UNIVERSIDADE FEDERAL DE MINAS GERAIS
Departamento de Ciência da Computação
Programa de Pós-graduação em Ciência da Computação

Artur Luis Fernandes de Souza

BAYESIAN OPTIMIZATION WITH A PRIOR FOR THE OPTIMUM

Belo Horizonte
2022

Artur Luis Fernandes de Souza

BAYESIAN OPTIMIZATION WITH A PRIOR FOR THE OPTIMUM

Versão final

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientador: Leonardo Barbosa e Oliveira

Coorientador: Luigi Nardi

Belo Horizonte
2022

Souza, Artur Luis Fernandes de

S729b Bayesian optimization with a prior for the optimum [manuscrito] / Artur Luis Fernandes de Souza. – 2022.
85 f. il.

Orientador: Leonardo Barbosa e Oliveira.
Coorientador: Luigi Nardi.
Tese (Doutorado) - Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Ciências da Computação.
Referências: f. 71-76.

1. Computação – Teses. 2. Otimização bayesiana – Tese. 3. Aprendizado de máquina – Teses. 4. Markov, Processos de – Teses.I. Oliveira, Leonardo Barbosa e. II. Nardi, Luigi III. Universidade Federal de Minas Gerais, Instituto de Ciências Exatas, Departamento de Computação. IV.Título.

CDU 519.6*82.10(043)

Ficha Ficha catalográfica elaborada pela bibliotecária Belkiz Inez Rezende Costa CRB 6/1510 Universidade Federal de Minas Gerais - ICEx

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Bayesian Optimization with a Prior for the Optimum

ARTUR LUIS FERNANDES DE SOUZA

Tese defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]

Prof. Leonardo Barbosa e Oliveira - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Luigi Nardi - Coorientador
Department of Computer Science - Lund University

Gisele L Pappa

Profa. Gisele Lobo Pappa
Departamento de Ciência da Computação - UFMG

[illegible]

Prof. Ricardo Bastos Cavalcante Prudencio
Centro de Informática - UFPE

Eduardo Valle

Prof. Eduardo Alves do Valle Junior
Departamento de Engenharia Elétrica e da Computação - UNICAMP

[illegible]

Prof. Heitor Soares Ramos Filho
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 24 de março de 2022.

Resumo

Otimização Bayesiana (Bayesian Optimization – BO) é uma ferramenta eficiente para a otimização de decisões de projeto que tem ganhado grande popularidade nos últimos anos. BO tem impactado uma vasta gama de áreas, de aprendizado de máquina à cristalografia serial. Porém, embora BO seja um método popular para a otimização de funções caixa-preta, BO não é capaz de aproveitar da experiência de especialistas humanos no processo de otimização. Especialistas humanos frequentemente tem intuições sobre quais regiões do espaço de busca tem maior chance de trazer bons resultados, porém, não tem suporte apropriado para injetar esse conhecimento na BO. Isso faz com que BO desperdice avaliações da função em regiões sabidamente ruins do espaço de busca, desacelerando a convergência. Para tratar esse problema, nós introduzimos *Bayesian Optimization with a Prior for the Optimum* (BOPrO). BOPrO permite que usuários injetem seu conhecimento no processo de otimização na forma de priors de quais regiões do espaço de busca levarão à melhor performance, no lugar dos priors sob funções tradicionais de BO, que são muito menos intuitivos para usuários. BOPrO então combina esses priors com o modelo probabilístico tradicional de BO para construir uma distribuição pseudo-posterior de boas regiões do espaço de busca. Nós avaliamos BOPrO em um conjunto de funções sintéticas e demonstramos que BOPrO é mais eficiente em número de avaliações do que métodos do estado da arte sem priors de usuários, outras abordagens com suporte para injeção de priors e 10,000× mais rápido do que a busca aleatória. Nós também comparamos BOPrO com o estado da arte em uma aplicação real de projeto de hardware com priors providenciados por um especialista humano e mostramos que BOPrO atinge um novo estado da arte. Por fim, demonstramos que BOPrO converge mais rápido mesmo que o prior do usuário não seja perfeitamente preciso e que BOPrO robustamente recupera de priors incorretos.

Palavras-chave: Otimização Bayesiana. Aprendizado de Máquina Automatizado. Processos Gaussianos.

Abstract

Bayesian Optimization (BO) is a data-efficient tool for the joint optimization of design choices that is gaining great popularity in recent years. BO has impacted a wide range of areas, ranging from ML hyperparameter optimization to serial crystallography. However, while BO has become a popular method for optimizing expensive black-box functions, it fails to leverage the experience of human domain experts. Human experts often have intuitions on which regions of the design space are more likely to yield good results, but have no good way to encode this knowledge into BO. This causes BO to waste function evaluations on commonly known bad regions of design choices, slowing down convergence. To address this issue, we introduce Bayesian Optimization with a Prior for the Optimum (BOPrO). BOPrO allows users to inject their knowledge into the optimization process in the form of priors about which parts of the input space will yield the best performance, rather than BO's standard priors over functions which are much less intuitive for users. BOPrO then combines these priors with BO's standard probabilistic model to yield a pseudo-posterior distribution on good regions of the design space. We evaluate BOPrO on a suite of synthetic benchmarks, specially tailored to evaluating BO methods, and show that BOPrO is more sample efficient than state-of-the-art methods without user priors, previous approaches that support prior injection, and 10,000× faster than random search. We also compare BOPrO to the state-of-the-art on a real-world hardware design application with priors provided by a human application expert and once again show that BOPrO sets a new state-of-the-art performance. We also perform a series of ablation studies showing that BOPrO converges faster even if the user priors are not entirely accurate and that it robustly recovers from misleading priors.

Keywords: Bayesian Optimization. Automated Machine Learning. Gaussian Process.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Replacing manual tasks with new automated solutions is a growing tendency in both industry and academia. Scientists develop automated models to speed up experiments and discovery, while industry solutions grow smarter at a remarkable rate. However, as these automated solutions become more powerful, they also become more complex and require the fine-tuning of more parameters in order to be effective. To complicate things further, the fine-tuning of these parameters is often domain-specific and, thus, there is no single configuration that works well for all applications.

Traditionally, human experts are tasked with manually fine-tuning these parameters. These experts spend days, or even months, experimenting with different decisions to build an intuition of what are the (approximate) best parameters for each application. This is an expensive approach that becomes highly ineffective as automated solutions become more complex. The high cost of fine-tuning these increasingly complex automations calls for the development of efficient parameter optimization solutions.

Bayesian Optimization (BO) has emerged as a powerful solution for this parameter optimization problem. Optimization problems can be described as optimizing an unknown (black-box) function f subject to a set of parameters X. BO tackles this problem by building a surrogate model $\mathcal{M}$ to approximate the behavior of f and then using an acquisition function on top of $\mathcal{M}$ to decide on promising parameter combinations to explore [Shahriari et al., 2015]. This approach allows BO to find good parameter configurations with few function evaluations, making BO particularly well suited to optimize expensive black-box functions [Brochu et al., 2010]. Figure 1.1 illustrates a standard BO algorithm.

BO has gained great popularity in recent years. It is impacting a wide range of areas, including hyperparameter optimization [Snoek et al., 2012; Falkner et al., 2018], AutoML [Feurer et al., 2015a; Hutter et al., 2018], robotics [Calandra et al.,

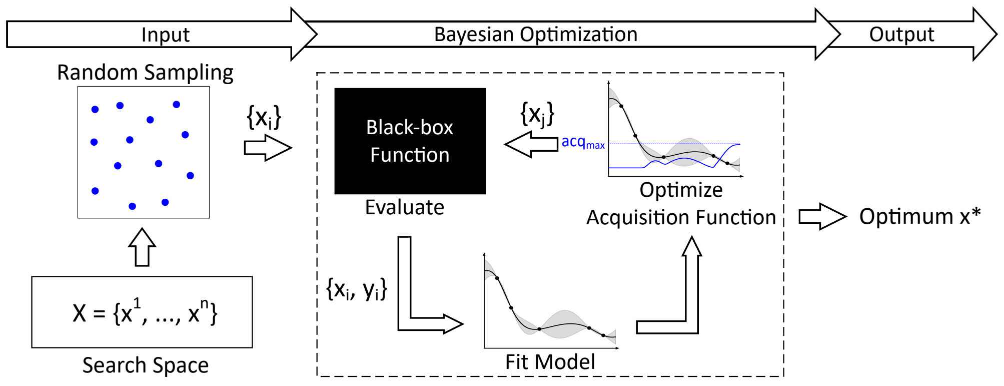


Figure 1.1. Illustration of a BO algorithm. x^i are the input parameters, with $x^i \in \mathcal{X}$. $\{x_i, y_i\}$ is the input parameter configuration explored at iteration i and the respective objective function value.

2016], computer vision [Bodin et al., 2016; Nardi et al., 2017], environmental monitoring [Marchant and Ramos, 2012], combinatorial optimization [Hutter et al., 2011], experimental design [Azimi et al., 2012], RL [Brochu et al., 2010], Computer Go [Chen et al., 2018], hardware design [Nardi et al., 2018; Koeplinger et al., 2018], seria crystallography [Souza et al., 2019], and many others. It promises greater automation so as to increase both product quality and human productivity. As a result, BO is also established in many large tech companies (e.g., with Google Vizier [Golovin et al., 2017] and Facebook BoTorch [Balandat et al., 2019]).

However, domain experts often have substantial prior knowledge that BO cannot incorporate. Users can incorporate prior knowledge by narrowing the search space; however, this type of hard prior can lead to poor performance by missing important regions. BO also natively supports a prior over functions $p(f)$, e.g., specified by a kernel function. However, this is not the prior human experts have: users often know which ranges of hyperparameters tend to work best, and are able to specify a probability distribution $p_{\text{best}}(\boldsymbol{x})$ to quantify these priors. For example, many users of the Adam optimizer [Kingma and Ba, 2015] know that its best learning rate is often in the vicinity of 1e-3 (give or take one order of magnitude), yet one may not know the accuracy one may achieve in a new application.

As a result, many competent users instead revert to manual search, which can fully incorporate their prior knowledge. A recent survey showed that most NeurIPS 2019 and ICLR 2020 papers that reported having tuned hyperparameters used manual search, with only a very small fraction using BO [Bouthillier and Varoquaux, 2020].

In order for BO to be adopted widely, and help facilitate faster progress in the ML community by tuning hyperparameters faster and better, it is, therefore, crucial to devise a method that fully incorporates priors into BO. This is precisely the main contribution of this proposal.

1.1 Goals

Our goal in this work is to develop a BO framework that allows human experts to inject their priors into the optimization process. We aim at a solution that is flexible and simple in the prior injection, so that it can be easily employed by non-BO experts. We also aim at a solution that is robust in that it can still converge to good function values, even if the user priors are wrong about its locality. At last, naturally, our solution must also be more efficient in number of evaluations than previous state-of-the-art methods when priors are provided.

To achieve these goals, we propose Bayesian Optimization with a Prior for the Optimum (BOPrO). BOPrO allows users to inject their knowledge into the optimization process in the form of priors about which parts of the input space will yield the best performance. BOPrO then combines these priors with BO's standard probabilistic model to yield a pseudo-posterior on "good" regions of the space. BOPrO achieves improved statistical efficiency over state-of-the-art methods by leveraging the injected prior knowledge. At the same time, BOPrO is also a robust approach that can recover in case the user-defined prior is misleading and still achieve good performance. In addition, BOPrO is flexible in that it allows its users to specify priors for any subset of input parameters, including all or none, and any form of probability distribution.

We evaluate BOPrO empirically on a number of synthetic benchmarks, as well as a real-world application. We compare BOPrO with state-of-the-art BO approaches, including other approaches that support user priors in BO, and demonstrate that BOPrO outperforms these approaches. We also evaluate BOPrO on a real-world hardware design application with an unbiased human-expert prior and show that it leads to improved performance over the current state-of-the-art.

1.2 Contributions

The contributions we make with BOPrO are as follows:

- We introduce *Bayesian Optimization with a Prior over the Optimum*, short *BOPrO*, which allows users to inject priors that were previously difficult to inject into BO, such as Gaussian, exponential, multimodal, and multivariate priors for the location of the optimum.

- BOPrO's model bridges the gap between the well-established Tree-structured Parzen Estimator (TPE) methodology, which is based on Parzen kernel density estimators, and standard BO probabilistic models, such as Gaussian Processes (GPs) and Random Forests (RFs).
- We demonstrate that accurate prior knowledge helps BOPrO to dramatically outperform standard BO and 10,000x random search.
- We demonstrate that BOPrO outperforms the previous state-of-the-art of BO with support for prior injection.
- We demonstrate BOPrO on a real-world application, with priors provided by an unbiased human expert.
- We demonstrate that BOPrO overcomes misleading prior knowledge and still finds optimal configurations.

Simultaneous to the development of BOPrO, we have also worked on the development of the open-source HyperMapper black-box optimization framework[1]. BOPrO is made publicly available as part of the HyperMapper framework[2].

1.3 Dissertation Question

The research question that guided us in this Ph.D. was: can we leverage the prior knowledge that experts have in order to improve the performance of BO? In order to answer this question, we conceived a novel approach that combines user knowledge, in the form of prior distributions, with the probabilistic approach of BO. We then evaluated our novel approach to show that the user knowledge leads to improved performance in BO, in terms of function evaluations.

1.4 Dissertation Statement

Our hypothesis in this thesis is that we are able to inject user priors into BO and that it leads to improved performance. We believe this can be achieved by computing

[1]`https://github.com/luinardi/hypermapper/`
[2]`https://github.com/luinardi/hypermapper/wiki/prior-injection`

a pseudo-posterior that combines the user prior with BO's probabilistic model. The user prior will ensure BO focuses on important parts of the search space, while BO's probabilistic model will learn from the data and refine the prior knowledge in order to find better function values.

1.5 Organization

This work is structured as follows. In section 2, we provide a background of BO and the building blocks of this work. In section 4, we introduce BOPrO, together with a detailed explanation of its components. Section 5 shows empirical results for BOPrO and comparisons to other state-of-the-art methods. We discuss related work in section 3 and conclude the work in Section 7.

Chapter 2

Background

2.1 Bayesian Optimization

Bayesian Optimization (BO) is a powerful tool to solve optimization problems. BO aims to find the optimum of a function over a space of parameters. The function being optimized is often unknown and we can only query it for the outcome of a specific parameter configuration. This function is usually referred to as a *black-box function*, while the space of possible parameter configurations is often called the problem's *search space*. The main advantage of BO is that it is one of the most efficient approaches considering the number of black-box function evaluations [Shahriari et al., 2015], which makes BO a great tool for optimizing expensive black-box functions.

BO uses an approach for optimization based on Bayes' theorem. BO builds a posterior model on the black-box function based on an initial prior model[1] and a set of evaluated configurations (the evidence). BO, then, uses the posterior model to make informed decisions on which configuration should be evaluated next. BO is an iterative algorithm, at each iteration, a new configuration is evaluated and the posterior model is updated with the new evidence. It is common, however, for BO to randomly sample configurations from the search space to bootstrap the model, before starting the iterative BO loop. The BO loop is executed for a pre-specified budget, usually on the number of function evaluations or wall-clock time.

The posterior model is a key component of BO. It is paramount that the BO model accurately predicts the behavior of the black-box function and be cheap to evaluate. Another important property for BO models is to accurately estimate their own uncertainty regarding their predictions. This allows BO to reason over which

[1]Here, we refer to the model's prior over functions and not the user's prior

configurations are more valuable to explore considering the quality of the solution (exploitation) and its own uncertainty regarding the black-box function (exploration). This is the fundamental *exploration* $\times$ *exploitation* trade-off in BO. Common model choices for BO are Gaussian Process (GPs) [Snoek et al., 2012], Random Forests (RFs) [Hutter et al., 2011], and Tree-structured Parzen Estimators (TPEs) [Bergstra et al., 2011].

Another key component of BO is the acquisition function. The acquisition function dictates which configurations should be explored next, considering the exploration and exploitation trade-off. The acquisition function quantifies the quality of each configuration and the next configuration to explore is chosen by maximizing the acquisition function. To be effective, the acquisition function must properly balance exploration and exploitation. Too much focus on exploration will lead to a good understanding of the black-box function, but a weak optimized value. Too much focus on exploitation will lead to a poor understanding of the black-box function and likely lead optimization to a local minimum (or maximum). Common choices of acquisition function are Expected Improvement (EI), Upper Confidence Bound (UCB), and Thompson Sampling (TS) [Shahriari et al., 2015].

Algorithm 1 shows a standard BO algorithm. BO starts with an initialization phase, randomly sampling a number of configurations in line 3 and evaluating them in line 4. The main BO loop is then started in line 6: BO fits the model with the data observed so far in line 7, chooses the next configuration to explore by maximizing the acquisition function in line 8, evaluates the new configuration with the black-box function in line 9, and adds it to the history of evaluated configurations in line 10. This loop is repeated for a pre-specified number of times (the budget) and the best configuration found is returned at the end in line 12. The search space, budget, and number of initialization configurations are hyperparameters of the BO algorithm.

2.2 Gaussian Process Regression

A GP can be described as a regression model that defines a distribution over functions [Williams and Rasmussen, 2006]. A GP starts with an assumed prior distribution over possible functions. At first, this prior distribution only makes loose assumptions about the regression mean (often a constant) and the smoothness of the functions in the distribution, i.e., how fast the functions change given a change in the input. This distribution is then updated as new configurations are evaluated to consider only functions that are consistent with the configurations that have been observed. This

Algorithm 1 Bayesian optimization.

```
1: **Input:** Search space $\mathcal{X}$, initialization budget $N$, and BO budget $B$.
2: **Output:** Best configuration found $\boldsymbol{x}_{inc}$.
3: $\boldsymbol{D} \leftarrow Initialize(\mathcal{X}, N)$
4: **for** $t = 1$ **to** $B$ **do**
5:     $\mathcal{M} \leftarrow fit\_model(\boldsymbol{D})$
6:     $\boldsymbol{x_i} \in \arg\max_{\boldsymbol{x} \in \mathcal{X}} acq(\boldsymbol{x}, \mathcal{M})$
7:     $y_i \leftarrow f(\boldsymbol{x_i})$
8:     $\boldsymbol{D} \leftarrow \boldsymbol{D} \cup (\boldsymbol{x_i}, y_i)$
9: **end for**
10: $\boldsymbol{x}_{inc} \leftarrow ComputeBest(\boldsymbol{D})$
11: **return** $\boldsymbol{D}$
```

implies a change in the distribution's mean and the smoothness of the functions in the distribution. This updated distribution is the posterior used in BO.

GPs are defined entirely by their mean and kernel (or covariance function). The mean and kernel define a distribution of functions of the GP, with the kernel dictating the smoothness of these functions. When defining a GP, it is important to choose a kernel that correctly reflects the smoothness expected of the function being modeled. This is the prior over functions of the GP. Then, whenever new configurations are observed, the hyperparameters of the kernel are updated to match the shape of the data, forming the GP posterior. The hyperparameters are usually optimized by maximizing the log marginal likelihood of the observed function values. Common GP kernels are the squared-exponential and matérn kernels [Shahriari et al., 2015].

The predictions of the GP take the form of a gaussian distribution at each point. This means that for each point we can compute a mean value (the prediction of the regressor for that point) and a prediction variance (the uncertainty of the prediction at that point). Both mean and variance depend on the kernel chosen for the GP, though, naturally, the variance in GPs is low near points that have been explored and grows together with the distance to explored points. Formally, the predictive distribution of a GP with kernel K and kernel parameters $\boldsymbol{\theta}$ is a Gaussian $\mathcal{N}(\hat{y} \mid \hat{\mu}, \hat{\sigma^2})$ with posterior mean $\hat{\mu}$ and variance $\hat{\sigma^2}$, where the posterior mean $\hat{\mu}$ is estimated with:

$$\hat{\mu} = K(\boldsymbol{\theta}, \boldsymbol{x}, \mathbf{X}) K(\boldsymbol{\theta}, \mathbf{X}, \mathbf{X})^{-1} \mathbf{y}, \tag{2.1}$$

where K is the GP Kernel and $\boldsymbol{\theta}$ are the kernel hyperparameters. The pseudo-posterior standard deviation (or uncertainty) estimate $\hat{\sigma}$ is defined as:

$$\hat{\sigma^2} = K(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{x}) - K(\boldsymbol{\theta}, \boldsymbol{x}, \mathbf{X}) K(\boldsymbol{\theta}, \mathbf{X}, \mathbf{X})^{-1} K(\boldsymbol{\theta}, \boldsymbol{x}, \mathbf{X})^{\top}. \tag{2.2}$$

2.3 Tree-structured Parzen Estimator

Whereas the standard probabilistic model in BO directly models $p(y|\boldsymbol{x})$, the Tree-structured Parzen Estimator (TPE) approach by Bergstra et al. [2011] models $p(\boldsymbol{x}|y)$ and $p(y)$ instead.[2] This is done by constructing two parametric densities, $g(\boldsymbol{x})$ and $l(\boldsymbol{x})$, which are computed using the observations with function value above and below a given threshold, respectively. The separating threshold y^* is defined as a quantile of the observed function values. TPE uses the densities $g(\boldsymbol{x})$ and $l(\boldsymbol{x})$ to define $p(\boldsymbol{x}|y)$ as:

$$p(\boldsymbol{x}|y) = l(\boldsymbol{x})I(y < y^*) + g(\boldsymbol{x})(1 - I(y < y^*)), \tag{2.3}$$

where $I(y < y^*)$ is 1 when $y < y^*$ and 0 otherwise. The parametrization of the generative model $p(\boldsymbol{x}, y) = p(\boldsymbol{x}|y)p(y)$ facilitates the computation of EI as it leads to $EI_{y^*}(\boldsymbol{x}) \propto l(\boldsymbol{x})/g(\boldsymbol{x})$ and, thus, $\arg\max_{\boldsymbol{x}\in\mathcal{X}} EI_{y^*}(\boldsymbol{x}) = \arg\max_{\boldsymbol{x}\in\mathcal{X}} l(\boldsymbol{x})/g(\boldsymbol{x})$.

2.4 Expected Improvement

EI is an improvement-based acquisition function proposed by Mockus et al. [1978]. Improvement-based acquisition functions seek to improve upon a target value, usually, the target value is the incumbent f_{inc}. In other words, improvement-based acquisition functions quantify the quality of each configuration based on how likely they are to improve on f_{inc}. The first improvement-based policy, Probability of Improvement (PI) [Kushner, 1964], quantifies the quality of each configuration by simply taking the probability of the configuration leading to a value better than f_{inc}.

EI expands on the PI acquisition function by also incorporating the amount of improvement expected from each point [Shahriari et al., 2015]. In other words, EI quantifies the quality of configurations by weighting how likely they are to improve upon f_{inc}, as well as, how much of an improvement it will provide. Formally, EI quantifies the expected improvement upon f_{inc} as:

$$EI_{f_{inc}}(\boldsymbol{x}) = \int_{-\infty}^{\infty} \max(f_{inc} - y, 0)p(y|\boldsymbol{x})dy, \tag{2.4}$$

where the target f_{inc} is usually taken to be the incumbent function value, i.e., the best objective function value found so far, and $p(y|\boldsymbol{x})$ is a probabilistic model, e.g., a

[2] Technically, the model does not parameterize $p(y)$, since it is computed based on the observed data points, which are heavily biased towards low values due to the optimization process. Instead, it parameterizes a dynamically changing $p_t(y)$, which helps to constantly challenge the model to yield better observations.

