

[89] HOULSBY, N., HUSZAR, F., GHAHRAMANI, Z., AND HERNÁNDEZ-LOBATO, J. M. Collaborative gaussian processes for preference learning. In *Advances in neural information processing systems* (2012), pp. 2096–2104.

[90] HUANG, C.-W., KRUEGER, D., LACOSTE, A., AND COURVILLE, A. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779* (2018).

[91] HUANG, D., ALLEN, T. T., NOTZ, W. I., AND ZENG, N. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of global optimization 34*, 3 (2006), 441–466.

[92] HUANG, G.-B., AND CHEN, L. Convex incremental extreme learning machine. *Neurocomputing 70*, 16 (2007), 3056–3062.

[93] HUANG, G.-B., AND CHEN, L. Enhanced random search based incremental extreme learning machine. *Neurocomputing 71*, 16 (2008), 3460–3468.

[94] HUANG, G.-B., ZHU, Q.-Y., AND SIEW, C.-K. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on* (2004), vol. 2, IEEE, pp. 985–990.

[95] HUANG, G.-B., ZHU, Q.-Y., AND SIEW, C.-K. Extreme learning machine: theory and applications. *Neurocomputing 70*, 1 (2006), 489–501.

[96] HÜLLERMEIER, E., FÜRNKRANZ, J., CHENG, W., AND BRINKER, K. Label ranking by learning pairwise preferences. *Artificial Intelligence 172*, 16-17 (2008), 1897–1916.

[97] HUSMEIER, D. Random vector functional link (rvfl) networks. In *Neural Networks for Conditional Probability Estimation*. Springer, 1999, pp. 87–97.

[98] HUSZÁR, F. Variational inference using implicit distributions. *arXiv preprint arXiv:1702.08235* (2017).

[99] HUTTER, F., HOOS, H., AND LEYTON-BROWN, K. An evaluation of sequential model-based optimization for expensive blackbox functions. In *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation* (2013), ACM, pp. 1209–1216.

[100] HUTTER, F., HOOS, H. H., AND LEYTON-BROWN, K. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization* (2011), Springer, pp. 507–523.

[101] IGELNIK, B., AND PAO, Y.-H. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. *IEEE Transactions on Neural Networks 6*, 6 (1995), 1320–1329.

[102] ILIEVSKI, I., AKHTAR, T., FENG, J., AND SHOEMAKER, C. A. Hyperparameter Optimization of Deep Neural Networks Using Non-Probabilistic RBF Surrogate Model, July 2016.

[103] IZMAILOV, P., NOVIKOV, A., AND KROPOTOV, D. Scalable gaussian processes with billions of inducing inputs via tensor train decomposition. *arXiv preprint arXiv:1710.07324* (2017).

[104] JACOT, A., GABRIEL, F., AND HONGLER, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems* (2018), pp. 8571–8580.

[105] JAMIESON, K., AND RECHT, B. The news on auto-tuning.

[106] JONES, D. R. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization 21*, 4 (2001), 345–383.

[107] JONES, D. R., SCHONLAU, M., AND WELCH, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization 13*, 4 (1998), 455–492.

[108] JOY, T. T., RANA, S., GUPTA, S., AND VENKATESH, S. Fast hyperparameter tuning using bayesian optimization with directional derivatives. *arXiv preprint arXiv:1902.02416* (2019).

[109] JU, S., SHIGA, T., FENG, L., HOU, Z., TSUDA, K., AND SHIOMI, J. Designing nanostructures for phonon transport via bayesian optimization. *Physical Review X 7*, 2 (2017), 021024.

[110] KAUR, P., SINGH, M., AND JOSAN, G. S. Comparative analysis of rank aggregation techniques for metasearch using genetic algorithm. *Education and Information Technologies 22*, 3 (2017), 965–983.

[111] KAZAMA, M., AND TAKAHASHI, V. Active preference learning for generative adversarial networks. In *2017 IEEE International Conference on Big Data (Big Data)* (2017), IEEE, pp. 4389–4393.

[112] KERSTING, K., PLAGEMANN, C., PFAFF, P., AND BURGARD, W. Most likely heteroscedastic gaussian process regression. In *Proceedings of the 24th international conference on Machine learning* (2007), ACM, pp. 393–400.

[113] KIKUCHI, S., ODA, H., KIYOHARA, S., AND MIZOGUCHI, T. Bayesian optimization for efficient determination of metal oxide grain boundary structures. *Physica B: Condensed Matter 532* (2018), 24–28.

