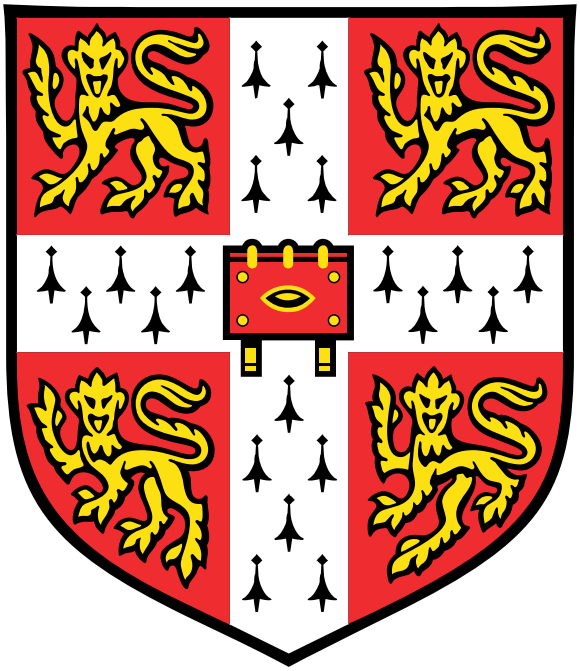
Parameter Optimization using high-dimensional Bayesian Optimization

Bachelor Thesis



David Yenicelik

Department of Computer Science
Swiss Federal Institute of Technology in Zurich

This dissertation is submitted for the degree of

July 2018

0.1 Project Proposal for Bachelor Thesis

0.1.1 Motivation

Tuning hyperparameters is considered a computationally intensive and tedious task, be it for neural networks, or complex physical instruments such as free electron lasers. Users for such applications could benefit from a 'one-click-search' feature, which would find optimal parameters in as few function evaluations as possible. This project aims to find such an algorithm which is both efficient and holds certain convergence guarantees. We focus our efforts on Bayesian Optimization (BO) and revise techniques for high-dimensional BO.

0.1.2 Background

In Bayesian Optimization, we want to use a Gaussian Process to find an optimal parameter setting $\mathbf{x}^*$ that maximizes a given utility function f. We assume the response surface to be Lipschitz-continuous.

Assume we have observations $\mathscr{Y} = \{y^{(1)}, \ldots, y^{(N)}\}$, each evaluated at a point $\mathscr{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$. The relationship between the observations y and individual parameter settings $\mathbf{x}$ is $y = f(\mathbf{x}) + \varepsilon$ where $\varepsilon \sim \mathscr{N}\left(0, \sigma_n^2\right)$. Any quantity to be predicted has a subscript-star (e.g. y_* is the function evaluation we want to predict).

In it's simplest form, a Gaussian Process is described by the following equation:

$$\begin{pmatrix} y \\ y_* \end{pmatrix} \sim N\left(\mu, \begin{pmatrix} K & K_*^T \\ K_* & K_{**} \end{pmatrix}\right), \tag{1}$$

Where μ is a mean function, $K = \text{kernel}(\mathbf{X}, \mathbf{X})$, $K_* = \text{kernel}(\mathbf{x}_*, \mathbf{X})$ and $K_{**} = \text{kernel}(\mathbf{x}_*, \mathbf{x}_*)$. We predict any new point y_*, (given all previously sampled points y) by estimating the probability $p(y_*|y) \sim N(K_* K^{-1} y, K_{**} - K_* K^{-1} K_*')$

This, in turn, can be used to build an acquisition function. This acquisition function describes where to best sample points next. Some popular acquisition functions include GP-UCB, Most probable improvement (MPI) and Expected Improvement (EI). The choice of the acquisition function has great influence on the performance of the optimization procedure.

We will talk about the problems and possible solutions for the task at hand in the next section.

0.1.3 Scope of the Project

Bayesian optimization suffers from the curse of dimensionality. The goal of this project is to arrive at a solution that resolves the curse of dimensionality for the specific task with regards to Bayesian optimization. This project includes, but is not limited to the following methods.

1. [16] Assume $f(x) \approx g(\mathbf{W}^T x)$ where $\mathbf{W} \in \mathbb{R}^{D \times d}$ and $D >> d$. We assume that $\mathbf{W}$ is orthogonal.

   This algorithm does not require gradient-information (thus, easier to implement, and robust to noise). The standard-deviation, kernel parameters and $\mathbf{W}$ can be found iteratively. First we fix $\mathbf{W}$, and optimize over the standard-deviation, kernel parameters. Then we fix the standard-deviation, kernel parameters. and optimize over $\mathbf{W}$. We repeat this procedure until the change of the log-likelihood between iterations is below some ε_l.