Gaussian Process. Note that the first term in the integral means that EI vanishes if the improvement is lower than 0, i.e., the quality of a configuration is 0 if it will not improve upon f_{inc}

The prediction of the expected improvement for each configuration is based on the predictions of the surrogate model for the black-box function. If the prediction of our model is normally distributed, as is the case for GPs, we can compute EI analytically with [Shahriari et al., 2015]:

$$EI(\boldsymbol{x}) = (f_{inc} - \mu_{\boldsymbol{x}})\Phi\left(\frac{f_{inc} - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}\right) + \sigma\phi\left(\frac{f_{inc} - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}\right), \tag{2.5}$$

where $\mu_{\boldsymbol{x}}$ and $\sigma_{\boldsymbol{x}}$ are the mean and standard deviation of the surrogate model at $\boldsymbol{x}$ and Φ and ϕ are the normal cumulative distribution function and probability density function, respectively. This formulation works seamlessly with GPs, making it another favorable aspect of GPs for BO, however, other works have shown that it is also possible to compute EI for other surrogate models such as RFs [Hutter et al., 2011] and TPE [Bergstra et al., 2013].

Equation 2.5 shows how EI balances the exploration vs exploitation trade-off. Namely, note that the EI is is higher when $\mu_{\boldsymbol{x}}$ is low and when $\sigma_{\boldsymbol{x}}$ is high. In other words, EI is high when the model's prediction is low, favoring exploitation, and when the uncertainty of the model is high, favoring exploration.

Chapter 3

Related Work

Several authors have worked on BO, proposed their own BO solutions, and steadily advanced the state of the art over the years. In this section, we review some prominent BO solutions in the literature. Our work also relates to meta-learning for BO, thus we also review prominent works on meta-learning for BO.

3.1 Previous BO Approaches

An early notable BO solution in the literature is the work of Jones *et al.* [Jones et al., 1998]. Jones *et al.* propose a BO solution they call Efficient Global Optimization (EGO), built on top of the DACE model [Sacks et al., 1989] and the EI acquisition function. The DACE model relies on a correlation function that treats the model's noise as a function of the distance between points in the space. This correlation function is so effective that the authors can simply assume a constant mean for their stochastic process. Jones *et al.* combine this model and an EI acquisition function, optimized via a branch-and-bound algorithm, to complete the EGO framework.

More recently, Snoek et al. [2012] propose Spearmint, a BO solution based on GPs. Initially, the authors proposed Spearmint as a solution to perceived limitations of previous BO solutions. The authors identified three limitations to existing BO solutions: how to define the GPs kernel and its associated hyperparameters, how to account for widely different black-box evaluation times, and how to leverage parallel computation in BO. In later works, the authors expanded Spearmint to support new features, such as a non-stationary approach based on input warping [Snoek et al., 2014].

Eriksson et al. [2019] also proposed a BO solution based on GPs, but more focused on scalable, high-dimensional optimization. Their solution, dubbed TuRBO, breaks down the search space into "trust regions" and performs BO in these trust regions.

Trust regions are shrunk or expanded based on the performance of BO in each trust region. Once their solution determines that a trust region has been fully explored, new trust regions are selected. This approach of dividing the search space into smaller local regions allows TuRBO to optimize well both locally and globally, which makes it well-suited for high-dimensional problems.

Hutter *et al.* propose a BO framework that replaces the commonly used GPs with a RF model [Hutter et al., 2011]. Their solution, dubbed SMAC, uses a RF model and the EI acquisition function, optimized with a multi-start local search. The RF model is cheaper than GPs and also more flexible, allowing SMAC to seamlessly support discrete and categorical input parameters[1]. Hutter *et al.* show that their RF-based solution performs better than previous solutions, while also being more flexible.

Gardner et al. [2014] propose a solution for constrained BO based on GPs dubbed cBO. Their solution models a separate GP on a cost function and, based on a pre-defined cost threshold, define a "probability of feasibility" for each configuration in the search space. This probability of feasibility allows their solution to handle constrained search spaces, where some parameter configurations are impossible to evaluate. Gardner *et al.* show that their cBO approach is better than traditional BO solutions and random sampling on a number of constrained benchmarks.

In a multi-objective setting, Knowles proposes an extension of the EGO algorithm for multiple objectives called ParEGO [Knowles, 2004]. ParEGO uses a scalarization function and a set of weights to scalarize multiple objectives into a single value. ParEGO then applies the EGO algorithm of Jones *et al.* [Jones et al., 1998] on the scalarized objectives. Similarly, Paria *et al.* propose an approach for multi-objective BO using random scalarizations [Paria et al., 2018]. They propose randomly sampling scalarization weights and using novel scalarized versions of the UCB and Thompson Sampling (TS) acquisition functions to tackle multiple objectives.

Knudde et al. [2017] have also proposed a BO solution based on GPs dubbed GpFlowOpt. Rather than proposing a key insight, GpFlowOpt seeks to provide a flexible, easy-to-use, and customizable framework for GP BO. GpFlowOpt supports an array of acquisition functions, constrained optimization, and multi-objective optimization, all based on GP models.

Recently, BO has also started to attract the attention of large tech companies, with both Google and Facebook developing their own optimization frameworks. Facebook's BoTorch (by Balandat *et al.* [Balandat et al., 2019]) combines state-of-the-art approaches, such as cBO [Gardner et al., 2014] and random scalarizations [Paria et al.,

[1]We note that it is also possible to employ GPs to discrete parameters, but these require input transformations or specially designed kernels, whereas RFs are able to handle these parameters naturally

2018], into a flexible BO framework that seamlessly integrates with their ML library PyTorch [Paszke et al., 2019]. Similarly, Google's Vizier (by Golovin *et al.* [Golovin et al., 2017]) implements a suite of state-of-the-art approaches, e.g., SMAC [Hutter et al., 2011], and their own novel probabilistic optimization approach into an optimization framework used internally at Google.

3.2 Priors in BO

A few authors have proposed to incorporate user experience into BO in the form of priors. Swersky *et al.* [Swersky, 2017], for instance, propose a number of adaptations to BO approaches for ML, based on prior knowledge that they have built working on ML problems. These adaptations include a multi-task BO algorithm to mimic knowledge transference between similar problems and an early-stopping BO algorithm that detects configurations that are not worth fully evaluating in iterative models. Although all of the proposed adaptations are based on human prior experience, the work of Swersky does not allow users to input their own custom priors into the optimization process.

Oh et al. [2018] and Siivola et al. [2018] propose structural priors for high-dimensional problems. They assume that users always center the search space at regions they expect to be good and then develop BO approaches that favor configurations near the center. However, this is a rigid assumption about optimum locality, which does not allow users to freely specify their priors. Similarly, Shahriari *et al.* Shahriari et al. [2016] focus on unbounded search spaces. The priors in their work are not about good regions of the space, but rather a regularizer that penalizes configurations based on their distance to the user-defined search space. The priors are automatically derived from the search space and not provided by users.

Other tools support limited custom user priors in the search space definition. For instance, black-box optimization tools such as SMAC [Hutter et al., 2011] or iRace [López-Ibáñez et al., 2016] support simple hand-designed priors, e.g. log-transformations. However, these are not properly reflected in the predictive models and both cannot explicitly recover from bad priors. Thus, both approaches are also less flexible than BOPrO.

In a parallel work, the BO approach of Li et al. [2020] supports user-defined priors in the form of probability distributions like ours. Their approach proposes sampling a number of maximum points from the GP, via Thompson Sampling, and then choosing to explore the maximum that has the highest prior. Their approach does not directly combine the model and prior. We argue our method leverages better the information from the prior and model by combining both distributions into a pseudo-posterior before computing the acquisition function.

The work of Ramachandran et al. [2020] also supports priors in the form of probability distributions. Their work uses the probability integral transform to warp the search space, stretching regions where the prior has high probability, and shrinking others. Once again, compared to their approach, BOPrO is agnostic to the probabilistic model used and directly controls the balance between prior and model via the β hyperparameter. Additionally, BOPrO's probabilistic model is fitted independently from the prior, which ensures it is not biased by the prior, while their approach fits the model to a warped version of the space, transformed by the prior, making it difficult to recover from misleading priors.

The most related work to ours is, as already mentioned before, the TPE approach by Bergstra et al. [2011], which also allows to define limited hand-designed priors in the form of normal distributions or log-normal distributions. TPE also builds two pseudo-posterior distributions based on the priors and the data and optimizes an EI acquisition function by looking for points that are likely under the "good" pseudo-posterior and unlikely under the "bad" pseudo-posterior. TPE is provided as part of the HyperOpt tool [Bergstra et al., 2013].

BOPrO is inspired by TPE in that it defines "good" and "bad" probability distributions to guide optimization; however, BOPrO defines these distributions with a novel Bayesian approach that sets it apart from TPE in several aspects:

1. BOPrO is not tied to a specific predictive model; in our experiments we use GPs and RFs, which are often more sample-efficient than TPE.

2. BOPrO is more flexible with respect to how the prior is defined, the prior for BOPrO can be any probability distribution.

3. BOPrO gives more importance to the model as iterations progress, gradually forgetting the prior.

These differences make BOPrO more flexible and efficient (c.f. Section 5.2.3) than TPE, even when using the exact same prior.

3.3 Meta-learning for BO

Our work also relates to meta-learning for BO. Meta-learning seeks to learn how learning algorithms perform on different problems, in order to use this experience to choose which algorithms to apply to new problems [Hutter et al., 2018]. In other words, meta-learning builds a prior knowledge based on experience and applies this prior to improve performance of learning algorithms. Following this idea, meta-learning for BO assumes that BO is applied to many similar optimization problems in a sequence such that knowledge about the general problem structure can be exploited on future optimization problems.

Meta-learning can be combined with different steps of BO. For instance, one approach is to use meta-learning to warmstart BO during initial design. The core idea here is to initialize BO with a set of configurations that are known to perform well on similar previously-solved problems. Feurer et al. [2015b], for instance, define a set of meta-features to characterize each optimization problem and then consider two problems similar based on a distance metric between meta-feature vectors. The authors implement a total of 46 meta-features to define the optimization problem and then use two distance metrics to identify similar problems. The authors apply their meta-learned BO algorithm to optimize ML hyperparameters on a set of 57 benchmarks and show that it improves the performance of BO considerably.

Another approach is to incorporate meta-learning in the surrogate model. Schilling et al. [2016] achieve this by training a separate GP for each previous problem and a GP for the new problem and combining the predictions of all GPs when optimizing the new problem. The GPs are combined with a weighing parameter so that more weight is given to the GPs of similar problems. After the GPs are trained, prediction can be done linearly in the number of GPs. The authors evaluate their solution and show that it is more scalable and leads to better results than previous approaches in the literature.

In a later work, Wistuba et al. [2018] expand on their work [Schilling et al., 2016] by incorporating meta-learning in the acquisition function. The authors adopt a similar approach of training a separate GP for each problem, including the new problem. However, instead of looking at the prediction of these GPs directly, their approach looks at the predicted improvement of each GP for their respective problem and combines it with the expected improvement on the new problem. The authors once again show that their solution is more scalable and leads to better results, when compared to previous solutions.

At last, other approaches consist of trying to identify the similarity to previous problems on the fly and can often recover from uninformative meta-learning priors [Lindauer and Hutter, 2018]. Lindauer and Hutter [2018] propose warmstarting BO with a initial design chosen so that configurations that are too similar are avoided, reducing the overhead in the initial design. The authors then use this initial design to initialize a meta-learning BO model that fits one separate model for each optimization problem (previous and current). Their model uses a set of weights to control how much weight is given to previous models compared to the current model. These weights are optimized, so that as iterations progress, the new model receives more weight, effectively forgetting uninformative priors.

The key difference between BOPrO and these meta-learning approaches for BO is in our prior flexibility. Meta-learning approaches assume that a prior in the form of previously optimized problems. This type of prior is not always present, as users may wish to apply BO to a problem without any previous optimization experience. Further, users might have an insight on which configurations are likely to perform well from their experience in the field, but not data from previous executions to show this prior. Without this data, they are not able to input these priors with meta-learning. Conversely, BOPrO provides a flexible prior definition that allows for both priors based on previous data and priors based on previous user knowledge.

3.4 The Novelty of BOPrO

In this section, we briefly compare the features and contributions of BOPrO to the current state-of-the-art of prior-injection in BO. Namely, we identify four paramount properties for a prior-guided BO framework and show that only BOPrO provides all four properties in a single optimization framework. First, the framework must be flexible in the prior definition, otherwise, users may be unable to input the priors they have in BO (*Flexibility*). Second, the framework must be able to forget wrong or uninformative priors and still converge to good function values (*Forgetting*). Third, the framework must be efficient in the number of function evaluations it requires to converge (*Efficiency*). At last, it is desirable for the framework to properly combine the prior knowledge with BO's predictive model into a unified pseudo-posterior (*Posterior*). This last step is important as it provides a direct balance between prior and model and allows BO's acquisition function to properly quantify promising configurations according to the prior and model jointly.

Table 3.1 shows a comparison of BOPrO and previous state-of-the-art approaches according to these four properties. We note that standard BO approaches, such as Spearmint [Snoek et al., 2012], simply do not support prior-injection. In the table, we combine all previous BO solutions that do not support priors as a single entry. Likewise, all meta-learning approaches are based on a less flexible prior definition, thus, for simplicity, we also combine all meta-learning approaches into a single entry. We consider that these combined entries provide a specific property if at least one approach

Table 3.1. Comparison of BOPrO and previous state-of-the-art prior-guided optimization solutions.

Approach	**Flexibility**	**Forgetting**	**Efficiency**	Posterior
Standard BO	×	×	×	n/a
Meta-learning	×	✓	✓	✓
Swersky [2017]	×	×	✓	✓
SMAC	×	×	✓	×
iRace	×	×	✓	×
Li et al. [2020]	✓	✓	×	×
Ramachandran et al. [2020]	✓	×	✓	×
TPE	×	×	×	✓
BOPrO	✓	✓	✓	✓

inside the category provides the property. Conversely, we compare our work to all BO frameworks that support prior-injection in our related work, since those are the approaches that more closely relate to ours.

At last, we note that Table 3.1 summarizes the properties that we consider more important for prior-injection in BO, however, BOPrO also provides a number of desirable features for BO that most other approaches do not. Namely, BOPrO is model agnostic and can be implemented with any traditional BO surrogate model, such as GPs, RFs, TPE, and BNNs. BOPrO can be combined with cBO to support constrained search spaces (c.f. Section 5.2.4). At last, BOPrO provides a hyperparameter that explicitly controls how much weight is given to the prior vs the model, allowing users to specify how confident they are in their own priors. This hyperparameter is also treated such that regardless of the hyperparameter value, BOPrO will always recover from misleading priors.

Chapter 4

Bayesian Optimization with Priors

We propose a BO approach dubbed BOPrO that allows field experts to inject user prior knowledge into the optimization. BOPrO uses a Bayesian approach to combine a user-defined prior with a probabilistic model that captures the likelihood of the observed data $(x_i, y_i)_{i=1}^n$. BOPrO is independent of the probabilistic model being used; it can be freely combined with, e.g., Gaussian processes (GPs), random forests, or Bayesian NNs.

4.1 Priors

BOPrO allows users to inject prior knowledge into BO. This is done via a prior distribution that informs where in the input space $\mathcal{X}$ we expect to find good $f(\boldsymbol{x})$ values. A point is considered "good" if it leads to low function values in a minimization problem. We denote the prior distribution $P_g(\boldsymbol{x})$, where g denotes that this is a prior on good points and $\boldsymbol{x} \in \mathcal{X}$ is a given point. Examples of priors are shown in Figures 4.2 and 5.8.

Similarly, we define a prior on where in the input space we expect to have "bad" points. Although we could have a user-defined probability distribution $P_b(\boldsymbol{x})$, we aim to keep the decision-making load on users low and thus, for simplicity, only require the definition of $P_g(\boldsymbol{x})$ and compute $P_b(\boldsymbol{x}) = 1 - P_g(\boldsymbol{x})$.[1] $P_g(\boldsymbol{x})$ is normalized to $[0, 1]$ by min-max scaling before computing $P_b(\boldsymbol{x})$.

In practice, $\boldsymbol{x}$ contains several dimensions but it is difficult for human experts to provide a multivariate distribution $P_g(\boldsymbol{x})$. Users can easily specify, e.g., draw, a univariate or bivariate probability distribution for continuous dimensions, or provide a

[1]We note that for continuous spaces, $P_b(\boldsymbol{x})$ is not a probability distribution as it does not integrate to 1 and therefore is only a pseudo-prior. For discrete spaces, we normalize $P_b(\boldsymbol{x})$ so that it sums to 1 and therefore is a proper distribution and prior.

list of probabilities for each level of an ordinal or categorical variable. In BOPrO, users are free to define a complex multi-variate distribution, but we expect the standard use case to be that users only want to specify univariate distributions, implicitly assuming a prior that factors as

$$P_g(\boldsymbol{x}) = \prod_{i=1}^{N} P_g(x_i), \tag{4.1}$$

where N is the number of variables in $\mathcal{X}$, x_i is the i-th input dimension of $\mathcal{X}$, and g denotes that this is a prior on good points. While a non-factorized prior can of course be more powerful, throughout our experiments, we use this factorized prior to mimic what we expect standard users to be able to provide. In Section 5.3.5 we show that these factorized priors can in fact lead to similar BO performance as multivariate priors.

4.2 Model

Whereas the standard probabilistic model in BO, e.g., a GP, quantifies $p(y|\boldsymbol{x})$ directly, that model is hard to combine with the prior $P_g(\boldsymbol{x})$. We therefore introduce a method to translate the standard probabilistic model $p(y|\boldsymbol{x})$ into a model that is easier to combine with this prior. Similar to the TPE work described in Sec. 2.3, our generative model combines $p(\boldsymbol{x}|y)$ and $p(y)$ instead of directly modeling $p(y|\boldsymbol{x})$.

The computation we perform for this translation is to quantify the probability that a given input $\boldsymbol{x}$ is "good" under our standard probabilistic model $p(y|\boldsymbol{x})$. As in TPE, we define configurations as "good" if their observed y-value is below a certain quantile γ of the observed function values (so that $p(y < f_\gamma) = \gamma$). We in addition exploit the fact that our standard probabilistic model $p(y|\boldsymbol{x})$ has a Gaussian form, and under this Gaussian prediction we can compute the probability $\mathcal{M}_g(\boldsymbol{x})$ of the function value lying below a certain quantile using the standard closed-form formula for PI [Kushner, 1964]:

$$\begin{aligned}\mathcal{M}_g(\boldsymbol{x}) &= p(f(\boldsymbol{x}) < f_\gamma|\boldsymbol{x}, (\boldsymbol{x}_i, y_i)_{i=1}^t) \qquad (4.2)\\ &= \Phi\left(\frac{f_\gamma - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}}\right),\end{aligned}$$

where $(\boldsymbol{x}_i, y_i)_{i=1}^t$ are the evaluated configurations, $\mu_{\boldsymbol{x}}$ and $\sigma_{\boldsymbol{x}}$ are the mean and standard deviation of the probabilistic model at $\boldsymbol{x}$, and Φ is the standard normal CDF, see Figure 4.1. Note that there are two probabilistic models here:

1. The standard probabilistic model of BO, with a prior over functions $p(f)$, updated by data $(\boldsymbol{x}_i, y_i)_{i=1}^t$ to yield a pseudo-posterior over functions $p(f|(\boldsymbol{x}_i, y_i)_{i=1}^t)$,

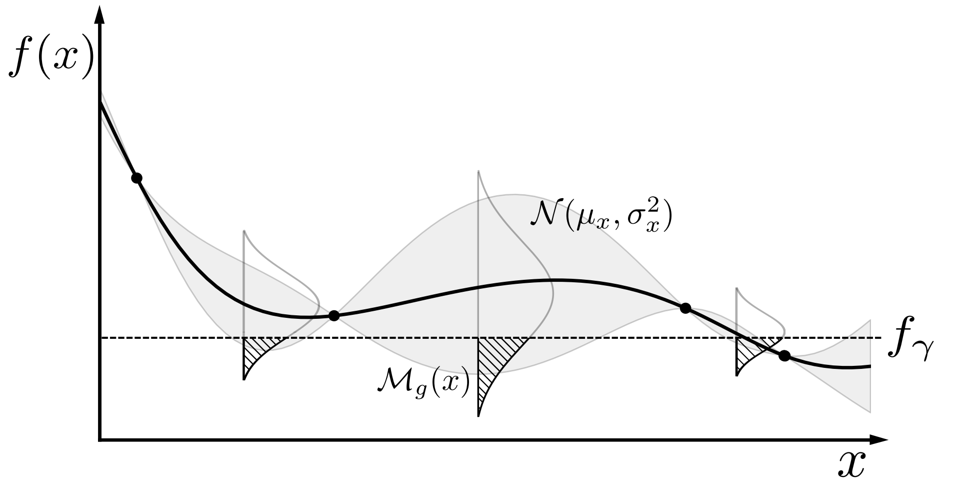


Figure 4.1. Our model is composed by a probabilistic model and the probability of improving over the threshold f_γ, i.e., right tail of the Gaussian. The black curve is the probabilistic model's mean and the shaded area is the model's variance.

allowing us to quantify the probability $\mathcal{M}_g(\boldsymbol{x}) = p(f(x) < f_\gamma | \boldsymbol{x}, (\boldsymbol{x}_i, y_i)_{i=1}^t)$ in Eq. (4.2).

2. The TPE-like generative model that combines $p(y)$ and $p(\boldsymbol{x}|y)$ instead of directly modelling $p(y|\boldsymbol{x})$.

Eq. (4.2) bridges these two models by using the probability of improvement from BO's standard model as the probability $\mathcal{M}_g(\boldsymbol{x})$ in TPE's model. With this same formulation, we also define a probability $\mathcal{M}_b(\boldsymbol{x})$ of $\boldsymbol{x}$ being bad as $\mathcal{M}_b(\boldsymbol{x}) = 1 - \mathcal{M}_g(\boldsymbol{x})$. This is equivalent to computing the probability of the function value lying above the quantile f_γ.

4.3 Pseudo-posterior

BOPrO combines the prior in Eq. (4.1) and the model in Eq. (4.2) into a pseudo-posterior on "good" points. This pseudo-posterior represents the updated beliefs on where we can find good points, based on the prior and data that has been observed. The pseudo-posterior on good points is computed as the product of the prior and the model:

$$b(\boldsymbol{x}) \propto P_b(\boldsymbol{x})\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}, \tag{4.3}$$

where t is the current optimization iteration, β is an optimization hyperparameter, and $P_g(\boldsymbol{x})$ and $\mathcal{M}_g(\boldsymbol{x})$ are defined in Eq. (4.1) and Eq. (4.2) respectively. We note that the pseudo-posterior computed in Eq. (4.3) is not normalized, but this suffices for BOPrO as the normalization constant cancels out (c.f. Section 4.5). Since $g(\boldsymbol{x})$ is not normalized and we include the exponent t/β in Eq. (4.3), we refer to $g(\boldsymbol{x})$ as a pseudo-posterior.

The t/β fraction in Eq. (4.3) controls how much weight is given to the model. As the optimization progresses, more weight is given to the model over the prior. Intuitively, this means that we put more emphasis on the predictions of the model as it observes more data and becomes more accurate. We do this under the assumption that the model will eventually be better than the user at predicting where to find good points. In addition, this also allows to recover from "bad" priors as we show in Section 5.3.2; similar to, and inspired by Bayesian models, the data ultimately washes out the prior. The β hyperparameter defines the balance between prior and model, with higher β values giving more importance to the prior.