[114] KINGMA, D. P., AND WELLING, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[115] KJELLSTROM, G., AND TAXEN, L. Stochastic optimization in system design. *IEEE Transactions on Circuits and Systems 28*, 7 (1981), 702–715.

[116] KLEIN, A., FALKNER, S., BARTELS, S., HENNIG, P., AND HUTTER, F. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079* (2016).

[117] KLEIN, A., FALKNER, S., MANSUR, N., AND HUTTER, F. Robo: A flexible and robust bayesian optimization framework in python. In *NIPS workshop on Bayesian Optimization (BayesOpt17)* (2017).

[118] KNOLLMÜLLER, J., AND ENSSLIN, T. A. Metric gaussian variational inference. *arXiv preprint arXiv:1901.11033* (2019).

[119] KRUEGER, D., HUANG, C.-W., ISLAM, R., TURNER, R., LACOSTE, A., AND COURVILLE, A. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759* (2017).

[120] KRUSCHKE, J. *Doing Bayesian data analysis: A tutorial introduction with R*. Academic Press, 2010.

[121] LAKSHMINARAYANAN, B., PRITZEL, A., AND BLUNDELL, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems* (2017), pp. 6402–6413.

[122] LAM, R., WILLCOX, K., AND WOLPERT, D. H. Bayesian optimization with a finite budget: An approximate dynamic programming approach. In *Advances in Neural Information Processing Systems* (2016), pp. 883–891.

[123] LÁZARO-GREDILLA, M., AND TITSIAS, M. K. Variational heteroscedastic gaussian process regression. In *ICML* (2011), pp. 841–848.

[124] LeCun, Y., Bengio, Y., et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks 3361*, 10 (1995), 1995.

[125] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165* (2017).

[126] Lemaréchal, C. Nondifferentiable optimization. *Handbooks in operations research and management science 1* (1989), 529–572.

[127] Levesque, J.-C., Gagn, C., and Sabourin, R. Bayesian Hyperparameter Optimization for Ensemble Learning. *CoRR abs/1605.06394* (2016).

[128] Li, C., Gupta, S., Rana, S., Nguyen, V., Venkatesh, S., and Shilton, A. High dimensional bayesian optimization using dropout. *arXiv preprint arXiv:1802.05400* (2018).

[129] Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560* (2016).

[130] Liaw, A., Wiener, M., et al. Classification and regression by randomforest. *R news 2*, 3 (2002), 18–22.

[131] Lin, S. Rank aggregation methods. *Wiley Interdisciplinary Reviews: Computational Statistics 2*, 5 (2010), 555–570.

[132] Livingstone, S., Faulkner, M. F., and Roberts, G. O. Kinetic energy choice in hamiltonian/hybrid monte carlo. *Biometrika 106*, 2 (2019), 303–319.

[133] Lizotte, D., Wang, T., Bowling, M., and Schuurmans, D. Automatic gait optimization with gaussian process regression. In *in Proc. of IJCAI* (2007), pp. 944–949.

[134] LOCATELLI, M. Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization 10*, 1 (1997), 57–76.

[135] LORRAINE, J., AND DUVENAUD, D. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419* (2018).

[136] LOUIZOS, C., AND WELLING, M. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning* (2016), pp. 1708–1716.

[137] LOUIZOS, C., AND WELLING, M. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (2017), JMLR. org, pp. 2218–2227.

[138] LU, X., AND VAN ROY, B. Ensemble sampling. In *Advances in Neural Information Processing Systems* (2017), pp. 3258–3266.

[139] MACKAY, D. J. Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks. *Network: Computation in Neural Systems 6*, 3 (1995), 469–505.

[140] MAIR, S., AND BREFELD, U. Distributed robust gaussian process regression. *Knowledge and Information Systems 55*, 2 (2018), 415–435.

[141] MARCHANT, T. Valued relations aggregation with the borda method. *Journal of Multi-Criteria Decision Analysis 5*, 2 (1996), 127–132.

[142] MARQUARDT, D. W., AND SNEE, R. D. Ridge regression in practice. *The American Statistician 29*, 1 (1975), 3–20.

[143] MATTHEWS, A. G. D. G., ROWLAND, M., HRON, J., TURNER, R. E., AND GHAHRAMANI, Z. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271* (2018).

[144] MCGRORY, C. A., AND TITTERINGTON, D. Variational approximations in bayesian model selection for finite mixture distributions. *Computational Statistics & Data Analysis 51*, 11 (2007), 5352–5367.