We note that, since our pseudo-posterior is not normalized, computing it directly as in Equation (4.3) can lead to numerical issues. Namely, the pseudo-posterior can reach extremely low values if the prior and model probabilities are low, especially as the t/β exponent grows. To prevent this, in practice, BOPrO uses the logarithm of the pseudo-posterior instead:

$$\log(g(\boldsymbol{x})) \propto \log(P_g(\boldsymbol{x})) + \tfrac{t}{\beta} \cdot \log(\mathcal{M}_g(\boldsymbol{x})). \tag{4.4}$$

Once again, we also define a pseudo-posterior distribution on bad $\boldsymbol{x}$, $b(\boldsymbol{x})$, with a similar approach, only using $P_b(\boldsymbol{x})$ and $\mathcal{M}_b(\boldsymbol{x})$ instead. We then use these quantities to compute a density model $p(\boldsymbol{x}|y)$ as follows:

$$p(\boldsymbol{x}|y) \propto \begin{cases} g(\boldsymbol{x}) & \text{if } \ f(\boldsymbol{x}) < f_\gamma \\ b(\boldsymbol{x}) & \text{if } \ f(\boldsymbol{x}) \geq f_\gamma. \end{cases} \tag{4.5}$$

4.4 Model and Pseudo-posterior Visualization

This section we illustrate the prior $P_g(\boldsymbol{x})$, the model $M_g(\boldsymbol{x})$ and the pseudo-posterior $g(\boldsymbol{x})$ for a 1-dimensional Branin function and their evolution over the optimization iterations. We define the 1D Branin function by setting the second dimension of the

Branin function to the global optimum $x_2 = 2.275$ and optimizing the first dimension. We perform an initial design of the usual budget of $D + 1 = 2$ random points and use a GP as predictive model. We use a Beta distribution prior $P_g(\boldsymbol{x}) = \mathcal{B}(3, 3)$ which resembles a truncated Gaussian centered close to the global optimum and compute the model $\mathcal{M}_g(\boldsymbol{x})$ and pseudo-posterior $g(\boldsymbol{x})$ following Eq. (4.2) and (4.3) respectively. Figure 4.2 shows the evolution at different stages.

In Figure 4.2a, after the initialization phase, the pseudo-posterior in the top row is almost equal to the prior: in the absence of much data, BOPrO relies heavily on the prior. In 4.2b, the pseudo-posterior is high near the global minimum, around $x = \pi$, where both the prior and the model agree there are good points. The optimum has almost been found at this stage.[2]

After 10 BO iterations in 4.2c, there are three regions with high pseudo-posterior which lead to future exploration steps on these regions as it can be seen in 4.2d on the right and left of the global optimum in light green. In the middle region of 4.2c, the pseudo-posterior vanishes because the model is certain that sampling from that region will not bring any improvement. After 20 iterations, the pseudo-posterior vanishes where the model is certain there will be no improvement but is high wherever there is uncertainty in the GP. Note that the influence of the prior after 20 iterations is weaker, because of t/β in Eq. (4.4). We also studied in preliminary experiments the influence of choosing different β on the 1D Branin and observed that the performance of BOPrO is fairly robust with respect to β.

4.5 Acquisition Function

We adopt the EI formulation used in [Bergstra et al., 2011] by replacing their Adaptive Parzen Estimators with our computation of the pseudo-posterior probabilities in Eq. (4.3). Namely, we compute EI as:

$$EI_{f_\gamma}(\boldsymbol{x}) := \int_{-\infty}^{\infty} \max(f_\gamma - y, 0) p(y|\boldsymbol{x}) dy = \int_{-\infty}^{f_\gamma} (f_\gamma - y) \frac{p(\boldsymbol{x}|y)p(y)}{p(\boldsymbol{x})} dy. \quad (4.6)$$

As defined in 4.2, $p(y < f_\gamma) = \gamma$ and γ is a quantile of the observed objective values $\{y^{(i)}\}$. Then $p(\boldsymbol{x}) = \int_{\mathbb{R}} p(\boldsymbol{x}|y)p(y)dy = \gamma g(\boldsymbol{x}) + (1 - \gamma) g(\boldsymbol{x})$, where $g(\boldsymbol{x})$ and

[2] Please note that the scale of the Model and the log scale of the Pseudo-posterior don't allow a proper visualization that explains all the bumps in the Pseudo-posterior. As an example in 4.2b around 1 we see a bump that is not supported by a bump in the model. Indeed, the model has a small bump that is not visible in the plot and the log scale magnifies that in a visible bump in the pseudo-posterior. On the other hand, the bumps that are visible in the model are plateaued in the pseudo-posterior so that different heights correspond to a pseudo-posterior of 1.

$b(\boldsymbol{x})$ are the pseudo-posterior introduced in Section 4.3. Therefore

$$
\begin{aligned}
EI_{f_\gamma}(\boldsymbol{x}) = \int_{-\infty}^{f_\gamma} (f_\gamma - y) p(\boldsymbol{x}|y) p(y) dy = \\
g(\boldsymbol{x}) \int_{-\infty}^{f_\gamma} (f_\gamma - y) p(y) dy = \\
\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\infty}^{f_\gamma} y p(y) dy, \quad (4.7)
\end{aligned}
$$

so that finally

$$
EI_{f_\gamma}(\boldsymbol{x}) = \frac{\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\infty}^{f_\gamma} y p(y) dy}{\gamma g(\boldsymbol{x}) + (1-\gamma) b(\boldsymbol{x})} \propto \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})} (1-\gamma) \right)^{-1}. \quad (4.8)
$$

Eq. (4.8) shows that to maximize improvement we would like points $\boldsymbol{x}$ with high probability under $g(\boldsymbol{x})$ and low probability under $b(\boldsymbol{x})$, i.e., minimizing the ratio $b(\boldsymbol{x})/g(\boldsymbol{x})$. We note that the point that maximizes the ratio for our unnormalized pseudo-posterior will be the same that maximizes the ratio for normalized pseudo-posterior and, thus, we do not need to compute the normalized pseudo-posterior.

4.6 Formal Analysis

The dynamics of the BOPrO algorithm can be understood in terms of the following proposition:

Proposition 1 *Given f_γ, $P_g(\boldsymbol{x})$, $P_b(\boldsymbol{x})$, $\mathcal{M}_g(\boldsymbol{x})$, $\mathcal{M}_b(\boldsymbol{x})$, $g(\boldsymbol{x})$, $b(\boldsymbol{x})$, $p(\boldsymbol{x}|y)$, and β as above, then*

$$
\lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} EI_{f_\gamma}(\boldsymbol{x}) = \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{M}_g(\boldsymbol{x}),
$$

where EI_{f_γ} is the Expected Improvement acquisition function as defined in Eq. (4.6) *and $\mathcal{M}_g(\boldsymbol{x})$ is as defined in Eq.* (4.2).

Which can be derived as shown.

$$\lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} EI_{f_\gamma}(\boldsymbol{x}) \tag{4.9}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \int_{-\inf}^{f_\gamma} (f_\gamma - y)p(\boldsymbol{x}|y)p(y)dy \tag{4.10}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} (f_\gamma - y)p(y)dy \tag{4.11}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy \right) \tag{4.12}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \frac{\gamma f_\gamma g(\boldsymbol{x}) - g(\boldsymbol{x}) \int_{-\inf}^{f_\gamma} yp(y)dy}{\gamma g(\boldsymbol{x}) + (1-\gamma)b(\boldsymbol{x})} \tag{4.13}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})}(1-\gamma) \right)^{-1} \tag{4.14}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{b(\boldsymbol{x})}{g(\boldsymbol{x})}(1-\gamma) \right)^{-\frac{1}{t}} \tag{4.15}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\gamma + \frac{P_b(\boldsymbol{x})\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{P_g(\boldsymbol{x})\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}}(1-\gamma) \right)^{-\frac{1}{t}} \tag{4.16}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{P_b(\boldsymbol{x})\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{P_g(\boldsymbol{x})\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}}(1-\gamma) \right)^{-\frac{1}{t}} \tag{4.17}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{P_b(\boldsymbol{x})}{P_g(\boldsymbol{x})} \right)^{-\frac{1}{t}} \left(\frac{\mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}}{\mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}} \right)^{-\frac{1}{t}} (1-\gamma)^{-\frac{1}{t}} \tag{4.18}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{\mathcal{M}_b(\boldsymbol{x})}{\mathcal{M}_g(\boldsymbol{x})} \right)^{-\frac{1}{\beta}} \tag{4.19}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{1-\mathcal{M}_g(\boldsymbol{x})}{\mathcal{M}_g(\boldsymbol{x})} \right)^{-\frac{1}{\beta}} \tag{4.20}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \left(\frac{1}{\mathcal{M}_g(\boldsymbol{x})} - 1 \right)^{-\frac{1}{\beta}} \tag{4.21}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} (\mathcal{M}_g(\boldsymbol{x}))^{\frac{1}{\beta}} \tag{4.22}$$

$$= \lim_{t\to\infty} \arg\max_{\boldsymbol{x}\in\mathcal{X}} \mathcal{M}_g(\boldsymbol{x}) \tag{4.23}$$

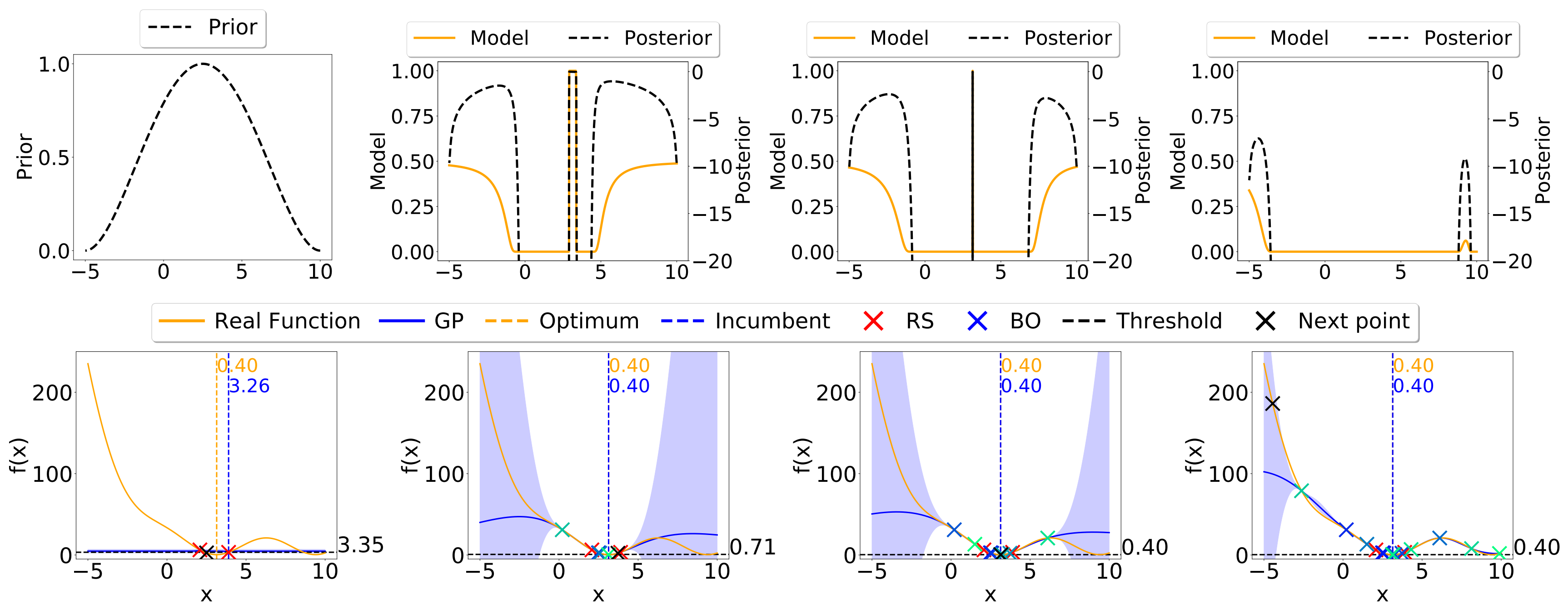
Proposition 1 shows that in early BO iterations the prior for the optimum will have a predominant role, but in later BO iterations the model will grow more important and the prior washes out. If BOPrO is run long enough, it will trust *only* the probabilistic model $\mathcal{M}_g(\boldsymbol{x})$ informed by the data.

4.7 Putting It All Together

Algorithm 2 shows the BOPrO algorithm, based on the components defined in the previous sections. In line 3, BOPrO starts with a design of experiments (DoE) phase, where it randomly samples a number of points from the user-defined prior $P_g(\boldsymbol{x})$. After initialization, the BO loop starts at line 4. In each loop iteration, BOPrO fits the probabilistic model on the previously evaluated points (lines 5 and 6) and computes the pseudo-posterior $g(\boldsymbol{x})$ and $b(\boldsymbol{x})$ (lines 7 and 8 respectively). The EI acquisition function is computed next, using the pseudo-posterior, and the point that maximizes EI is selected as the next point to evaluate at line 9. The black-box function evaluation is performed at line 10. This BO loop is repeated for a pre-defined number of iterations, according to the user-defined budget B.

Algorithm 2 BOPrO Algorithm. $\boldsymbol{D}$ keeps track of all evaluated configurations and their function value: $(\boldsymbol{x}_i, y_i)_{i=1}^t$

```
1: Input: Input space $\mathcal{X}$, user-defined prior distributions $P_g(\boldsymbol{x})$ and $P_b(\boldsymbol{x})$, quantile
   $\gamma$ and BO budget $B$.
2: Output: Optimized point $\boldsymbol{x}_{inc}$.
3: $\boldsymbol{D} \leftarrow Initialize(\mathcal{X})$
4: for $t = 1$ to $B$ do
5:     $\mathcal{M}_g(\boldsymbol{x}) \leftarrow fit\_model\_good(\boldsymbol{D})$
6:     $\mathcal{M}_b(\boldsymbol{x}) \leftarrow fit\_model\_bad(\boldsymbol{D})$
7:     $g(\boldsymbol{x}) \leftarrow P_g(\boldsymbol{x}) \cdot \mathcal{M}_g(\boldsymbol{x})^{\frac{t}{\beta}}$
8:     $b(\boldsymbol{x}) \leftarrow P_b(\boldsymbol{x}) \cdot \mathcal{M}_b(\boldsymbol{x})^{\frac{t}{\beta}}$
9:     $\boldsymbol{x}_i \in \arg\max_{\boldsymbol{x} \in \mathcal{X}} EI_{f_\gamma}(\boldsymbol{x})$
10:    $y_i \leftarrow f(\boldsymbol{x}_i)$
11:    $\boldsymbol{D} = \boldsymbol{D} \cup (\boldsymbol{x}_i, y_i)$
12: end for
13: $\boldsymbol{x}_{inc} \leftarrow ComputeBest(\boldsymbol{D})$
14: return $\boldsymbol{x}_{inc}$
```



(a) 0 BO iterations (b) 5 BO iterations (c) 10 BO iterations (d) 20 BO iterations

Figure 4.2. Breakdown of the prior $P_g(\boldsymbol{x}) = \mathcal{B}(3,3)$, the model $M_g(\boldsymbol{x})$ and the pseudo-posterior $g(\boldsymbol{x})$ (top row) for the 1-dimensional Branin function (bottom row) and their evolution over the optimization iterations. Two random points from the prior are sampled to initialize the GP model (bottom row) before starting BOPrO. The bottom row shows the samples explored during optimization. The blue/green x's denote BOPrO samples, with green samples denoting later iterations. The blue line and shaded area show the mean and uncertainty of the GP model.

Chapter 5

Experiments

In this section, we perform an empirical evaluation of BOPrO. We first present our experimental setup for BOPrO, then present our experiments separated in two sections. First, we show experiments comparing BOPrO to other state-of-the-art baselines, with and without priors. We compare BOPrO and the baselines on a suite of synthetic benchmarks and a real-world application. Next, we perform a suite of ablation studies on BOPrO, validating the assumptions and goals we set for our solution.

5.1 Experimental Setup

Before presenting our experiments, we will describe the experimental setup. All experiments use the setup described in this section, unless stated otherwise.

5.1.1 Implementation.

We use the BOPrO implementation provided in HyperMapper[1] [Nardi et al., 2018]. We use GPs for all experiments. We use GPy's [GPy, 2012] GP implementation with the Matérn5/2 kernel. We use different lengthscales for each input dimension, learned via Automatic Relevance Determination (ARD) [Neal, 2012]. We set the model weight hyperparameter to $\beta = 10$ and the model quantile to $\gamma = 0.05$ (See Sections 5.3.6 and 5.3.7). Before starting the main BO loop, all methods are initialized by randomly sampling points. We use the default for which baseline if there is one. For BOPrO we sample $D + 1$ points, randomly sampled from the user prior. We also implement interleaving which randomly samples a point to explore during BO with a 10% chance.

[1]https://github.com/luinardi/hypermapper

We optimize our EI acquisition function using a combination of a multi-start local search and CMA-ES Hansen and Ostermeier [1996]. Our multi-start local search is similar to the one used in SMAC Hutter et al. [2011]. Namely, we start local searches on the 10 best points evaluated in previous BO iterations, on the 10 best performing points from a set of 10,000 random samples, on the 10 best performing points from 10,000 random samples drawn from the prior, and on the mode of the prior. To compute the neighbors of each of these 31 total points, we normalize the range of each parameter to $[0, 1]$ and randomly sample four neighbors from a truncated Gaussian centered at the original value and with standard deviation $\sigma = 0.1$. For CMA-ES, we use the public implementation of pycma Hansen et al. [2019]. We run pycma with two starting points, one at the incumbent and one at the mode of the prior. For both initializations we set $\sigma_0 = 0.2$. We only use CMA-ES for our continuous search space benchmarks.

5.1.2 Benchmarks.

We use a suite of four synthetic benchmarks for the majority of our experiments. Our synthetic benchmarks are comprised of a mathematical function and four synthetically generated AutoML benchmarks, from the Profet framework [Klein et al., 2019]. All Profet benchmarks are generated by a generative model built using performance data on OpenML or UCI datasets. We use emukit's public implementation for these benchmarks [Paleyes et al., 2019].

Branin is a well-known synthetic benchmark for optimization problems [Dixon, 1978]. The Branin function has two input dimensions and three global minima. The Branin function is defined as:

$$f(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)cos(x_1) + s \tag{5.1}$$

where x_1 and x_2 are the function's input parameters and a, b, c, r, s, and t are usually set to $a = 1$, $b = 5.1/(4\pi 2)$, $c = 5/\pi$, $r = 6$, $s = 10$, and $t = 1/(8\pi)$.

SVM is a hyperparameter-optimization benchmark in 2D based on Profet. This benchmark is generated by a generative meta-model built using a set of SVM classification models trained on 16 OpenML tasks. The benchmark has two input parameters, corresponding to SVM hyperparameters.

FC-Net is a hyperparameter and architecture optimization benchmark in 6D based on Profet. The FC-Net benchmark is generated by a generative meta-model built using a set of feed-forward neural networks trained on the same 16 OpenML tasks as the SVM benchmark. The benchmark has six input parameters corresponding to network hyperparameters.

Table 5.1. Search spaces for our synthetic benchmarks. For the Profet benchmarks, we report the original ranges and whether or not a log scale was used.

Benchmark	**Parameter name**	**Parameter values**	**Log scale**
Branin	x_1	$[-5, 10]$	-
	x_2	$[0, 15]$	-
SVM	C	$[e^{-10}, e^{10}]$	✓
	γ	$[e^{-10}, e^{10}]$	✓
FCNet	learning rate	$[10^{-6}, 10^{-1}]$	✓
	batch size	$[8, 128]$	✓
	units layer 1	$[16, 512]$	✓
	units layer 2	$[16, 512]$	✓
	dropout rate l1	$[0.0, 0.99]$	-
	dropout rate l2	$[0.0, 0.99]$	-
XGBoost	learning rate	$[10^{-6}, 10^{-1}]$	✓
	gamma	$[0, 2]$	-
	L1 regularization	$[10^{-5}, 10^{3}]$	✓
	L2 regularization	$[10^{-5}, 10^{3}]$	✓
	number of estimators	$[10, 500]$	-
	subsampling	$[0.1, 1]$	-
	maximum depth	$[1, 15]$	-
	minimum child weight	$[0, 20]$	-

XGBoost is a hyperparameter-optimization benchmark in 8D based on Profet. The XGBoost benchmark is generated by a generative meta-model built using a set of XGBoost regression models in 11 UCI datasets. The benchmark has eight input parameters, corresponding to XGBoost hyperparameters.

The search spaces for each benchmark are summarized in Table 5.1. For the Profet benchmarks, we report the original ranges and whether or not a log scale was used. However, in practice, Profet's generative model transforms the range of all hyperparameters to a linear $[0, 1]$ range.

5.1.3 Priors

We build two synthetic priors in a controlled way for our experiments. We emphasize that in practice, manual priors would be based on the domain experts' expertise on their applications; here, we only use artificial priors to guarantee that our prior is not biased by our own expertise for the benchmarks we used. In practice, users will manually define these priors like in our real-world experiments (Section 5.2.4). We perform a more in-depth analysis of the effects of different prior strengths in Section 5.3.3.

Our synthetic priors take the form of Gaussian distributions centered near the

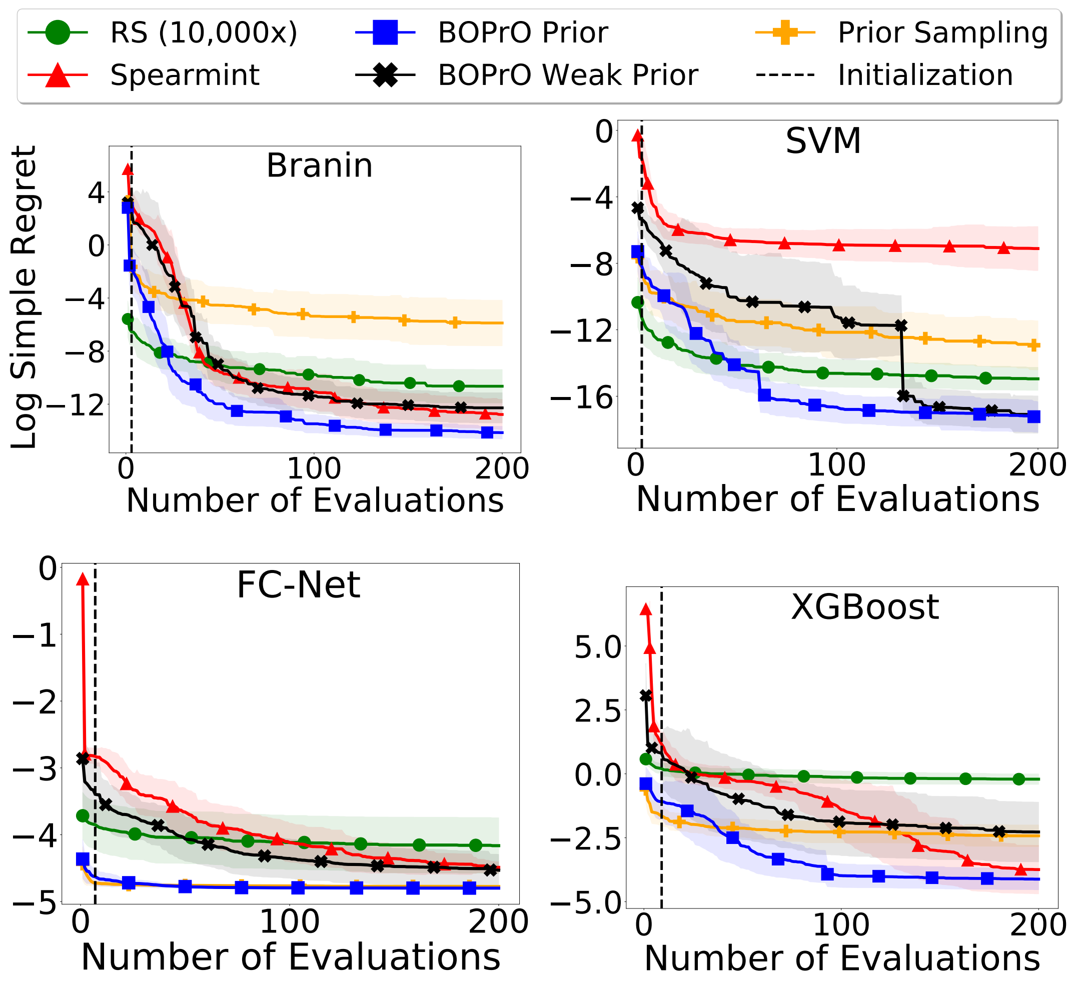


Figure 5.1. Log regret comparison of BOPrO with and without priors, random sampling, and Spearmint. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. All methods were initialized with $N + 1$ random samples, where N is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 100d iterations, capped at 200.

optimum. For each input $x \in \mathcal{X}$, we inject a prior of the form $\mathcal{N}(\mu_x, \sigma_x^2)$, where μ_x is sampled from a Gaussian centered at the optimum value x_{opt}[2] for that parameter $\mu_x \sim \mathcal{N}(x_{opt}, \sigma_x^2)$, and σ_x is a hyperparameter of our experimental setup determining the prior's strength. For each run of BOPrO, we sample new μ_x's. This setup provides us with a synthetic prior that is close to the optimum, but not exactly centered at it, and allows us to control the strength of the prior by σ_x. We use two prior strengths in our experiments: a strong prior, computed with $\sigma_x = 0.01$, and a weak prior, computed

[2]If the optimum for a benchmark is not known, we approximate it using the best value found during previous BO experiments.

with $\sigma_x = 0.1$.

5.2 Performance Comparison

We first compare BOPrO's optimization performance to several state-of-the-art baselines. We compare BOPrO to state-of-the-art approaches that do not support users priors, showing that BOPrO can leverage the user priors and achieve new state-of-the-art performance, and to other state-of-the-art approaches that support user priors, showing that BOPrO can better combine user priors and BO and achieve better results.

5.2.1 Performance on Synthetic Benchmarks

We first evaluate BOPrO's performance on the synthetic benchmarks, comparing to several baselines. Namely, we compare BOPrO with and without priors (both weak and strong) to Spearmint, randomly sampling points from the prior, and 10,000x random sampling (RS). 10,000x RS is a uniform random sampling approach (i.e. no priors) with 10,000x the budget of the other optimizers, i.e., for each sample evaluated by BO or prior sampling, 10,000x RS evaluates 10,000 samples. Spearmint is a well-adopted BO approach using GPs and the EI acquisition function. In contrast to BOPrO, Spearmint always uses 2 uniform random samples for initialization.

Figure 5.1 shows this comparison on our four synthetic benchmarks. The methods are compared using the log simple regret on five runs (mean and std error reported) on the four synthetic benchmarks. The log simple regret is defined as the natural logarithm of the distance between the function value at the best configuration found by the optimization method (i.e. the incumbent) and the optimum of the black-box function.

Figure 5.1 shows that BOPrO with the strong prior (dubbed BOPrO Prior) beats 10,000x Random Sampling on all benchmarks while this is not always the case for other optimization methods, including Spearmint. BOPrO Prior performs better than BOPrO with a weak prior on all benchmarks as expected. It also either outperforms or matches the performance of sampling from the prior; this is expected because prior sampling cannot recover from a non-ideal prior. The two methods are identical up to the initialization phase because they both sample from the same prior in that phase.

BOPrO with a strong prior is also more sample efficient and finds better or similar results to Spearmint on all benchmarks. Importantly, in all our experiments, BOPrO with a good prior consistently shows tremendous speedups in the early phases of the

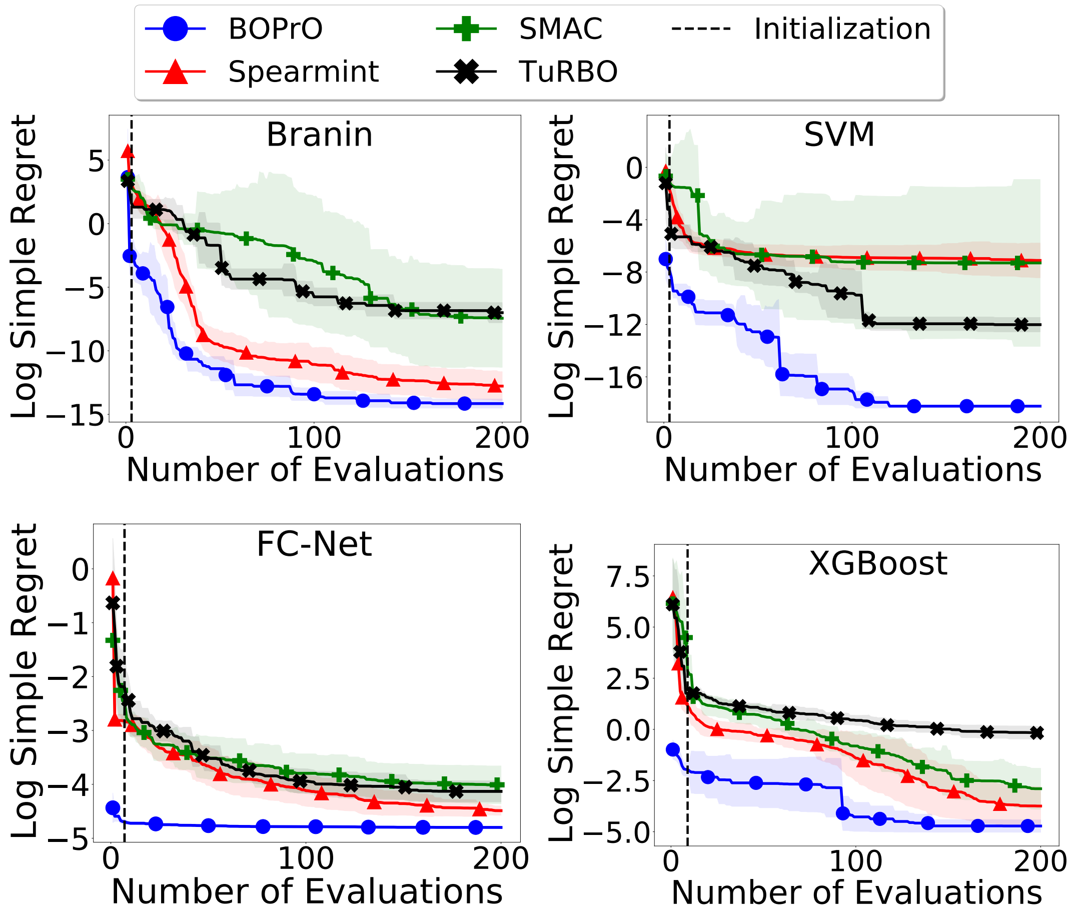


Figure 5.2. Log regret comparison of BOPrO, SMAC, TuRBO, and Spearmint. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. All methods were initialized with $N + 1$ random samples, where N is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 100d iterations, capped at 200.

optimization process, requiring on average only 15 iterations to reach the performance that Spearmint reaches after 100 iterations (6.67× faster).

5.2.2 Comparison to Baselines without Priors

We next extend our performance comparison to other state-of-the-art optimizers without priors. Namely, we compare BOPrO to Spearmint, SMAC, and TuRBO using log regret on our four synthetic benchmarks. SMAC is a state-of-the-art optimizer that uses RFs rather than GPs and the EI acquisition function [Hutter et al., 2011], while TuRBO uses GPs and EI, but employs a novel mix of local and global optimization for improved performance. For TuRBO, we also have to choose the number of trust regions that

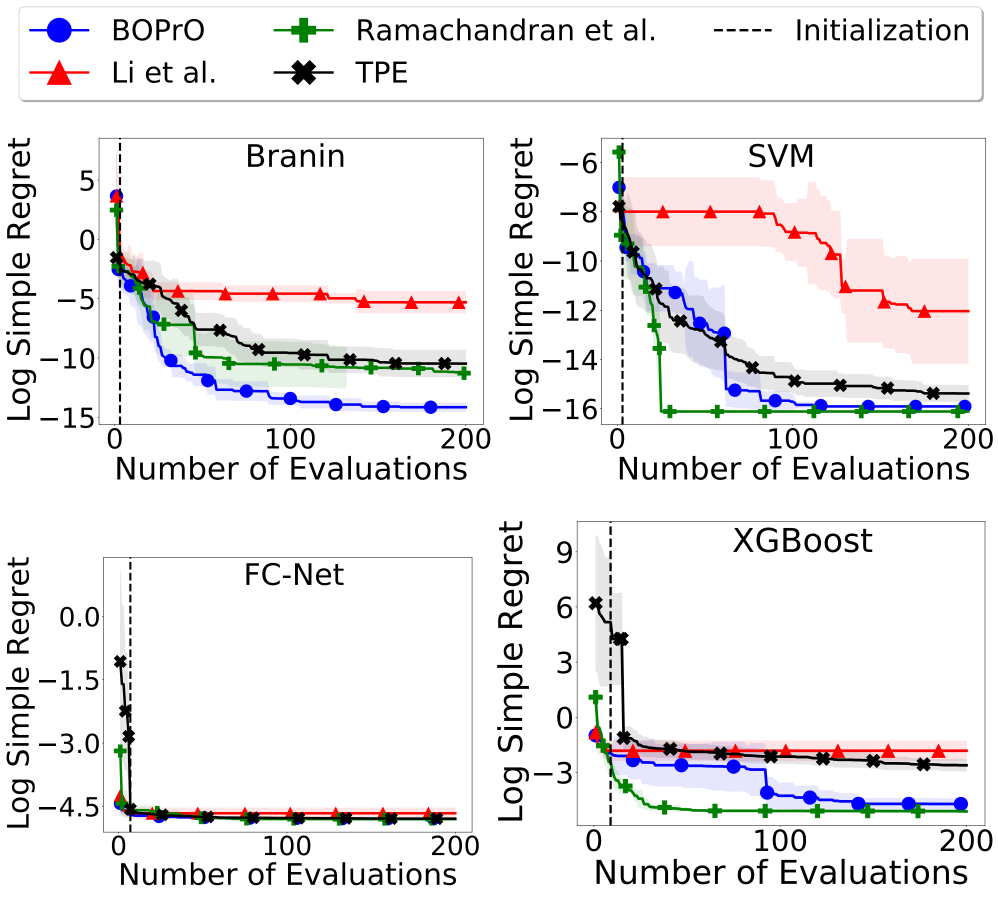


Figure 5.3. Log regret comparison of BOPrO, TPE, and the approaches of Li *et al.* and Ramachandran *et al.* , all with a strong prior. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. BOPrO was initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 200 iterations.

TuRBO will use during optimization (c.f. Section 3), we experiment with different values for this hyperparameter and ultimately use two trust regions for this comparison, which lead to the best performance. SMAC and TuRBO are initialized with $D + 1$ uniform random samples.

Figure 5.2 shows the comparison. BOPrO still achieves better or similar performance to all baselines on all benchmarks. Notably, BOPrO outperforms SMAC and TuRBO considerably on all benchmarks and, as before, achieves either better or similar performance to Spearmint on all benchmarks. Spearmint outperforms SMAC and TuRBO on all benchmarks except for SVM, where TuRBO achieves the best performance out of the no-prior baselines.

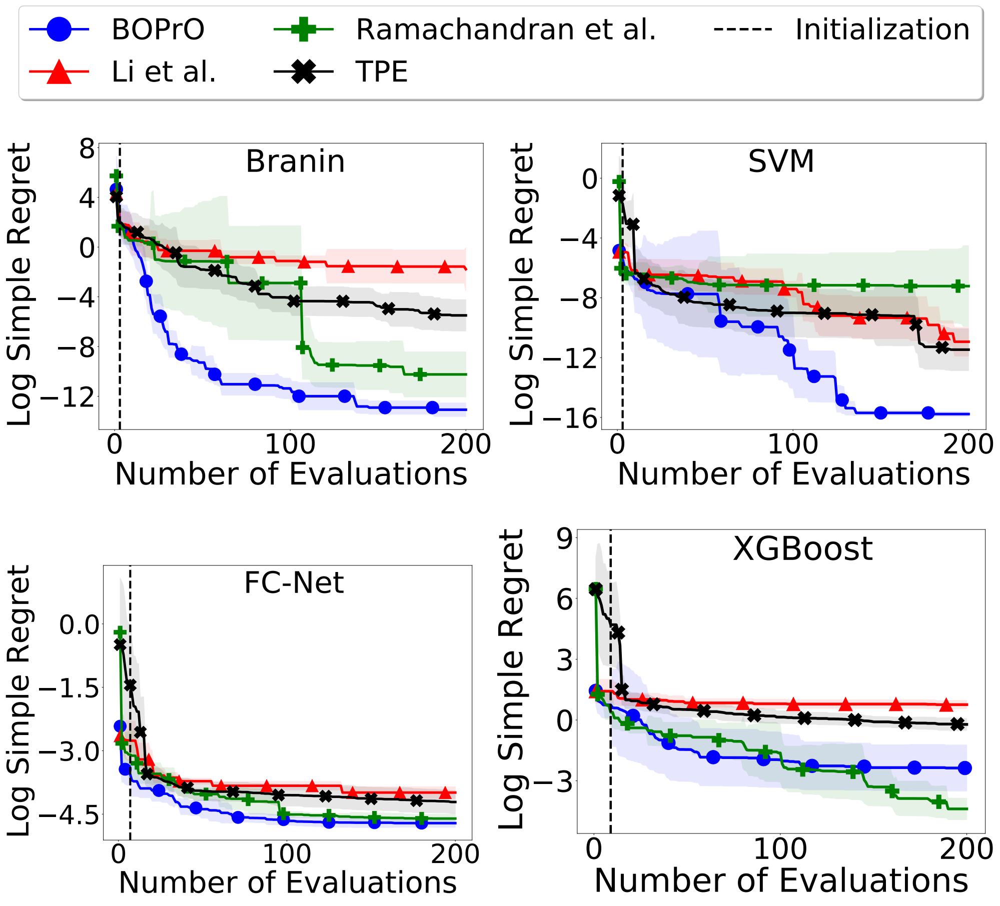


Figure 5.4. Log regret comparison of BOPrO, TPE, and the approaches of Li *et al.* and Ramachandran *et al.* , all with a weak prior. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. BOPrO was initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 200 iterations.

5.2.3 Comparison to Baselines with Priors

We compare BOPrO to other BO approaches that support user priors. Namely, we compare BOPrO to the approach of Li et al. [2020], Ramachandran et al. [2020], and TPE [Bergstra et al., 2013]. All three approaches are detailed in Section 3. We compare the four methods using log regret on our four synthetic benchmarks and compare them under three different prior strengths. We inject a strong prior, a weak (but accurate) prior, and a misleading prior and evaluate the performance of each approach. The strong and weak priors are as defined in our experimental setup (Section 5.1). For our misleading prior, we use a Gaussian centered at the worst point out of 10,000,000D uniform random samples. Namely, for each parameter, we inject a prior of the form

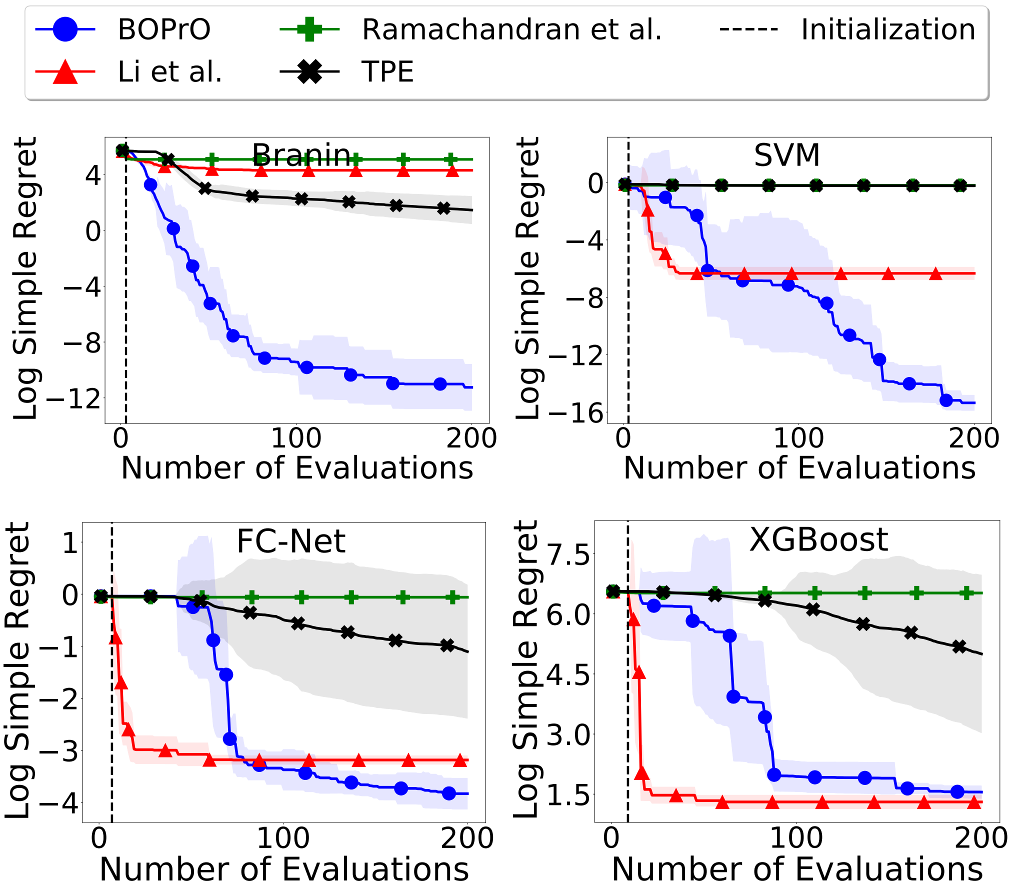


Figure 5.5. Log regret comparison of BOPrO, TPE, and the approaches of Li *et al.* and Ramachandran *et al.* , all with a misleading prior. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. BOPrO was initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 200 iterations.

$\mathcal{N}(x_w, \sigma_w^2)$, where x_w is the value of the parameter at the point with highest function value out of 10,000,000D uniform random samples and $\sigma_w = 0.01$.

Figure 5.3 shows the four methods with the strong prior. BOPrO achieves superior or similar final performance on all four benchmarks. Compared to TPE and the approach of Li *et al.* , BOPrO achieves better performance in three out of four benchmarks and equivalent performance on the final benchmark. Compared to the approach of Ramachandran *et al.* , BOPrO performs better on the Branin benchmark and achieves similar final performance on the other three. We note, however, that the approach of Ramachandran *et al.* converges to the final performance faster than BOPrO on the SVM and XGBoost benchmarks.

Figure 5.4 shows the four methods with the weak prior. BOPrO once again

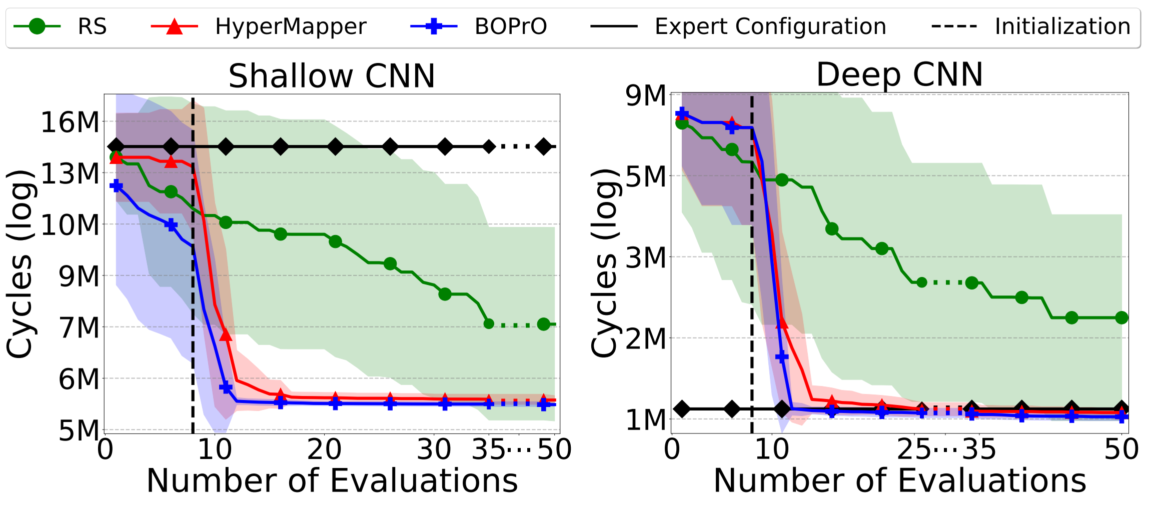


Figure 5.6. Log regret comparison of random sampling, HyperMapper, BOPrO, and a human expert configuration on the CNN benchmarks. The line and shaded regions show the mean and standard deviation of the log simple regret after 3 runs. BOPrO and HyperMapper were initialized with $N + 1$ random samples, where N is the number of input dimensions, indicated by the vertical dashed line.

outperforms TPE and the approach of Li *et al.* , this time on all four synthetic benchmarks. Compared to the approach of Ramachandran *et al.* , BOPrO now performs better on two of the synthetic benchmarks (Branin and SVM); has similar performance, but converges faster on the FCNet benchmark; and performs worse on the XGBoost benchmark.

At last, Figure 5.4 shows the four methods with the misleading prior. BOPrO is the only method that can consistently recover from the misleading prior and converge to good solutions. In particular, BOPrO vastly outperforms the TPE and the method of Ramachandran *et al.* on all four benchmarks. BOPrO also outperforms the approach of Li *et al.* on three out of four benchmarks, and achieves similar performance on the fourth (XGBoost), although it also takes longer to converge to the final solution.