[145] MELKUMYAN, A., AND RAMOS, F. T. A sparse covariance function for exact gaussian process inference in large datasets. In *Twenty-First International Joint Conference on Artificial Intelligence* (2009).

[146] MENDES-MOREIRA, J., SOARES, C., JORGE, A. M., AND SOUSA, J. F. D. Ensemble approaches for regression: A survey. *Acm computing surveys (csur) 45*, 1 (2012), 10.

[147] MICHE, Y., SORJAMAA, A., BAS, P., SIMULA, O., JUTTEN, C., AND LENDASSE, A. Op-elm: optimally pruned extreme learning machine. *Neural Networks, IEEE Transactions on 21*, 1 (2010), 158–162.

[148] MINKA, T., ET AL. Divergence measures and message passing. Tech. rep., Technical report, Microsoft Research, 2005.

[149] MISHRA, H. K., AND SEKHAR, C. C. Variational gaussian mixture models for speech emotion recognition. In *2009 Seventh International Conference on Advances in Pattern Recognition* (2009), IEEE, pp. 183–186.

[150] MOČKUS, J. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference* (1975), Springer, pp. 400–404.

[151] MOCKUS, J. *The Bayesian approach to global optimization*. Springer, 1982.

[152] MOCKUS, J. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization 4*, 4 (1994), 347–365.

[153] MOHAMED, S., AND LAKSHMINARAYANAN, B. Learning in implicit generative models. *arXiv preprint arXiv:1610.03483* (2016).

[154] MORALES-ENCISO, S., AND BRANKE, J. Tracking global optima in dynamic environments with efficient global optimization. *European Journal of Operational Research 242*, 3 (2015), 744–755.

[155] MOUSTAPHA, M., SUDRET, B., BOURINET, J.-M., AND GUILLAUME, B. Adaptive kriging reliability-based design optimization of an automotive body structure under crashworthiness constraints. 12th International Conference on Applications of Statistics and Probability in Civil Engineering, 2015, Kanada.

[156] NEAL, R. M. Priors for infinite networks. In *Bayesian Learning for Neural Networks*. Springer, 1996, pp. 29–53.

[157] NEAL, R. M. MCMC using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo 2* (2011).

[158] NEAL, R. M. *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[159] NOWOZIN, S., CSEKE, B., AND TOMIOKA, R. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems* (2016), pp. 271–279.

[160] ONG, V. M.-H., NOTT, D. J., AND SMITH, M. S. Gaussian variational approximation with a factor covariance structure. *Journal of Computational and Graphical Statistics 27*, 3 (2018), 465–478.

[161] ORMEROD, J. T., AND WAND, M. P. Gaussian variational approximate inference for generalized linear mixed models. *Journal of Computational and Graphical Statistics 21*, 1 (2012), 2–17.

[162] OSBAND, I., ASLANIDES, J., AND CASSIRER, A. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems* (2018), pp. 8626–8638.

[163] OSBORNE, M. A., GARNETT, R., AND ROBERTS, S. J. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)* (2009), vol. 2009.

[164] PAO, Y.-H., PARK, G.-H., AND SOBAJIC, D. J. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing 6*, 2 (1994), 163–180.

[165] PAPAMAKARIOS, G., PAVLAKOU, T., AND MURRAY, I. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems* (2017), pp. 2338–2347.

[166] PAWLOWSKI, N., BROCK, A., LEE, M. C., RAJCHL, M., AND GLOCKER, B. Implicit weight uncertainty in neural networks. *arXiv preprint arXiv:1711.01297* (2017).

[167] PEARCE, T., ZAKI, M., BRINTRUP, A., AND NEEL, A. Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546* (2018).

[168] PEARCE, T., ZAKI, M., AND NEELY, A. Bayesian neural network ensembles. *arXiv preprint arXiv:1811.12188* (2018).

[169] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[170] PICHENY, V., GINSBOURGER, D., RICHET, Y., AND CAPLIN, G. Quantile-based optimization of noisy computer experiments with tunable precision. *Technometrics 55*, 1 (2013), 2–13.

[171] POŠÍK, P., HUYER, W., AND PÁL, L. A comparison of global search algorithms for continuous black box optimization. *Evolutionary computation 20*, 4 (2012), 509–541.

[172] QUIÑONERO CANDELA, J., AND RASMUSSEN, C. E. A unifying view of sparse approximate gaussian process regression. *The Journal of Machine Learning Research 6* (2005), 1939–1959.

[173] QUTTINEH, N.-H., AND HOLMSTRÖM, K. *Implementation of a one-stage efficient global optimization (EGO) algorithm*. 2009.

[174] RAHIMI, A., AND RECHT, B. Random features for large-scale kernel machines. In *Advances in neural information processing systems* (2008), pp. 1177–1184.

[175] RAHIMI, A., AND RECHT, B. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing* (2008), IEEE, pp. 555–561.

[176] RANGANATH, R., GERRISH, S., AND BLEI, D. Black box variational inference. In *Artificial Intelligence and Statistics* (2014), pp. 814–822.

[177] RATZLAFF, N., AND FUXIN, L. Hypergan: A generative model for diverse, performant neural networks. *arXiv preprint arXiv:1901.11058* (2019).

[178] REGIS, R. G., AND SHOEMAKER, C. A. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing 19*, 4 (2007), 497–509.

[179] REGLI, J.-B., AND SILVA, R. Alpha-beta divergence for variational inference. *arXiv preprint arXiv:1805.01045* (2018).

[180] REZENDE, D. J., AND MOHAMED, S. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770* (2015).

[181] REZENDE, D. J., MOHAMED, S., AND WIERSTRA, D. Stochastic backpropagation and variational inference in deep latent gaussian models. *arXiv preprint arXiv:1401.4082* (2014).

[182] ROUX, E., AND BOUCHARD, P.-O. Kriging metamodel global optimization of clinching joining processes accounting for ductile damage. *Journal of Materials Processing Technology 213*, 7 (2013), 1038–1047.

[183] RUDI, A., AND ROSASCO, L. Generalization properties of learning with random features. In *Advances in Neural Information Processing Systems* (2017), pp. 3215–3225.

[184] SALIMANS, T., AND KINGMA, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems* (2016), pp. 901–909.

[185] SANTNER, T. J., WILLIAMS, B. J., AND NOTZ, W. I. *The Design and Analysis of Computer Experiments* . Springer Science & Business Media, 2013.

[186] SCARDAPANE, S., WANG, D., AND UNCINI, A. Bayesian random vector functional-link networks for robust data modeling. *IEEE transactions on cybernetics 48*, 7 (2018), 2049–2059.

[187] SCHMIDT, W. F., KRAAIJVELD, M., DUIN, R. P., ET AL. Feedforward neural networks with random weights. In *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on* (1992), IEEE, pp. 1–4.

[188] SNELSON, E. L. *Flexible and efficient Gaussian process models for machine learning*. PhD thesis, Citeseer, 2007.

[189] SNOEK, J., LAROCHELLE, H., AND ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (2012), pp. 2951–2959.

[190] SNOEK, J., RIPPEL, O., SWERSKY, K., KIROS, R., SATISH, N., SUNDARAM, N., PATWARY, M., PRABHAT, M., AND ADAMS, R. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning* (2015), pp. 2171–2180.

[191] SNOEK, J., RIPPEL, O., SWERSKY, K., KIROS, R., SATISH, N., SUNDARAM, N., PATWARY, M. M. A., PRABHAT, AND ADAMS, R. P. Scalable Bayesian Optimization Using Deep Neural Networks. In *ICML* (2015), F. R. Bach and D. M. Blei, Eds., vol. 37 of *JMLR Workshop and Conference Proceedings*, JMLR.org, p. 21712180.

[192] SPRINGENBERG, J. T., KLEIN, A., FALKNER, S., AND HUTTER, F. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems* (2016), pp. 4134–4142.

[193] SRIVASTAVA, A., VALKOV, L., RUSSELL, C., GUTMANN, M. U., AND SUTTON, C. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems* (2017), pp. 3308–3318.

[194] STAN DEVELOPMENT TEAM. Pystan: the python interface to stan, version 2.9.0, 2016.

[195] SUI, Y., GOTOVOS, A., BURDICK, J. W., AND KRAUSE, A. Safe exploration for optimization with gaussian processes. *Proceedings of Machine Learning Research 37* (2015), 997–1005.

[196] TAN, L. S., AND NOTT, D. J. Gaussian variational approximation with sparse precision matrices. *Statistics and Computing 28*, 2 (2018), 259–275.

[197] TESCH, M., SCHNEIDER, J., AND CHOSET, H. Using response surfaces and expected improvement to optimize snake robot gait pa-

rameters. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on* (2011), IEEE, pp. 1069–1074.