Considering the four benchmarks and three prior strengths, BOPrO performs better than TPE in 11 out of 12 benchmarks and achieves the same performance on the last (FCNet with a strong prior). BOPrO also outperforms the approach of Li *et al.* in 10 out of 12 benchmarks, has similar performance on one, and performs slightly worse on one (XGBoost with the wrong prior). Lastly, BOPrO performs better than the approach of Ramachandran *et al.* in 7 out of 12 benchmarks, achieves similar performance in 4 benchmarks, and performs worse in 1 benchmark. We note that the method of Ramachandran *et al.* performs the best out of the three competitors, but cannot recover from misleading priors. On the other hand, the approach of Li *et*

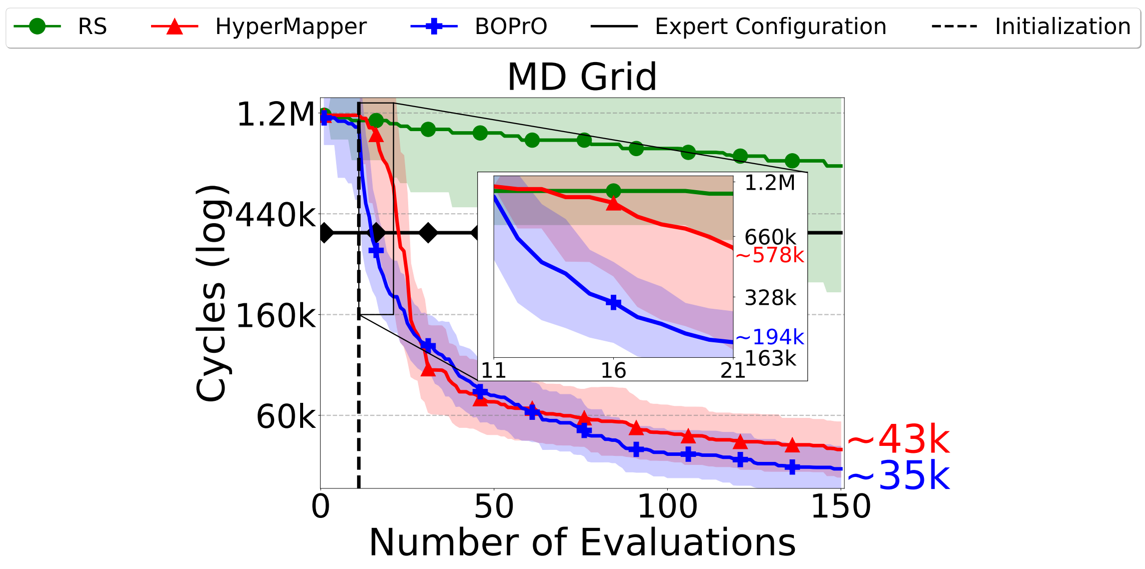


Figure 5.7. Log regret comparison of random sampling, HyperMapper, BOPrO, and a human expert configuration on the MD Grid benchmark. The line and shaded regions show the mean and standard deviation of the log simple regret after 3 runs. BOPrO and HyperMapper were initialized with $N + 1$ random samples, where N is the number of input dimensions, indicated by the vertical dashed line.

al. is the only competitor able to recover from misleading priors, but achieves weak performance overall when given accurate priors. BOPrO is the only approach that can consistently achieve good performance with both strong and weak priors, while also being robust to misleading priors.

5.2.4 The `Spatial` DSL Use-case

Lastly, we apply BOPrO to the `Spatial` DSL real-world application.`Spatial` is a programming language and corresponding compiler for the design of application accelerators, i.e., FPGAs. We apply BOPrO to three `Spatial` benchmarks, namely, 7D shallow and deep CNNs, and a 10D molecular dynamics grid application (MD Grid). We compare the performance of BOPrO to RS, manual optimization, and HyperMapper Nardi et al. [2018], the current state-of-the-art BO solution for `Spatial`. For a fair comparison between BOPrO and HyperMapper, since HyperMapper uses RFs as its surrogate model, here, we also use RFs in BOPrO. The manual optimization and the prior for BOPrO were provided by an unbiased `Spatial` developer. The priors were provided once and kept unchanged for the whole project.

We apply BOPrO to three `Spatial` benchmarks. The first two benchmarks are shallow and deep convolutional neural networks, while the third is a molecular dynamics application. The three benchmarks contain only discrete (ordinal and categorical) parameters. The search spaces and priors used for the three benchmarks are presented in Tables 5.2, 5.3, and 5.4. The three benchmarks also include feasibility constraints, meaning that some input parameter combinations lead to invalid configurations. This happens because some parameter combinations lead to configurations that use more logic units (or require more physical space) than what is provided by the target FPGA. To handle these constraints, we implement an approach similar to constrained Bayesian Optimization (cBO) [Gardner et al., 2014] in BOPrO. cBO weights the acquisition function at each configuration by the probability of the configuration being feasible, as predicted by a feasibility model. In our implementation, we use a RF model trained on previously evaluated configurations to predict the probability of a new configuration being feasible. When feasibility constraints are present, we also normalize the acquisition function before multiplying by the probability of feasibility, so that both are defined in the same range.

Figures 5.6 and 5.7 show the log regret on the `Spatial` benchmarks. BOPrO vastly outperforms RS in all benchmarks. BOPrO is also able to leverage the expert's prior and outperforms the expert in all benchmarks (2.73×, 1.05×, and 10.41× speedup for shallow CNN, deep CNN, and MD Grid, respectively). In the MD Grid benchmark, BOPrO achieves better performance than HyperMapper in the early stages of optimization (2.98× speedup after the first 10 iterations, see the plot inset), and achieves better final performance (1.21× speedup). For context, this is a significant improvement in the FPGA field, where a 10% improvement could qualify for acceptance in a top-tier conference. In the CNN benchmarks, BOPrO converges to the minima regions faster than HyperMapper (1.18× and 2× faster for shallow and deep, respectively). Thus, BOPrO leverages both the expert's prior knowledge and BO to provide a new state-of-the-art for `Spatial`.

Table 5.2. Search space, priors, and expert configuration for the Shallow CNN application. The default value for each parameter is shown in bold.

Parameter name	**Type**	**Values**	**Expert**	**Prior**
LP	Ordinal	[**1**, 4, 8, 16, 32]	16	[0.4, 0.065, 0.07, 0.065, 0.4]
P1	Ordinal	[**1**, 2, 3, 4]	1	[0.1, 0.3, 0.3, 0.3]
SP	Ordinal	[**1**, 4, 8, 16, 32]	16	[0.4, 0.065, 0.07, 0.065, 0.4]
P2	Ordinal	[**1**, 2, 3, 4]	4	[0.1, 0.3, 0.3, 0.3]
P3	Ordinal	[**1**, 2, ..., 31, 32]	1	[0.1, 0.1, 0.033, 0.1, 0.021, 0.021, 0.021, 0.1, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021, 0.021]
P4	Ordinal	[**1**, 2, ..., 47, 48]	4	[0.08, 0.0809, 0.0137, 0.1, 0.0137, 0.0137, 0.0137, 0.1, 0.0137, 0.0137, 0.0137, 0.05, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137, 0.0137]
x276	Categorical	[false, **true**]	true	[0.1, 0.9]

Table 5.3. Search space, priors, and expert configuration for the Deep CNN application. The default value for each parameter is shown in bold.

Parameter name	**Type**	**Values**	**Expert**	**Prior**
LP	Ordinal	[**1**, 4, 8, 16, 32]	8	[0.4, 0.065, 0.07, 0.065, 0.4]
P1	Ordinal	[**1**, 2, 3, 4]	1	[0.4, 0.3, 0.2, 0.1]
SP	Ordinal	[**1**, 4, 8, 16, 32]	8	[0.4, 0.065, 0.07, 0.065, 0.4]
P2	Ordinal	[**1**, 2, 3, 4]	2	[0.4, 0.3, 0.2, 0.1]
P3	Ordinal	[**1**, 2, ..., 31, 32]	1	[0.04, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.2, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.1, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.2]
P4	Ordinal	[**1**, 2, ..., 47, 48]	4	[0.05, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.13, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.2, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.11, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.2, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.005, 0.1]
x276	Categorical	[false, **true**]	true	[0.1, 0.9]

Table 5.4. Search space, priors, and expert configuration for the MD Grid application. The default value for each parameter is shown in bold.

Parameter name	**Type**	**Values**	**Expert**	**Prior**
loop_grid0_z	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
loop_q	Ordinal	[**1**, 2, ..., 31, 32]	8	[0.08, 0.08, 0.02, 0.1, 0.02, 0.02, 0.02, 0.1, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.1, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02]
par_load	Ordinal	[**1**, 2, 4]	1	[0.45, 0.1, 0.45]
loop_p	Ordinal	[**1**, 2, ..., 31, 32]	2	[0.1, 0.1, 0.1, 0.1, 0.05, 0.03, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02, 0.02]
loop_grid0_x	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
loop_grid1_z	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.2, 0.1, 0.1, 0.07, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]
loop_grid0_y	Ordinal	[**1**, 2, ..., 15, 16]	1	[0.2, 0.1, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05]
ATOM1LOOP	Categorical	[false, **true**]	true	[0.1, 0.9]
ATOM2LOOP	Categorical	[false, **true**]	true	[0.1, 0.9]
PLOOP	Categorical	[false, **true**]	true	[0.1, 0.9]

5.3 Ablation Studies

We perform a series of ablation experiments to demonstrate and validate some of the properties we envisioned for BOPrO. We explore the influence that priors of varying qualities, including misleading priors, have in BOPrO and the impact that the hyperparameters of BOPrO have in its optimization performance.

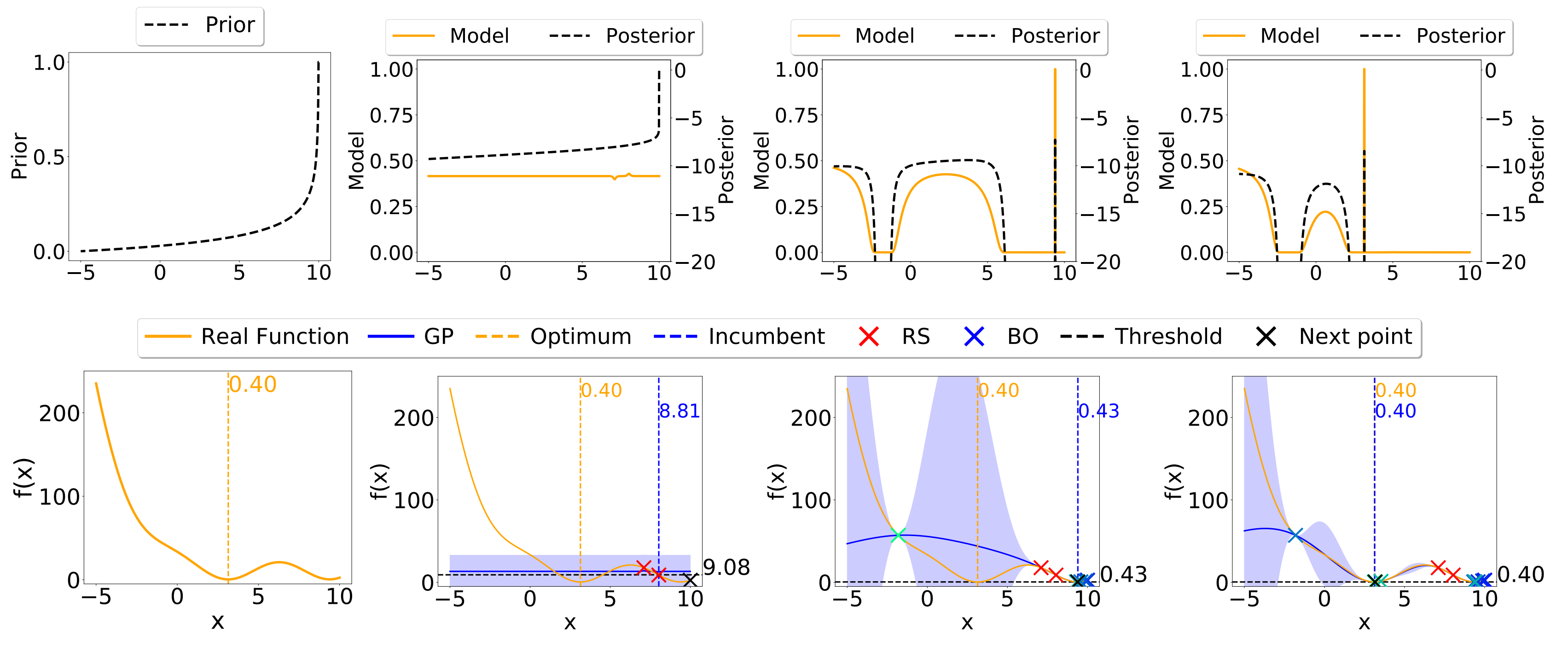
5.3.1 Prior Forgetting

In this section, we show that BOPrO can recover from a wrong prior and still find good function values. This is thanks to our predictive model and the t/β parameter in the pseudo-posterior computation. As BO progresses, the predictive model becomes more accurate and receives more weight in the pseudo-posterior computation, guiding optimization away from the wrong prior and towards better values of the objective function. We show this behavior on the 1D and 2D Branin functions.

Figure 5.8a shows BOPrO on the 1D Branin function with an exponential prior. Columns (b), (c), and (d) show BOPrO on the 1D Branin after $D+1=2$ initial samples and 0, 10, 20 BO iterations respectively. After initialization, as shown in column (b), the pseudo-posterior is nearly identical to the exponential prior and guides BOPrO towards the region of the space on the right, which is towards the local optimum. This happens until the predictive model becomes certain there will be no more improvement from sampling that region (columns (c) and (d)). After that, the predictive model guides the pseudo-posterior towards exploring regions with high uncertainty. Once the global minimum region is found, the pseudo-posterior starts balancing exploiting the global minimum and exploring regions with high uncertainty, as shown in column (d). Notably, the pseudo-posterior after $x > 6$ falls to 0 in 5.8d, as the predictive model is certain there will be no improvement from sampling this region which contains the local optimum.

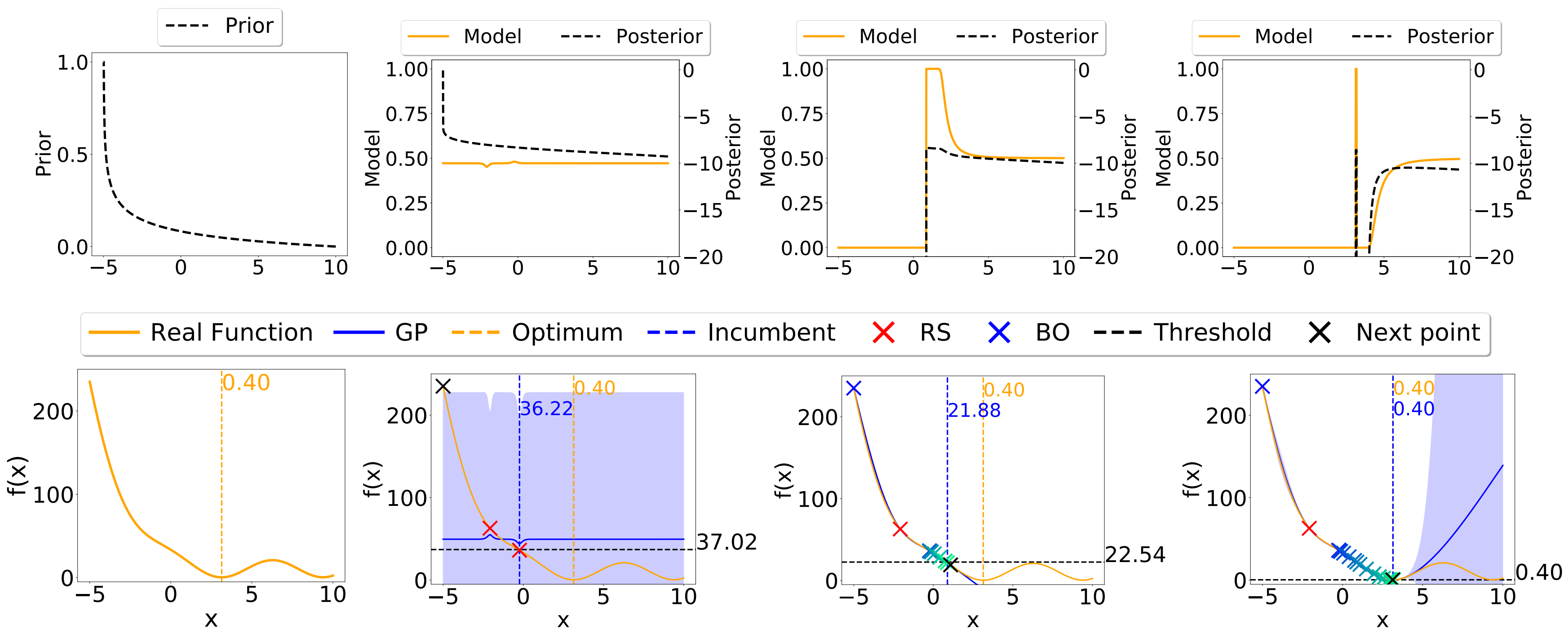
Figure 5.9 shows BOPrO on the 1D Branin function as in Figure 5.8 but with a decay prior. Column (a) of Figure 5.9 shows the decay prior and the 1D Branin function. Columns (b), (c), and (d) of Figure 5.9 show BOPrO on the 1D Branin after $D+1=2$ initial samples and 0, 10, and 20 BO iterations, respectively. At the beginning of BO, as shown in column (b), the pseudo-posterior is nearly identical to the prior and guides BOPrO towards the left region of the space. As more points are sampled, the model becomes more accurate and starts guiding the pseudo-posterior away from the wrong prior (column (c)). Notably, the pseudo-posterior before x = 2.5 falls to 0, as the predictive model is certain there will be no improvement from sampling this region. After 20 iterations, BOPrO finds the optimum region, despite the poor start (column (d)). The peak in the pseudo-posterior in column (d) shows BOPrO will continue to exploit the optimum region as the exact optimum has not been found yet. The pseudo-posterior is also high in the high uncertainty region after $x = 4$, showing BOPrO will explore that region after it finds the optimum.

Figure 5.10 shows BOPrO on the standard 2D Branin function. We use exponential priors for both dimensions, which guides optimization towards a region with only poor-performing high function values. 5.10a shows the prior and 5.10b shows optimization results after $D+1=3$ initialization samples and 50 BO iterations. Note that, once again, optimization begins near the region incentivized by the prior, but moves away from the prior and towards the optima as BO progresses.



(a) No samples (b) 0 BO iterations (c) 10 BO iterations (d) 20 BO iterations

Figure 5.8. BOPrO on the 1D Branin function. The leftmost column shows the exponential prior. The other columns show the model and the log pseudo-posterior after 0 (initialization only), 10, and 20 BO iterations. BOPrO forgets the wrong prior on the local optimum and converges to the global optimum.



(a) No samples (b) 0 BO iterations (c) 10 BO iterations (d) 20 BO iterations

Figure 5.9. BOPrO on the 1D Branin function with a decay prior. The leftmost column shows the log pseudo-posterior before any samples are evaluated, in this case, the pseudo-posterior is equal to the decay prior. The other columns show the model and pseudo-posterior after 0 (only random samples), 10, and 20 BO iterations. 2 random samples are used to initialize the GP model.

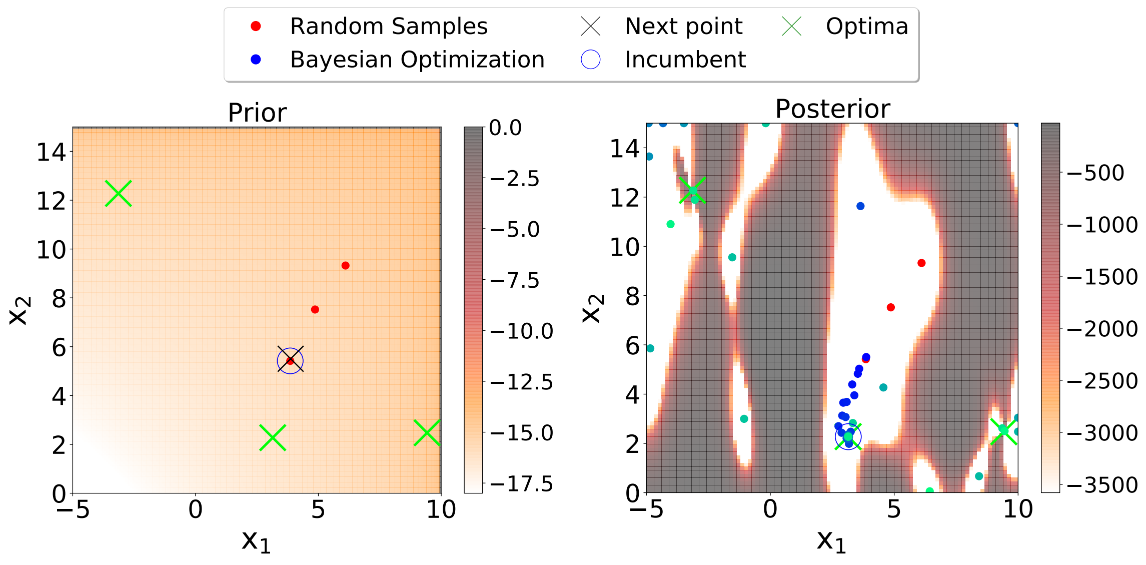


Figure 5.10. BOPrO on the Branin function with exponential priors for both dimensions. (a) shows the log pseudo-posterior before any samples are evaluated, in this case, the pseudo-posterior is equal to the prior; the crosses are the optima. (b) shows the result of optimization after 3 initialization samples drawn from the prior at random and 50 BO iterations. The dots in (b) show the points the exploration, with greener points denoting later iterations. The colored heatmap shows the prediction of the GP model for the value of the Branin function.

5.3.2 Misleading Prior Comparison

We showed that BOPrO can forget misleading priors and still converge towards good function values in Section 5.3.1. We also showed a comparison with misleading priors against other methods in Section 5.2.3. In this section, we expand the analysis to what is the actual optimization impact of a misleading prior, compared to both not having a prior and having correct priors in our four synthetic benchmarks. For that, we compare the performance of BOPrO with a misleading prior, with no prior, and with the weak and strong priors we used in our baseline comparisons.

Figure 5.11 shows the effect of injecting a misleading prior in BOPrO. We note similar behavior in all benchmarks, the misleading prior slows down convergence and hurts optimization. This is expected since the prior pushes the optimization away from the optima in the initial phase. However, in all benchmarks BOPrO still achieves similar final regret to BOPrO without a prior. These results show that BOPrO can effectively forget priors and achieve similar final performance to the case where there were no priors at all. However, it is still beneficial to have accurate priors, as the curves for the weak and strong correct priors show.

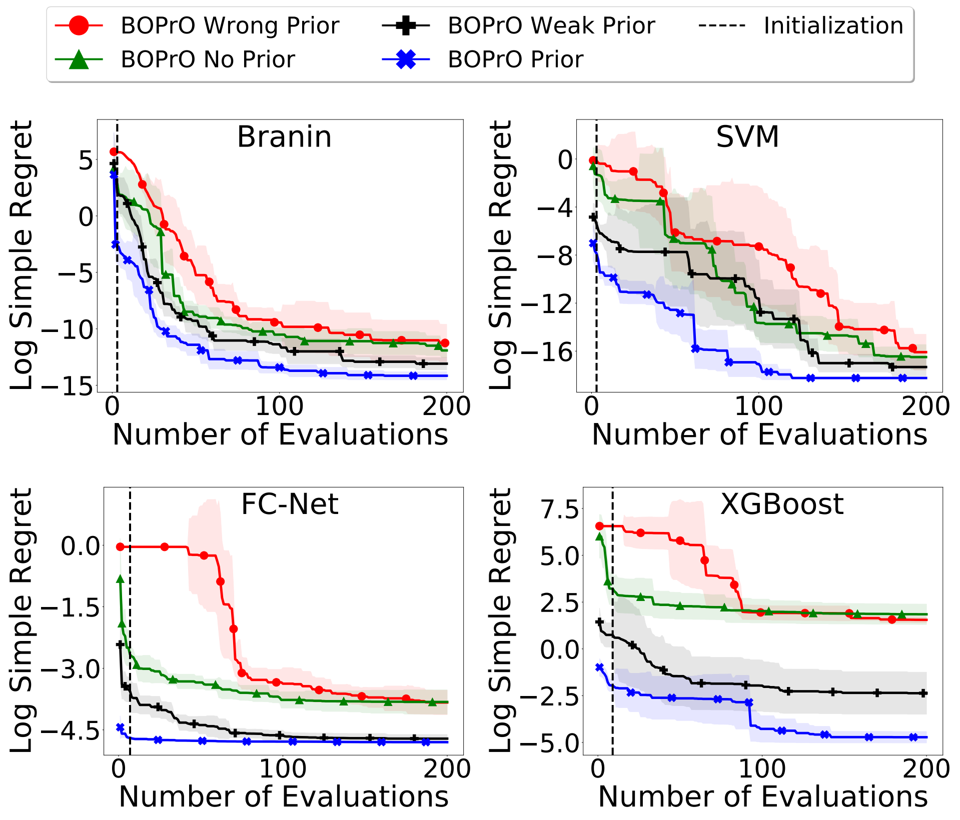


Figure 5.11. Log regret comparison of BOPrO with varying prior quality. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. All methods were initialized with $D + 1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 200 iterations.