[198] TRAN, D., RANGANATH, R., AND BLEI, D. Hierarchical implicit models and likelihood-free variational inference. In *Advances in Neural Information Processing Systems* (2017), pp. 5523–5533.

[199] TRAN, D., RANGANATH, R., AND BLEI, D. M. The variational gaussian process. *arXiv preprint arXiv:1511.06499* (2015).

[200] VANDERPLAATS, G. Methods of mathematical optimization. In *Optimization: Methods and Applications, Possibilities and Limitations*. Springer, 1989, pp. 22–41.

[201] VAZQUEZ, E., AND BECT, J. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and inference 140*, 11 (2010), 3088–3095.

[202] VILLEMONTEIX, J., VAZQUEZ, E., AND WALTER, E. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization 44*, 4 (2009), 509.

[203] WAINWRIGHT, M. J., JORDAN, M. I., ET AL. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning 1*, 1–2 (2008), 1–305.

[204] WANG, B., TITTERINGTON, D., ET AL. Convergence properties of a general algorithm for calculating variational bayesian estimates for a normal mixture model. *Bayesian Analysis 1*, 3 (2006), 625–650.

[205] WANG, D., AND LI, M. Stochastic configuration networks: Fundamentals and algorithms. *IEEE transactions on cybernetics 47*, 10 (2017), 3466–3479.

[206] WANG, K.-C., VICOL, P., LUCAS, J., GU, L., GROSSE, R., AND ZEMEL, R. Adversarial distillation of bayesian neural network posteriors. *arXiv preprint arXiv:1806.10317* (2018).

[207] WANG, W., AND WELCH, W. J. Bayesian optimization using monotonicity information and its application in machine learning hyperparameter. *arXiv preprint arXiv:1802.03532* (2018).

[208] WANG, Z., BARDSLEY, J. M., SOLONEN, A., CUI, T., AND MARZOUK, Y. M. Bayesian inverse problems with l_1 priors: A randomize-then-optimize approach. *SIAM Journal on Scientific Computing 39*, 5, S140–S166.

[209] WATANABE, S., AND LE ROUX, J. Black box optimization for automatic speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on* (2014), IEEE, pp. 3256–3260.

[210] WILLIAMS, B. J., SANTNER, T. J., AND NOTZ, W. I. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica* (2000), 1133–1152.

[211] WILLIAMS, C. K., AND RASMUSSEN, C. E. Gaussian processes for machine learning. *the MIT Press 2*, 3 (2006), 4.

[212] WILSON, A. G., DANN, C., AND NICKISCH, H. Thoughts on massively scalable gaussian processes. *arXiv preprint arXiv:1511.01870* (2015).

[213] WILSON, A. G., AND NICKISCH, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *ICML* (2015), pp. 1775–1784.

[214] XIAO, Y., DENG, Y., WU, J., DENG, H.-Z., AND LU, X. Comparison of rank aggregation methods based on inherent ability. *Naval Research Logistics (NRL) 64*, 7 (2017), 556–565.

[215] YIN, B., TRAN, L., LI, H., SHEN, X., AND LIU, X. Towards interpretable face recognition. *arXiv preprint arXiv:1805.00611* (2018).

[216] YONDO, R., ANDRÉS, E., AND VALERO, E. A review on design of experiments and surrogate models in aircraft real-time and many-query aerodynamic analyses. *Progress in Aerospace Sciences 96* (2018), 23–61.

[217] YU, F. X. X., SURESH, A. T., CHOROMANSKI, K. M., HOLTMANN-RICE, D. N., AND KUMAR, S. Orthogonal random features. In *Advances in Neural Information Processing Systems* (2016), pp. 1975–1983.

[218] ZHANG, C., BUTEPAGE, J., KJELLSTROM, H., AND MANDT, S. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[219] ZHANG, C., REN, M., AND URTASUN, R. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749* (2018).

[220] ZHANG, L., AND SUGANTHAN, P. N. A comprehensive evaluation of random vector functional link networks. *Information sciences 367* (2016), 1094–1105.

[221] ZHU, Q.-Y., QIN, A. K., SUGANTHAN, P. N., AND HUANG, G.-B. Evolutionary extreme learning machine. *Pattern recognition 38*, 10 (2005), 1759–1763.

[222] ZOU, H., AND HASTIE, T. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology) 67*, 2 (2005), 301–320.