5.3.3 Prior Impact

In this section, we study the effect of varying prior qualities in BOPrO. How does prior selection affect the performance of BOPrO? A suitable property of the prior is that, by selecting a tighter prior around an optimum, we would expect sampling from the prior to have an increased performance. To the limit, if the prior is composed by only one point which is one of the global optima, then the first sample (and all of them) from the prior will hit the optimum.

To have a sanity check of this property, we build an artificial prior in a controlled way. We rely on an automated computation of the prior by computing a univariate Kernel Density Estimation (KDE) using a Gaussian kernel on the synthetic benchmarks introduced above.[3] Compared to scipy's Scott's Rule $n^{-\frac{1}{D+4}}$, we set the original

[3]We chose a univariate KDE here following our assumption that most users will not be able to provide a multivariate prior with more than 2 dimensions.

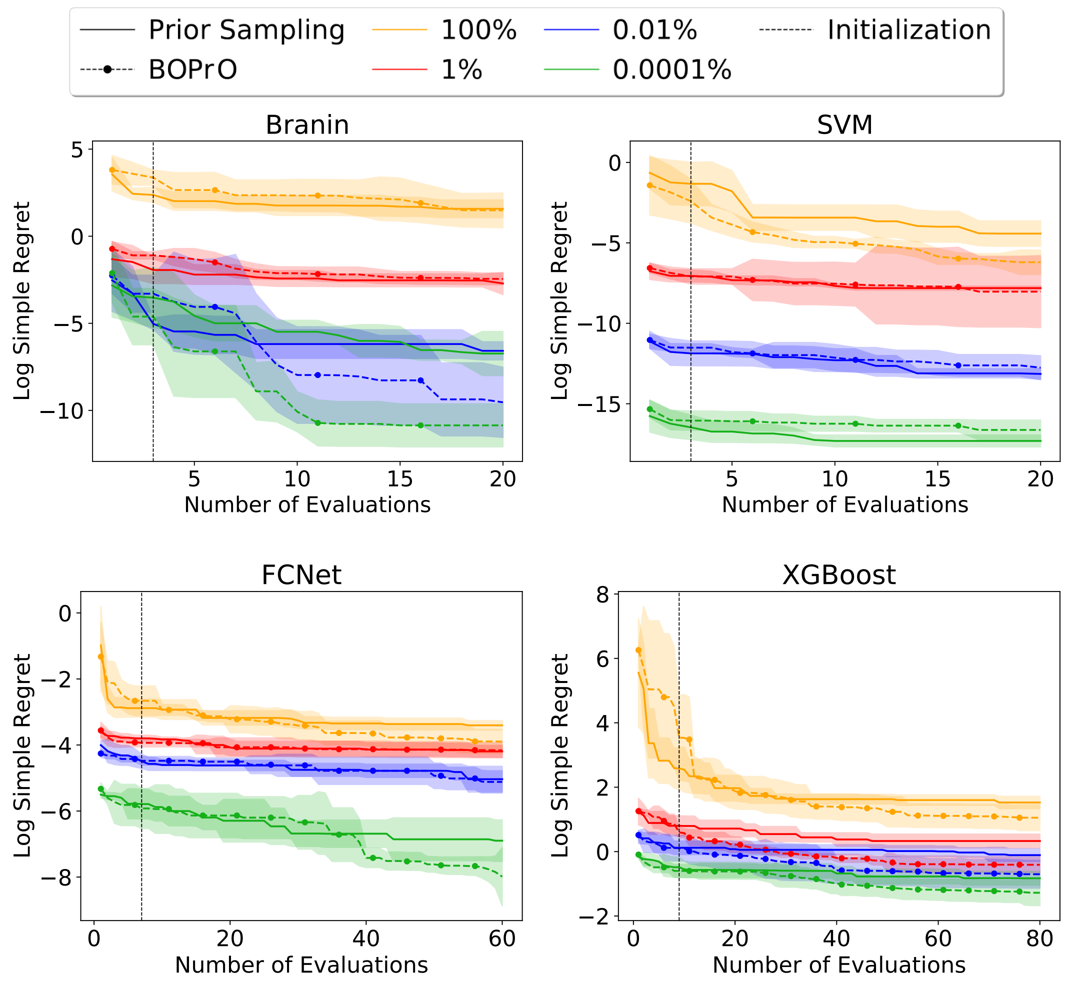


Figure 5.12. Log simple regret comparison between BOPrO and sampling from the prior. The shaded lines are mean +/- one std error. BOPrO was initialized with $D + 1$ random samples, indicated by the vertical dashed line.

bandwidth a bit smaller $100n^{-\frac{1}{D}}$, where D is the number of input dimensions. We run experiments with a budget $T = 10D$ and consider an array of varying quality priors. We set this bandwidth as it led to better performance in BOPrO and allowed us to make the prior stronger in a controlled environment. In Appendix A we show experiments with different bandwidth sizes and that they still lead to the same behavior and similar performance.

To control the strength of the prior, we select a constant $10D$ points in each prior and vary the size of the random sample dataset so that we can make the priors more sharply peaked around the optima in a controlled environment. We follow the following rule: we use the best performing $10D$ samples to create the prior from a random sample dataset size of $10D\frac{100}{x}$; we refer to this prior as $x\%$ in Figure 5.12.

As an example the XGBoost benchmark has $d = 8$, so, 100% means we sample 80 points and use all 80 to create the prior, 10% means we sample 800 points and use the best performing 80 to create the prior, 1% means we sample 8000 and use the best 80 to create the prior, and so on. A bigger random sample dataset and a smaller percentage leads to a tighter prior making the argument for a stronger prior.

Figure 5.12 shows the performance of purely sampling from the prior and running BOPrO with the prior respectively in all synthetic benchmarks. As expected a sharply peaked prior leads to better performance in all benchmarks. In addition, we observe that in contrast to sampling from the prior, BOPrO achieves a smaller regret in the vast majority of cases by being able to evolve from the initial prior and making independent steps towards better values of the objective function.

5.3.4 Prior Baselines Comparison

We show that simply initializing a BO method in the DoE phase by sampling from a prior on the locality of an optimum doesn't necessarily lead to better performance. Instead in BOPrO, it is the pseudo-posterior in Eq. (4.3) that drives its stronger performance by combining prior and new observations. To show that, we compare BOPrO with Spearmint and HyperMapper such as in section Sections 5.2.1 and 5.2.4, respectively, but we initialize all three methods using the same approach. Namely, we initialize all methods with $D + 1$ samples from the prior. Our goal is to show that simply initializing Spearmint and HyperMapper with the prior will not lead to the same performance as BOPrO, because, unlike BOPrO, these baselines do not leverage the prior after the DoE initialization phase. We report results on both our synthetic and real-world benchmarks

Figure 5.13 shows the comparison between BOPrO and Spearmint Prior. In most benchmarks, the prior initialization does not lead to improved final performance. In particular, for XGBoost, the prior leads to improvement in early iterations, but to worse final performance. We note that for SVM, the prior leads to better performance, however, we note that the improved performance is given solely from sampling from the prior. There is no improvement for Spearmint Prior after initialization. In contrast, in all cases, BOPrO is able to leverage the prior both during initialization and its Bayesian Optimization phase, leading to improved performance. BOPrO still outperforms Spearmint Prior in the same 3 out of 4 benchmarks.

Figure 5.14 shows similar results for our `Spatial` benchmarks. Once again, the prior does not lead HyperMapper to improved final performance. For the Shallow CNN benchmark, the prior leads HyperMapper to improved performance in early iterations, compared to HyperMapper with default initialization, but HyperMapper Prior is still outperformed by BOPrO. Additionally, the prior leads to degraded performance in the

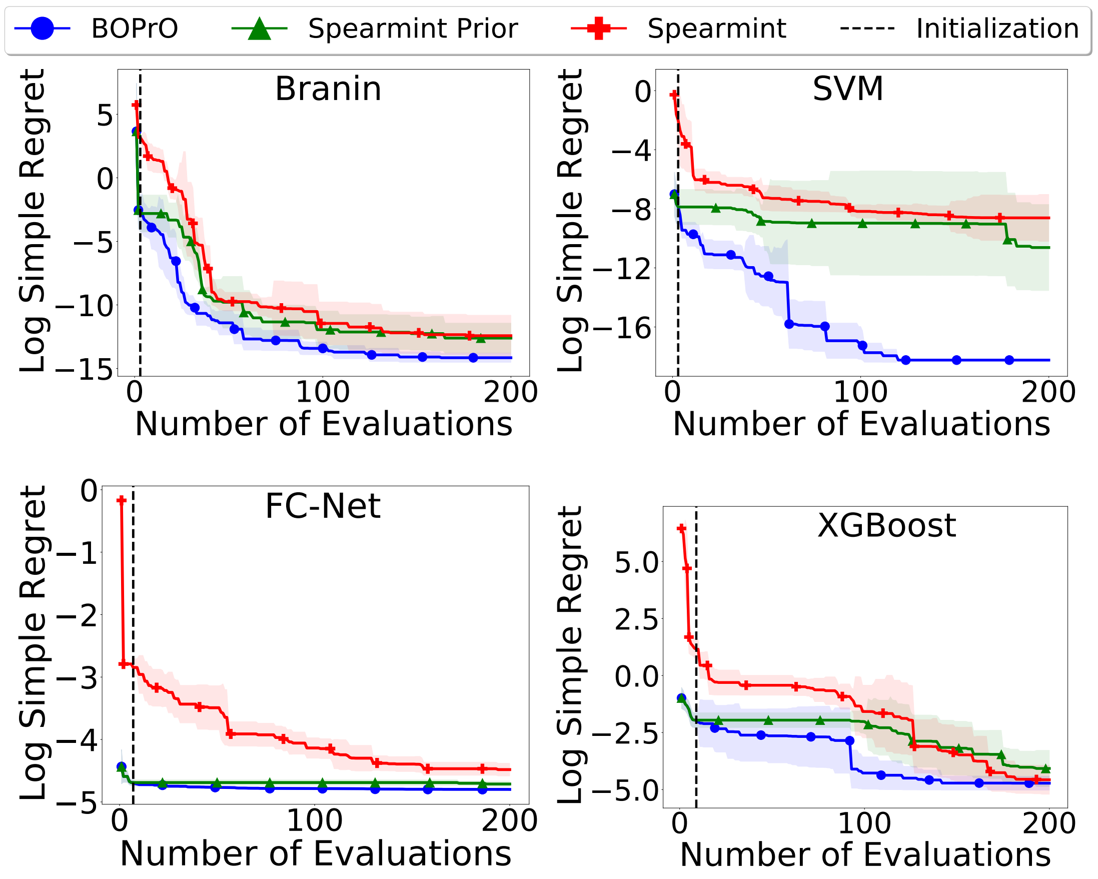


Figure 5.13. Log regret comparison of BOPrO, Spearmint with prior initialization, and Spearmint with default initialization. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. BOPrO and Spearmint Prior were initialized with $D + 1$ random samples from the prior, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for $100D$ iterations, capped at 300.

Deep CNN benchmark. These results confirm that BOPrO is able to leverage the prior in its pseudo-posterior during optimization, leading to improved performance in almost all benchmarks compared to state-of-the-art BO baselines.

5.3.5 Multivariate Prior Comparison

In this section, we compare the performance of BOPrO with univariate and multivariate priors. For this, we construct synthetic univariate and multivariate priors using Kernel Density Estimation (KDE) with a Gaussian kernel. We build strong and weak versions of the KDE priors. The strong priors are computed using a KDE on the best $10D$ out of $10{,}000{,}000D$ uniformly sampled points, while the weak priors are computed

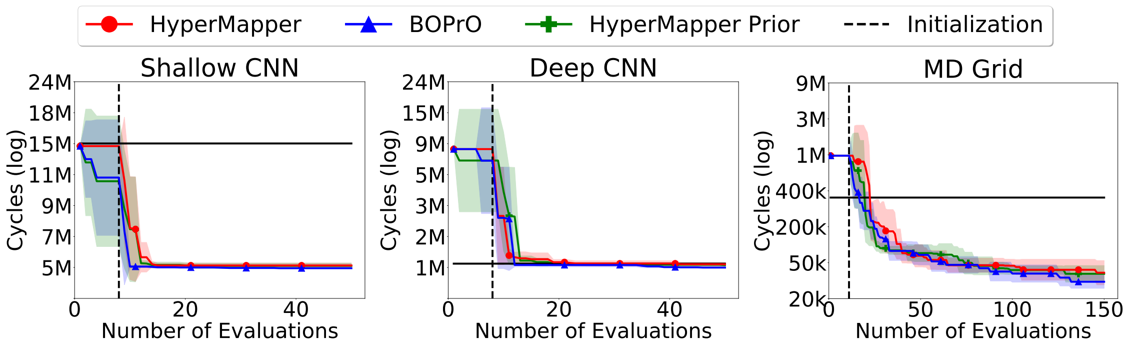


Figure 5.14. Log regret comparison of BOPrO, HyperMapper with prior initialization, and HyperMapper with default initialization. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. BOPrO and HyperMapper Prior were initialized with $D+1$ random samples from the prior, where D is the number of input dimensions, indicated by the vertical dashed line.

using a KDE on the best $10D$ out of $1{,}000D$ uniformly sampled points. We use the same points for both univariate and multivariate priors. We use scipy's Gaussian KDE implementation, but adapt its Scott's Rule bandwidth to $100n^{-\frac{1}{d}}$, where d is the number of variables in the KDE prior, to make our priors more peaked.

Figure 5.15 shows a log regret comparison of BOPrO with univariate and multivariate KDE priors. We note that in all cases BOPrO achieves similar performance with univariate and multivariate priors. For the Branin and SVM benchmarks, the weak multivariate prior leads to slightly better results than the weak univariate prior. However, we note that the difference is small, in the order of 10^{-4} and 10^{-6}, respectively.

Surprisingly, for the XGBoost benchmark, the univariate version for both the weak and strong priors lead to better results than their respective multivariate counterparts, though, once again, the difference in performance is small, around 0.2 and 0.03 for the weak and strong prior, respectively, whereas the XGBoost benchmark can reach values as high as $f(\boldsymbol{x}) = 700$. Our hypothesis is that this difference comes from the bandwidth estimator ($100n^{-\frac{1}{d}}$), which leads to larger bandwidths, consequently, smoother priors, when a multivariate prior is constructed.

5.3.6 β Sensitivity Study

In this section, we show the effect of the γ hyperparameter on BOPrO. We recall that the β parameter controls how much weight is given to the model vs the prior in BOPrO's pseudo-posterior computation. To show the effects of β, we compare the performance of BOPrO with the 1% prior and different β values on our four synthetic benchmarks. For

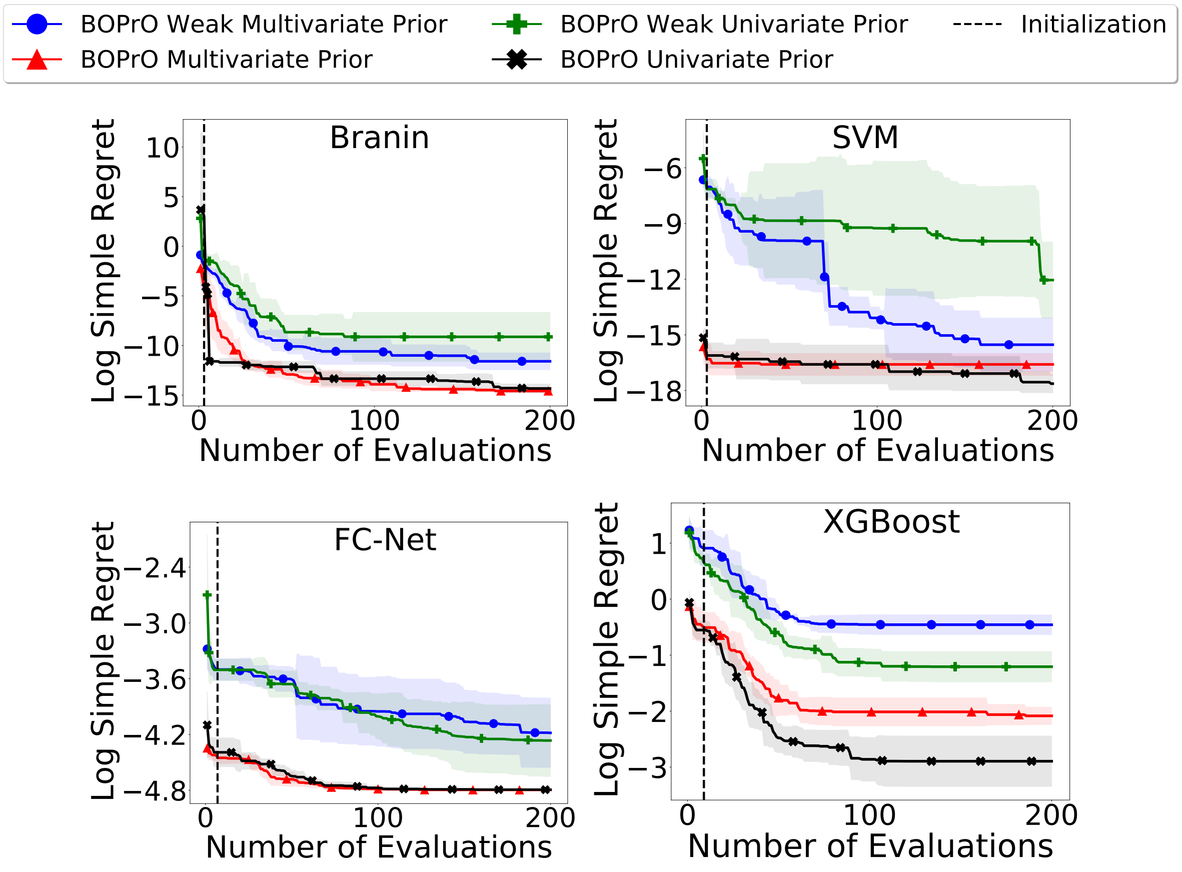


Figure 5.15. Log regret comparison of BOPrO with multivariate and univariate KDE priors. The line and shaded regions show the mean and standard deviation of the log simple regret after 20 runs. All methods were initialized with $D+1$ random samples, where D is the number of input dimensions, indicated by the vertical dashed line. We run the benchmarks for 200 iterations.

all experiments, we initialize BOPrO with D+1 random samples and then run BOPrO until it reaches 10D function evaluations. For each β value, we run BOPrO five times and analyze mean and standard deviation.

Figure 5.16 shows the results of our comparison. We note that values of β that are too low (near 0.01) or too high (near 1000) lead to lower performance. This shows that putting too much emphasis on the model or the prior will lead to degraded performance, as expected. Further, we note that $\beta = 10$ leads to the best performance in three out of our four benchmarks. This result is reasonable, as $\beta = 10$ means BOPrO will put more emphasis in the prior in early iterations, when the GP model is still not accurate, and slowly shift towards putting more emphasis in the model as the model sees more data and becomes more accurate.

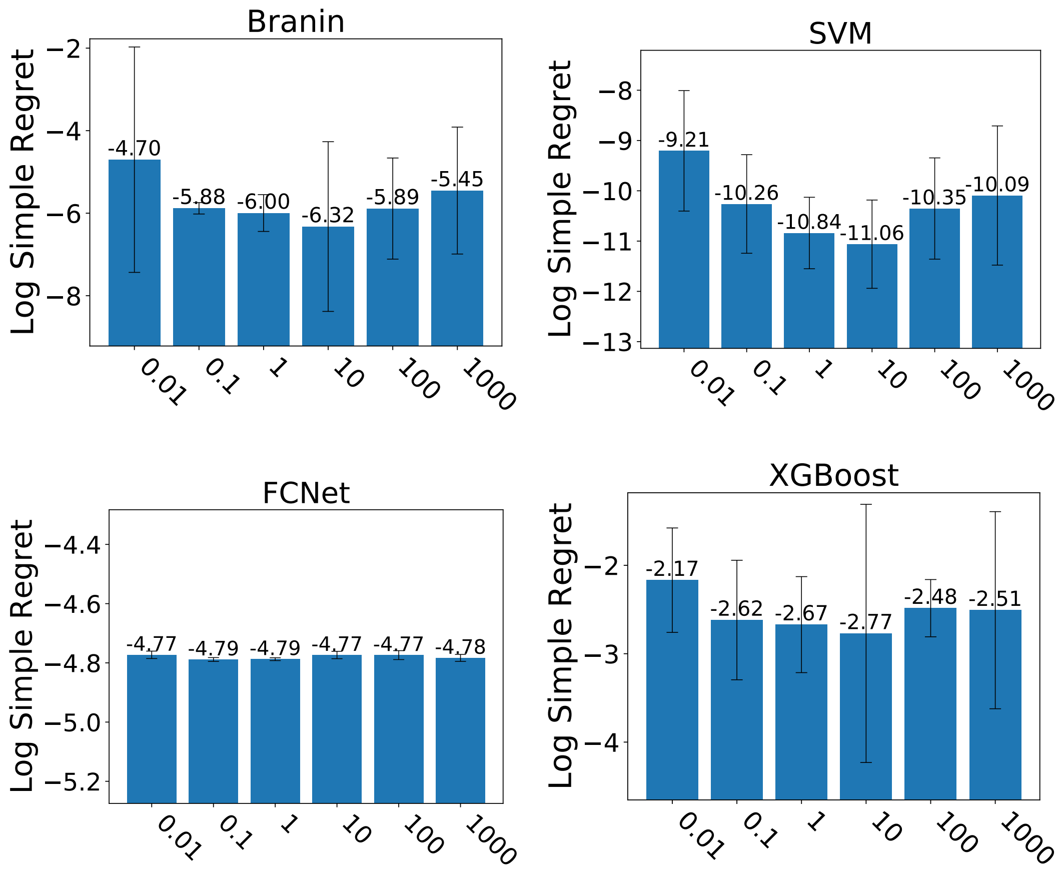


Figure 5.16. Comparison of BOPrO with the 1% prior and different values for the β hyperparameter on our four synthetic benchmarks. We run BOPrO with a budget of 10D function evaluations, including D+1 randomly sampled DoE configurations.

5.3.7 γ Sensitivity Study

In this section, we show the effect of the γ hyperparameter on BOPrO. We note that γ directly affects BOPrO's exploration vs exploitation tradeoff. Higher γ values emphasize exploitation, while lower γ values emphasize exploration. This is because γ dictates the threshold that separates good regions from bad regions. This threshold is defined as a quantile of the previously explored points, this means the threshold is always higher than a fraction of the explored points. Thus, a higher gamma will lead to higher thresholds, and points near the incumbent will likely be considered good. However, a lower gamma will cause the threshold to approximate the incumbent faster and points near the incumbent will be less likely to bring an improvement.

To show the effects of γ, we compare the performance of BOPrO with different γ values. We compare the performance of BOPrO with the 1% prior on our four synthetic

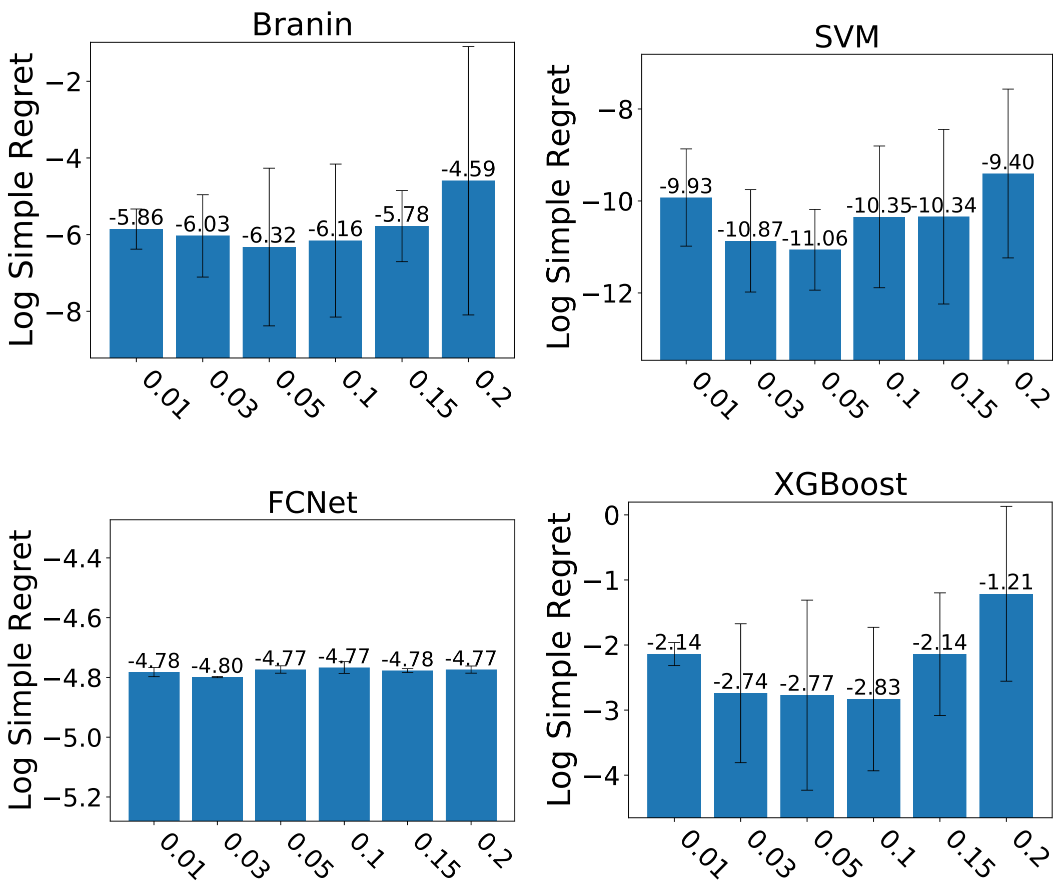


Figure 5.17. Comparison of BOPrO with the 1% prior and different values for the γ hyperparameter on our four synthetic benchmarks. We run BOPrO with a budget of 10D function evaluations, including D+1 randomly sampled DoE configurations.

benchmarks. For all experiments, we initialize BOPrO with D+1 random samples and then run BOPrO until it reaches 10D function evaluations. For each γ value, we run BOPrO five times and analyze mean and standard deviation.

Figure 5.17 shows the results of our comparison. We first note that values near the lower and higher extremes lead to degraded performance, this is expected since these values will lead to an excess of either exploitation or exploration. These results emphasize the importance of a proper balance in the exploration vs exploitation trade-off. Further, we note that BOPrO achieves similar performance for all values of γ, however, $\gamma = 0.03$ and $\gamma = 0.05$ consistently lead to better performance, with $\gamma = 0.05$ usually leading to lower deviation. Thus, we set $\gamma = 0.05$ as the default in our experiments.

Chapter 6

Other Projects and Future Work

6.1 The HyperMapper Framework

Throughout the course of my Ph.D. and my work in the BO field, we leveraged our studies and novel techniques developed to implement an open-source BO framework dubbed HyperMapper. HyperMapper is a BO framework that implements several state-of-the-art BO strategies, including some of our own design. Additionally, HyperMapper provides several optimization features that are often required by real-world application, such as constrained BO, discrete variables, and support for mixed search spaces. The combination of high-performance methods and flexibility features makes HyperMapper stand out among competition as a great solution for real-world applications.

We summarize the features provided by HyperMapper in Table 6.1. We implemented this features based on known requirements present in the literature and our own personal experience talking to prospective users. First, HyperMapper implements four type of variables: real (numbers in a real-valued interval), integer (numbers in an integer-valued interval), ordinal (an ordered list of numbers), and categorical (a unordered list of values, possibly non-numeric). To better support these discrete and possibly mixed search spaces, we implement both GP and RF models, which handle better continuous and discrete search spaces, respectively. Our GP implementation is the one provided by GPy, while our RF implementation is a variation of the implementation described by Hutter *et al.* Hutter et al. [2014], using the model provided by scikit-learn Pedregosa et al. [2011].

Second, we implement a feasibility model in HyperMapper, to support feasibility constraints. Feasbility constraints refer to applications where some configurations are considered infeasible solutions for the application. For instance, in our `Spatial` application, some configurations require more resources (e.g. more processing units)

	RIOC	Feasibility Constraints	Multi-objective	Prior-injection
HyperMapper	✓	✓	✓	✓
KTT	×	×	×	×
GPTune	×	×	×	×
SMAC3	✓	×	×	×
GpyOpt	×	×	×	×
Spearmint	×	✓	×	×
Hyperopt	✓	×	×	✓
Hyperband	×	×	×	×
GPflowOpt	×	✓	✓	×
cBO	×	✓	×	×
BOHB	✓	×	×	×

Table 6.1. Features supported by HyperMapper and competing BO frameworks. RIOC abbreviates Real/Integer/Ordinal/Categorical, for the different variable types supported.

than the target FPGA could provide and are, thus, impossible to synthesize in practice. These feasibility constraints are often impossible or too hard to describe beforehand in optimization, thus, they need to be discovered (and learned) during optimization. For our feasibility constraint approach, we use an approach similar to that of Gardner *et al.* Gardner et al. [2014], which trains a model to predict which configurations will be feasible/infeasible as optimization progresses. We adapt the solution of Gardner *et al.* to use a RF classification model, instead of the proposed GP regression model combined with a feasibility threshold. Using a classification model directly simplifies our approach as it no longer requires a feasibility threshold from the user.

Thid, we support applications with multiple optimization objectives. In applications with multiple objectives, there are usually two or more objectives that one wants to optimize simultaneously. Further, these multiple objectives are often contradicting, meaning configurations that improve one objective, make the others worse. In these situations, we want to find the objectives' Pareto-front Knowles [2006], the combination of all configurations that cannot be improved further in any objective without making another objective worse (i.e. the configurations that are Pareto-efficient). Our multi-objective approach is based on random scalarizations, similar to the work of Paria *et al.* Paria et al. [2018]. Simply put, at each optimization iteration, we randomly sample a set of weights $\boldsymbol{\lambda}$ and use it to scalarize the multiple objectives into a single value through a scalarization function $g(\boldsymbol{\lambda}, \boldsymbol{x})$. Our work diverges from the work of Paria *et al.* in our scalarized acquisition function. Paria *et al.* propose their version of the Tchebyshev scalarization function, defined as:

$$g_{tch_m}(\lambda, x) = \min_{k=1}^{K} \lambda_k (f_k(x) - z_k^*) \tag{6.1}$$

Where z_k^* is an ideal reference point, often taken to be the best possible value of the objectives. Although they dub it "Tchebyshev scalarization function", there are notable differences from their equation and other versions of Tchebyshev scalarization in the literature Knowles [2004]; Nakayama et al. [2009], hence, we instead use an tchebyshev scalarization function more aligned with the literature, defined as:

$$g_{tch}(\lambda, x) = \max_{k=1}^{K} \lambda_k f_k(x) + \alpha \sum_{k=1}^{K} \lambda_k f_k(x) \tag{6.2}$$

Fourth, and at last, HyperMapper supports expert prior-injection. HyperMapper supports two types of prior-injection. The first is the one described in this thesis project, the second is an altered approach we also developed, as a follow-up to this thesis work. This follow-up approach is dubbed πBO and has been described in detail in an ICLR publication Hvarfner et al. [2022]. The key idea of our follow-up approach is that it diverges less from a standard BO approach, using the priors as a weighting factor on top of the standard BO acquisition function.

6.2 Multi-objective Prior Optimization

In my thesis work, we also investigated extending HyperMapper's prior-injection approach for multiple objectives. Our proposal to achieve this was to use random scalarizations on the computed posterior. For multi-objective applications, we would consider separate priors and models for each objective. For the priors, the user provides one probability distribution per variable i and per objective k. We then construct the prior for each objective k by multiplying the probability distributions for that objective, as in Eq. (4.1). Similarly, for the model, we fit a separate GP for each objective k and compute a probabilistic model for each objective following Eq. (4.2). We then combine the prior and model for each objective into a posterior $g_k(\boldsymbol{x})$ for each objective k. In order to scalarize the posteriors, we then randomly sample a set of weights $\boldsymbol{\lambda}$ and multiply the weighted posteriors:

$$g(\boldsymbol{x}) = \prod_{k=1}^{K} g_k(\boldsymbol{x})^{\lambda_k} \tag{6.3}$$

where K is the number of objectives and λ_k is the weight given to objective k. The same equation is used to scalarize the bad posteriors $b(\boldsymbol{x})$, only using $b_k(\boldsymbol{x})$ instead of $g_k(\boldsymbol{x})$. This leads us to a single scalarized good posterior and a single scalarized bad posterior, which allows us to continue with BOPrO as in single-objective applications.

Although this approach is conceptually simple and theoretically similar to the original scalarization approach, it did not perform as well as we hoped it would in our experiments. Because of the extra added components, performance in optimization became somewhat inconsistent. We suspect this approach requires some fine-tuning in order to balance the differing terms and ensure the space is properly explored. For now, this approach remains as future work for us.

6.3 Sparse Space Optimization

We also investigated the issue of optimizing applications with sparse search spaces. In these applications, there are feasibiliy constraints in the search space that are known in advance. These feasibility constraints describe combinations of input parameters that are considered infeasible and, thus, represent known holes in the search space (i.e. the search space is sparse). To tackle this issue, we adapt the chain-of-trees approach proposed by Rasch *et al.* Rasch et al. [2017] for a BO setup. The crux of our approach lies on optimizing the acquisition function considering only configurations that are known to be feasible, coupled with several optimizations to the GP model used in optimization. We also change how random sampling is done in the chain-of-trees, to better cover the search space uniformly. Details of our approach will be presented in an upcoming publication.

Chapter 7

Conclusion

In this thesis and my Ph.D. work, we introduce BOPrO, a Bayesian Optimization framework that allows users to inject their expert knowledge into the optimization process in the form of priors about which parts of the input space will yield the best performance. These are different than BO's standard priors over functions which are much less intuitive for users. BO failed so far to leverage the experience of human domain experts, not only causing function evaluations waste but also getting users away from applying of BO approaches because they could not exploit their years of knowledge in optimizing their black-box functions, e.g., hyperparameter optimization of machine learning algorithms. BOPrO addresses this issue and will nudge new users to adopt BO.

BOPrO advanced the state-of-the-art of BO by combining standard BO probabilistic models with flexible user priors in an efficient, yet robust manner. BOPrO adopts a Bayesian-inspired approach that combines user priors with BO's probabilistic model, forming a pseudo-posterior that balances both the prior and the model. Additionally, we add a weight hyperparameter to this computation to ensure that, given time and enough samples, the probabilistic model eventually washes out the prior. This approach allows BOPrO to leverage the prior to speed up BO convergence while remaining robust to misleading priors.

We validate BOPrO with a suite of synthetic and real benchmarks. We show that BOPrO is more sample-efficient than state-of-the-art BO solutions with no user prior support and that BOPrO is also more sample-efficient and robust than current BO solutions with user prior support. We also show BOPrO in a real-world setting, with priors provided by an unbiased human expert. We also perform a series of ablation studies on BOPrO, validating our assumptions and showing that BOPrO achieves the goals we set forth for it. We show that BOPrO can recover from misleading priors and still converge to find good solutions, even for priors that are extremely confident and wrong. We also study the impact of different prior qualities in the performance of BOPrO and validate our assumption of univariate versus multivariate priors leading to similar qualities. At last, we perform sensitivity studies to show the impact of choosing BOPrO's hyperparameters.

We also made several other contributions, besides BOPrO itself, during the course of this Ph.D.. Most notably, we developed HyperMapper, a black-box multi-objective optimization tool that combines several state-of-the-art approaches (including many of our own design) to provide a complete and efficient solution for BO. We also worked on BO for sparse spaces and a follow-up prior work in prior-injection for BO. At last, we also investigated expanding our prior-injection for multi-objective applications, but decided that this requires further investigation to be published and used in real-world applications, thus, we will tackle it in future work.

Bibliography

Azimi, J., Jalali, A., and Fern, X. (2012). Hybrid batch bayesian optimization. *arXiv preprint arXiv:1202.5597*.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. (2019). Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*.

Bergstra, J., Yamins, D., and Cox, D. D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546--2554.

Bodin, B., Nardi, L., Zia, M. Z., Wagstaff, H., Sreekar Shenoy, G., Emani, M., Mawer, J., Kotselidis, C., Nisbet, A., Lujan, M., et al. (2016). Integrating algorithmic parameters into benchmarking and design space exploration in 3d scene understanding. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation*, pages 57--69. ACM.

Bouthillier, X. and Varoquaux, G. (2020). Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020. Research report, Inria Saclay Ile de France.

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

Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. (2016). Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5--23.

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

Dixon, L. C. W. (1978). The global optimization problem: an introduction. *Toward global optimization*, 2:1--15.

Eriksson, D., Pearce, M., Gardner, J. R., Turner, R., and Poloczek, M. (2019). Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1436--1445.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015a). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 2962--2970. Curran Associates, Inc.

Feurer, M., Springenberg, J. T., and Hutter, F. (2015b). Initializing bayesian hyperparameter optimization via meta-learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 1128--1135.

Gardner, J. R., Kusner, M. J., Xu, Z. E., Weinberger, K. Q., and Cunningham, J. P. (2014). Bayesian Optimization with Inequality Constraints. In *ICML*.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J. E., and Sculley, D. (2017). Google vizier: A service for black-box optimization. In Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J. E., and Sculley, D., editors, *Proceedings of KDD'17*.

GPy (since 2012). GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Hansen, N., Akimoto, Y., and Baudis, P. (2019). CMA-ES/pycma on Github.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507--523. Springer.

Hutter, F., Kotthoff, L., and Vanschoren, J., editors (2018). *Automated Machine Learning: Methods, Systems, Challenges.* Springer. In press, available at http://automl.org/book.

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

Hvarfner, C., Stoll, D., Souza, A., Lindauer, M., Hutter, F., and Nardi, L. (2022). \ π bo: Augmenting acquisition functions with user beliefs for bayesian optimization. *arXiv preprint arXiv:2204.11051.*

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

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

Klein, A., Dai, Z., Hutter, F., Lawrence, N. D., and Gonzalez, J. (2019). Meta-surrogate benchmarking for hyperparameter optimization. In *Advances in Neural Information Processing Systems 32 NeurIPS*, pages 6267--6277.

Knowles, J. (2004). ParEGO: A Hybrid Algorithm with On-line Landscape Approximation for Expensive Multiobjective Optimization Problems. Technical report TR-COMPSYSBIO-2004-01, University of Manchester.

Knowles, J. (2006). Parego: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50--66.

Knudde, N., van der Herten, J., Dhaene, T., and Couckuyt, I. (2017). Gpflowopt: A bayesian optimization library using tensorflow. *arXiv preprint arXiv:1711.03845.*

Koeplinger, D., Feldman, M., Prabhakar, R., Zhang, Y., Hadjis, S., Fiszel, R., Zhao, T., Nardi, L., Pedram, A., Kozyrakis, C., et al. (2018). Spatial: A language and compiler for application accelerators. In *ACM Sigplan Notices*, volume 53, pages 296--311. ACM.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97--106.

Li, C., Gupta, S., Rana, S., Nguyen, V., Robles-Kelly, A., and Venkatesh, S. (2020). Incorporating expert prior knowledge into experimental design via posterior sampling. *arXiv preprint arXiv:2002.11256*.

Lindauer, M. and Hutter, F. (2018). Warmstarting of model-based algorithm configuration. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 1355--1362.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43--58.

Marchant, R. and Ramos, F. (2012). Bayesian optimisation for intelligent environmental monitoring. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 2242--2249. IEEE.

Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2.

Nakayama, H., Yun, Y., and Yoon, M. (2009). *Sequential Approximate Multiobjective Optimization Using Computational Intelligence*. Springer Science & Business Media.

Nardi, L., Bodin, B., Saeedi, S., Vespa, E., Davison, A. J., and Kelly, P. H. (2017). Algorithmic performance-accuracy trade-off in 3d vision applications using hypermapper. In *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1434--1443. IEEE.

Nardi, L., Koeplinger, D., and Olukotun, K. (2018). Practical Design Space Exploration. *arXiv preprint arXiv:1810.05236*.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Oh, C., Gavves, E., and Welling, M. (2018). Bock: Bayesian optimization with cylindrical kernels. *arXiv preprint arXiv:1806.01619*.

Paleyes, A., Pullin, M., Mahsereci, M., Lawrence, N., and González, J. (2019). Emulation of physical processes with emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*.

Paria, B., Kandasamy, K., and Póczos, B. (2018). A Flexible Multi-Objective Bayesian Optimization Approach using Random Scalarizations. *CoRR*, abs/1805.12168.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024--8035.

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

Ramachandran, A., Gupta, S., Rana, S., Li, C., and Venkatesh, S. (2020). Incorporating expert prior in bayesian optimisation via space warping. *Knowledge-Based Systems*, 195:105663.

Rasch, A., Haidl, M., and Gorlatch, S. (2017). Atf: A generic auto-tuning framework. In *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 64--71. IEEE.

Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science*, pages 409--423.

Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2016). Scalable hyperparameter optimization with products of gaussian process experts. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, pages 33--48.

Shahriari, B., Bouchard-Côté, A., and Freitas, N. (2016). Unbounded bayesian optimization via regularization. In *Artificial intelligence and statistics*, pages 1168--1176.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148--175.

Siivola, E., Vehtari, A., Vanhatalo, J., González, J., and Andersen, M. R. (2018). Correcting boundary over-exploration deficiencies in bayesian optimization with virtual derivative sign observations. In *International Workshop on Machine Learning for Signal Processing*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951--2959.

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

Souza, A., Oliveira, L. B., Hollatz, S., Feldman, M., Olukotun, K., Holton, J. M., Cohen, A. E., and Nardi, L. (2019). Deepfreak: Learning crystallography diffraction patterns with automated machine learning. *arXiv preprint arXiv:1904.11834*.

Swersky, K. (2017). *Improving Bayesian Optimization for Machine Learning using Expert Priors*. PhD dissertation, University of Toronto (Canada).

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2018). Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43--78.

Appendix A

KDE Prior Bandwidth

In this section, we show the effect of different bandwidth sizes on the univariate KDE prior. For that, we compare the performance of sampling from the prior and BOPrO with different bandwidth sizes. We consider four variations of scipy's Scott's Rule $an^{-\frac{1}{D+b}}$. We experiment with $a = 1$, $b = 4$ (scipy's default); $a = 1$, $b = 0$; $a = 10$, $b = 0$; and $a = 100$, $b = 0$. Note that larger values for a and smaller values for b lead to smaller bandwidths. For each bandwidth size, we show results for an array of varying quality priors. We select a constant $10D$ points in each prior and vary the size of the random sample dataset. We follow the following rule: we use the best performing $10D$ samples to create the prior from a random sample dataset size of $10D\frac{100}{x}$; we refer to this prior as $x\%$. We experiment with dataset sizes varying from $10D$ to 10^7D.

Figures A.1- A.4 show the performance of purely sampling from the prior. We note that, in most cases, using a larger dataset leads to better results. This is expected, sampling more points means we find more points near the optima and, therefore, the prior will be built with points closer to the optima. Likewise, we note that smaller bandwidths often lead to better results, especially as more points are sampled. This is also expected since a smaller bandwidth means the prior distribution will be more peaked around the optima. However, there are a couple of exceptions to these trends. First, for the Branin, sampling more points does not lead to a better prior when we use $a = 1, b = 4$, this is likely because the multiple minima of the Branin and the bigger bandwidth lead the prior to be oversmoothed, missing the peaks near the optima. Second, smaller bandwidths do not always lead to better performance for smaller random samples datasets. This happens because we find points farther from the optima in these datasets and end up computing priors peaked at points that are farther from the optima, i.e., our priors become misleading. The effects of these misleading priors can be especially noticed for the 100% random samples dataset.

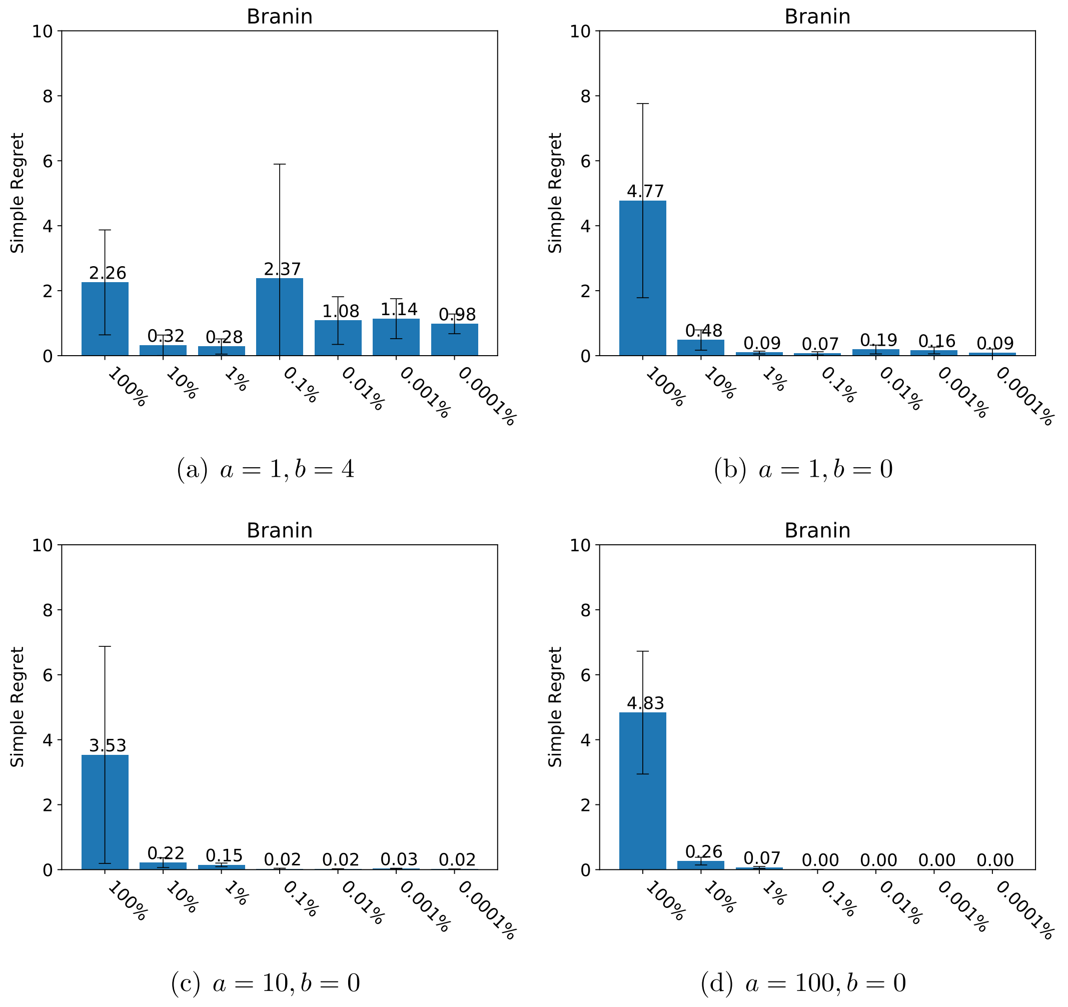
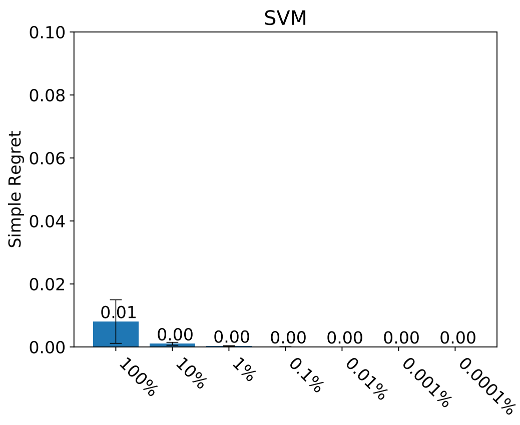
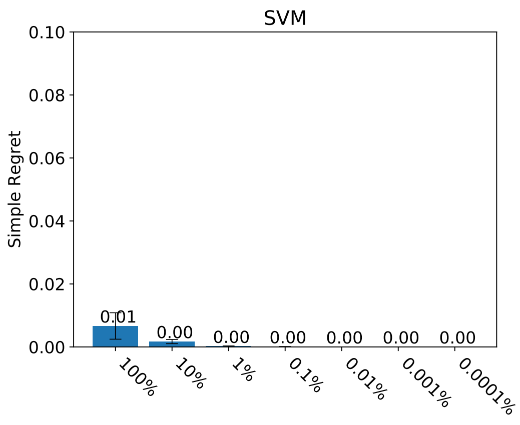


Figure A.1. Simple regret of sampling from the prior with different priors for our Branin benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars.

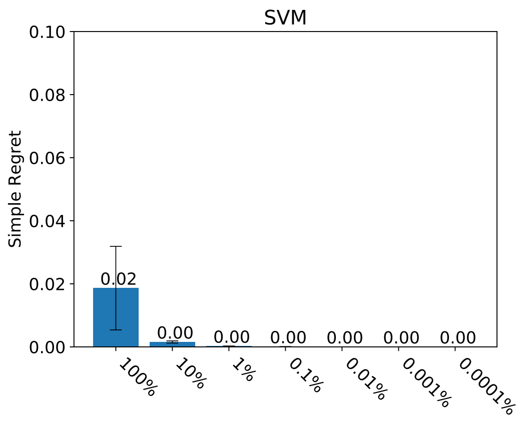
Figures A.5- A.8 shows the performance of BOPrO for different priors. The same observations from prior sampling hold here. Namely, sampling more points and using smaller bandwidths lead to better performance. Also, the 100% dataset once again leads to inconsistent results, since it is a misleading prior for BOPrO. Based on these results, we use the smallest bandwidth and largest dataset in our experiments, i.e. $a = 100, b = 0$, and 0.0001%. Intuitively, this is a reasonable choice, since these priors will be our closest approximation to an ideal prior that is centered exactly at the optima, where sampling from the prior always leads to the optimum. Our results in Figure 5.12 show that this combination leads to the best results in all benchmarks.



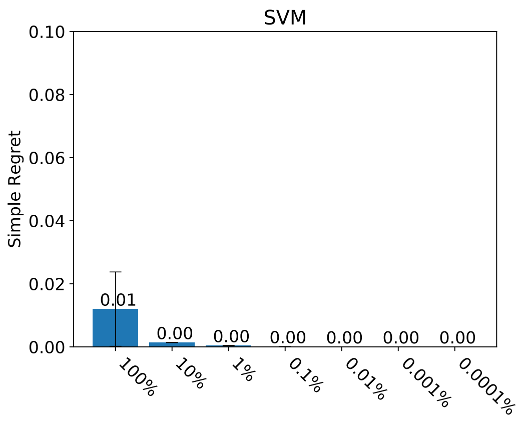
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$

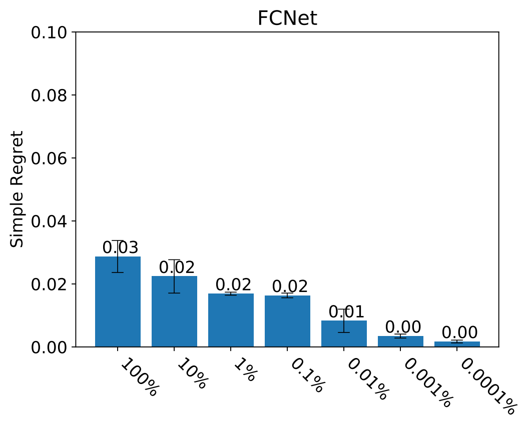


(c) $a = 10, b = 0$

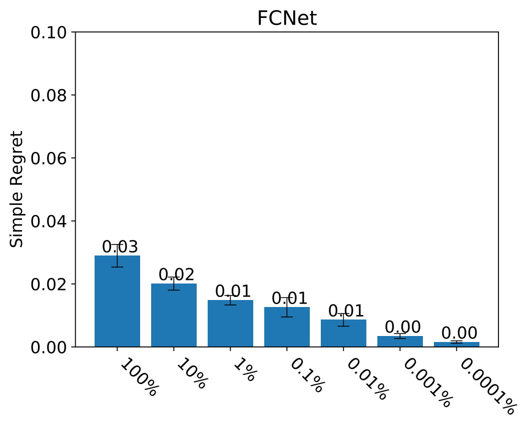


(d) $a = 100, b = 0$

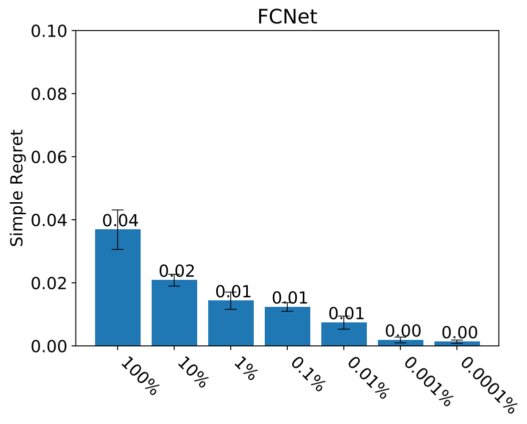
Figure A.2. Simple regret of sampling from the prior with different priors for our SVM benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.



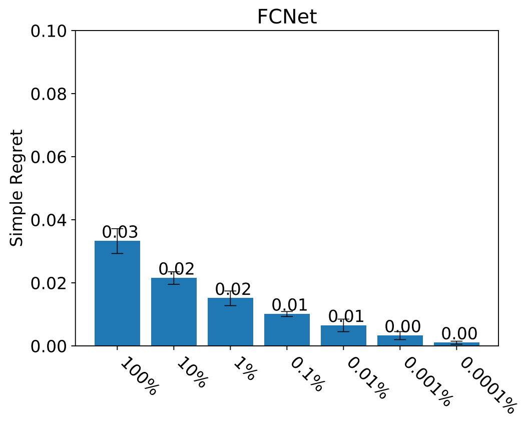
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$

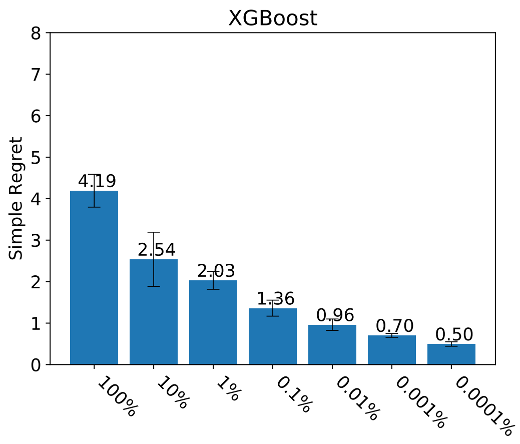


(c) $a = 10, b = 0$

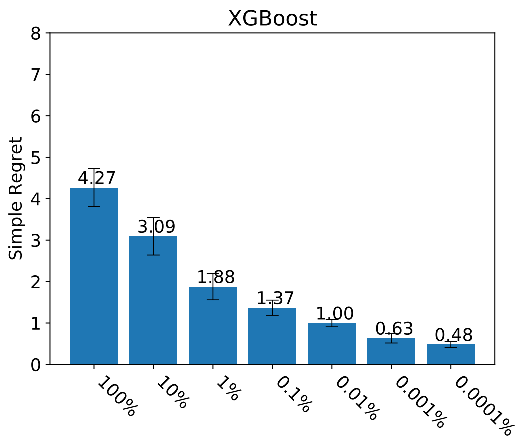


(d) $a = 100, b = 0$

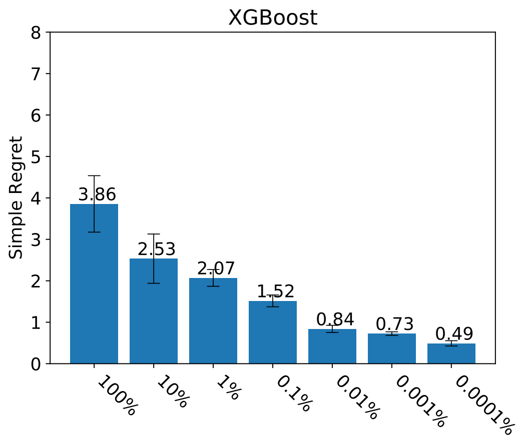
Figure A.3. Simple regret of sampling from the prior with different priors for our FCNet benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.



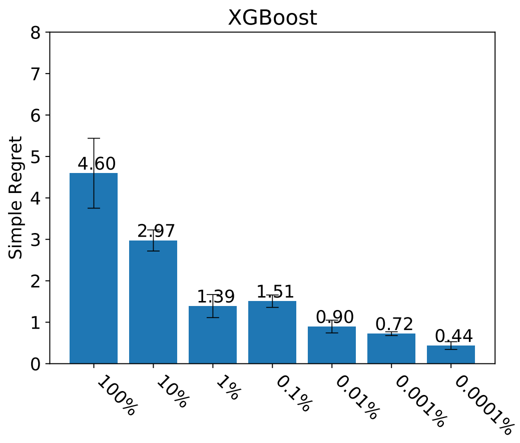
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$

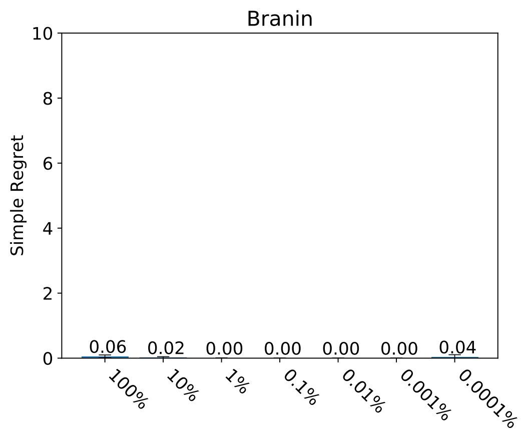


(c) $a = 10, b = 0$

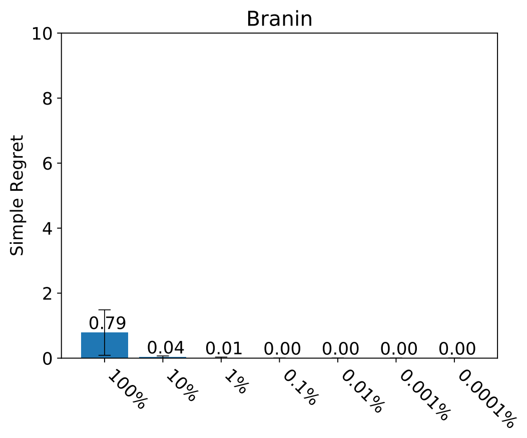


(d) $a = 100, b = 0$

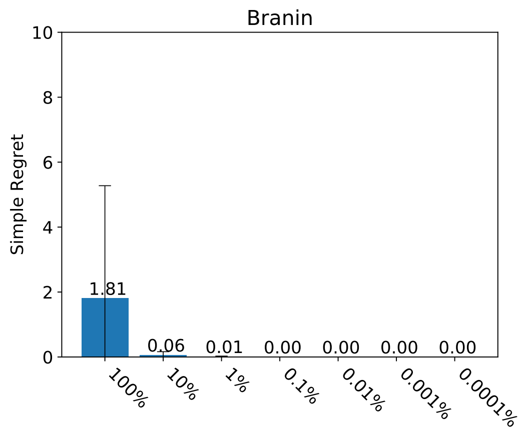
Figure A.4. Simple regret of sampling from the prior with different priors for our XGBoost benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars.



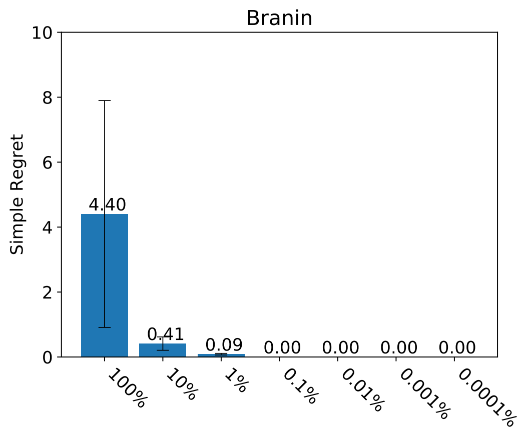
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$

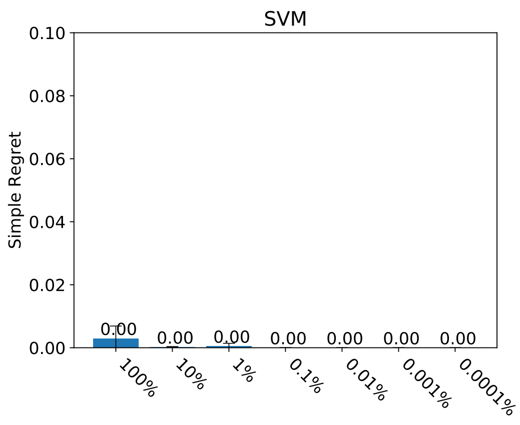


(c) $a = 10, b = 0$

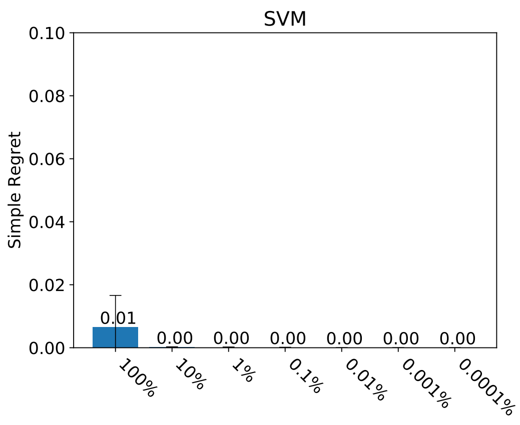


(d) $a = 100, b = 0$

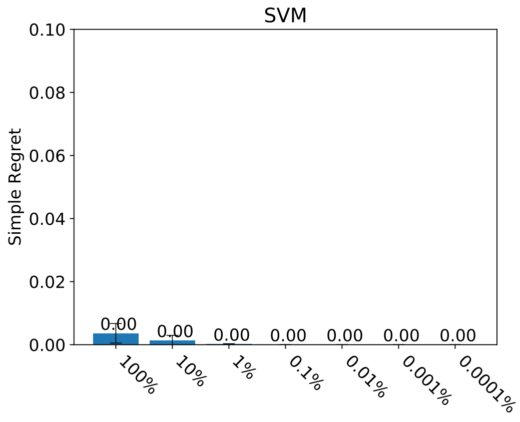
Figure A.5. Simple regret of BOPrO with different priors for our Branin benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.



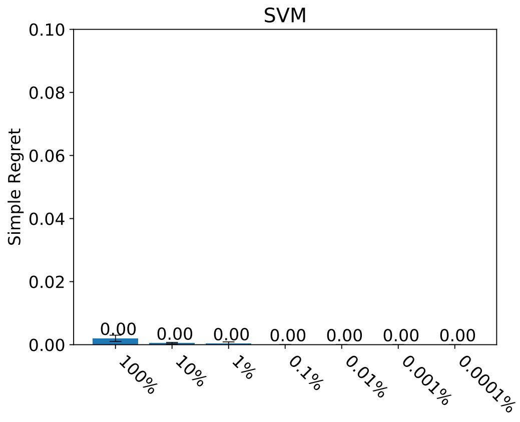
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$

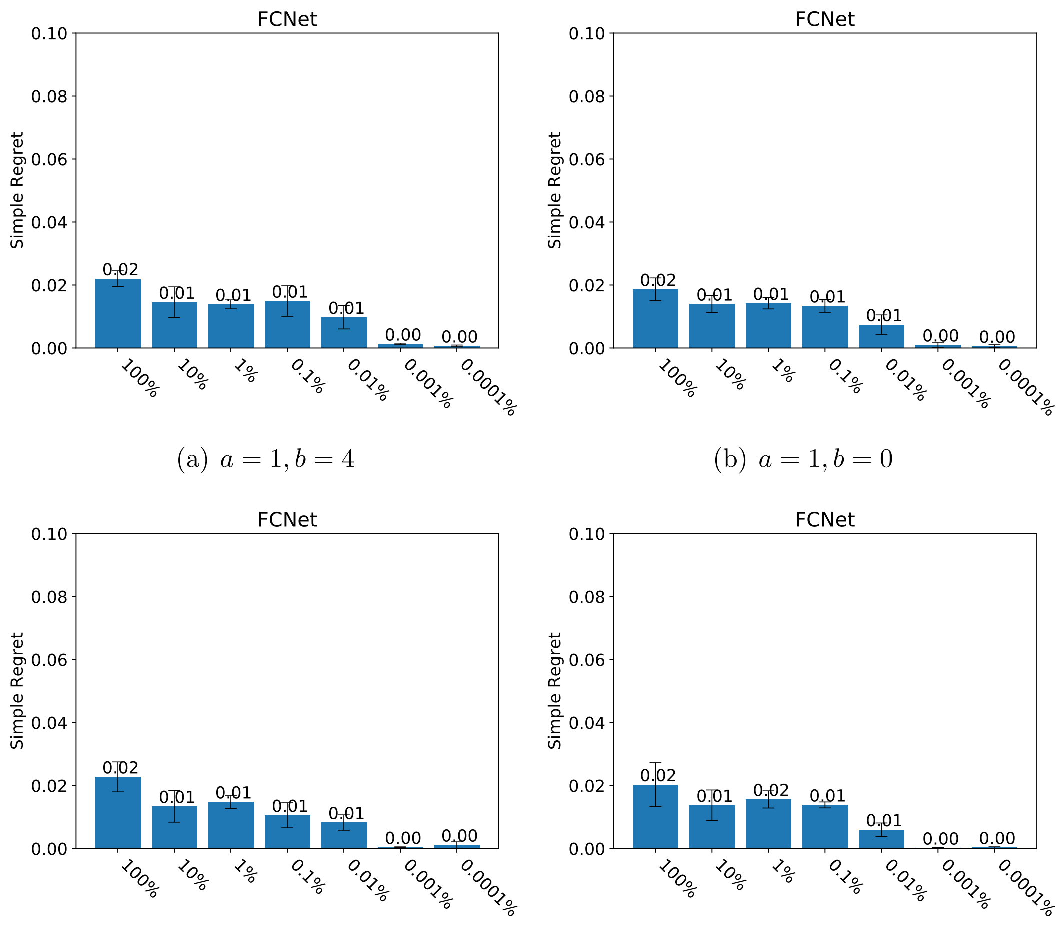


(c) $a = 10, b = 0$



(d) $a = 100, b = 0$

Figure A.6. Simple regret of BOPrO with different priors for our SVM benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.



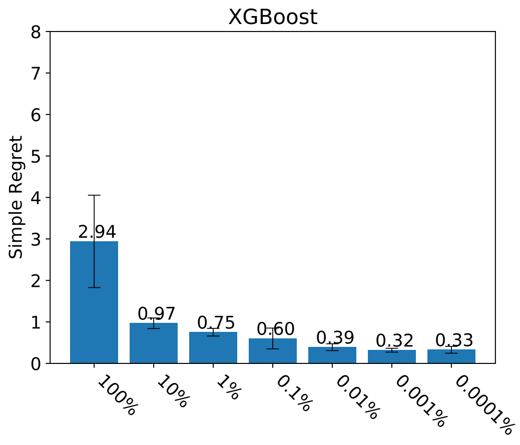
(a) $a = 1, b = 4$

(b) $a = 1, b = 0$

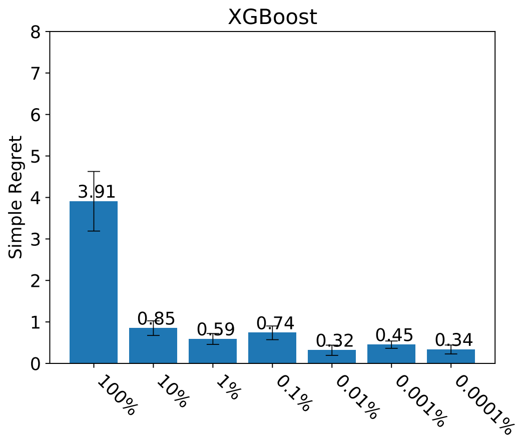
(c) $a = 10, b = 0$

(d) $a = 100, b = 0$

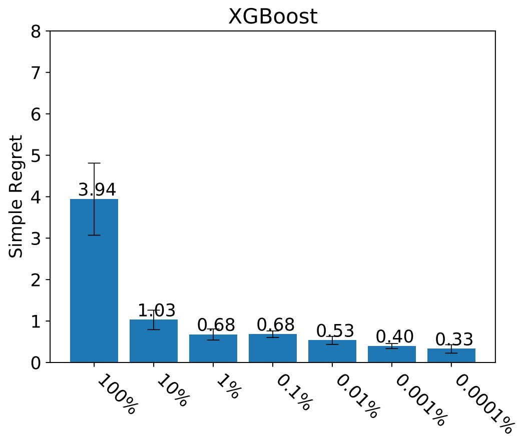
Figure A.7. Simple regret of BOPrO with different priors for our FCNet benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.



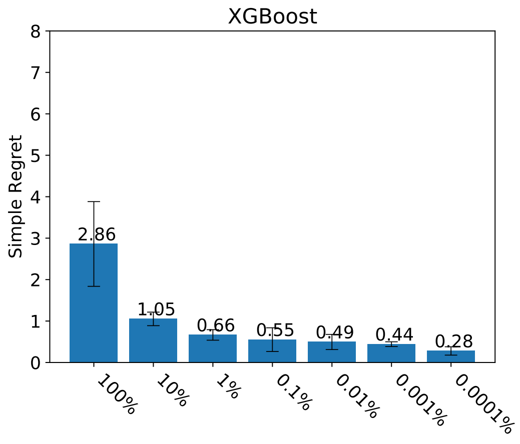
(a) $a = 1, b = 4$



(b) $a = 1, b = 0$



(c) $a = 10, b = 0$



(d) $a = 100, b = 0$

Figure A.8. Simple regret of BOPrO with different priors for our XGBoost benchmark. We provide 5 repetitions for each experiment and mean +/- one std error bars. A more informative prior gives better results.