2. [14] Assume $f(x) = f^{(1)}(x^{(1)}) + f^{(2)}(x^{(2)}) + \ldots + f^{(M)}(x^{(M)})$ where $x^{(i)} \in \mathscr{X}^{(i)} \subseteq \mathscr{X}$, i.e. each function component $f^{(i)}$ takes some lower-dimensional subspace as the input. The lower-dimensional subspaces may overlap. The mean and covariance of $f(x)$ is then the sum of the individual component's means and covariances.

   An additive decomposition (as described above) can be represented by a dependency graph. The dependency graph is built by joining variables i and j with an edge whenever they appear together within some set $x(k)$.

   The goal is to maximize an acquisition function $\phi_t(x) = \sum_{i=1}^{M} \phi_t^{(i)}(x^{(i)})$. This maximization is achieved by maximizing the probability of Markov Random Fields within the graph. A junction tree is created from the graph, which is then used to find the global maximum of the acquisition function.

   The dependencies between the variable-subsets are represented through a graph, which can be learned through Gibbs sampling. This, in turn, is used to create a kernel for the GP.

3. [**?**] A function $f : \mathbf{R}^D \to \mathbf{R}$ is said to have effective dimensionality d_e (where $d_e < D$), if there exists a linear subspace $\mathscr{T}$ of dimension d_e such that for all $x_\top \in \mathscr{T} \subset \mathbf{R}^D$ and $x_\perp \in \mathscr{T}_\perp \subset \mathbf{R}^D$, we have $f(x) = f(x_\top + x_\perp) = f(x_\top)$. $\mathscr{T}^\perp$ is the orthogonal complement of $\mathscr{T}$.

   Assume $f : \mathbf{R}^D \to \mathbf{R}$ has effective dimensionality d_e. Given a random matrix $\mathbf{A} \in \mathbf{R}^{D \times d}$ (where $d \geq d_e$) with independent entries sampled fom $\mathscr{N}(0,1)$. For any $x \in \mathbf{R}^D$, there

exists a $y \in \mathbf{R}^d$ such that $f(x) = f(\mathbf{A}y)$. We now only need to optimize over all possible $y \in \mathbf{R}^d$, instead of all possible $x \in \mathbf{R}^D$.

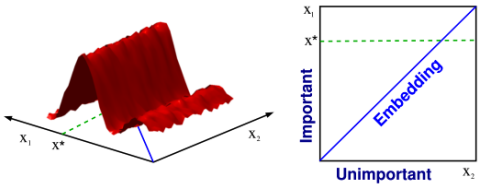


Fig. 1 This function in D=2 dimesions only has d=1 effective dimension. Hence, the 1-dimensional embedding includes the 2-dimensional function's optimizer. It is more efficient to search for the optimum along the 1-dimensional random embedding than in the original 2-dimensional space

If, for some reason, finding an active subspace or an effective lower dimension is not possible, we are open to adapt the procedure of optimization.

Abstract

In this thesis, I explore the possibilities of conducting bayesian optimization techniques in high dimensional domains. Although high dimensional domains can be defined to be between hundreds and thousands of dimensions, we will primarily focus on problem settings that occur between two and 20 dimensions. As such, we focus on solutions to practical problems, such as tuning the parameters for an electron accelerator, or for even simpler tasks that can be run and optimizated just in time with a common laptop at hand.

I follow a systematic methodology, where we identify problems currently occurring with Bayesian Optimization methods. For this, I take on the following steps:

I first provide a theoretic background to the mathematical foundations of Gaussian Processes at hand. I present the derivation of the surrogate functions as a foundation to better understand what parts of the process can be optimized. I then shortly discuss the different acquisition functions that are most often used in practice.

I continue by exploring different techniques that are currently considered as state-of-the-art. Most of these techniques concentrate on doing one of three things. The most common approach is to do a linear dimensionality reduction. The second most common approach is to identify variables that rely on each other, and create different Gaussian Processes for each group of variables that depend on each other.

I identify the shortcomings of the current methods and present quantitative ways on how we can measure improvements for the goal at hand. Together with my supervisors, we present a novel algorithm called "BORING" to do Bayesian Optimization at hand. This algorithm combines the advantages of linear projection methods, but also allows to take into consideration some small perturbations in the target function. Taking into account smaller perturbations is something that linear reduction models have not taken into account so far. BORING proves to have all the advantages that other state-of-the-art linear dimensionality reduction algorithms have. However, BORING improves this state by allowing small pertur-

bations to be taken into account when creating the surrogate function. This feature ultimately allows BORING to be more accurate when modelling functions that we want to optimize over.

Finally, we evaluate BORING and compare it to the competitive state-of-the-art in Bayesian Optimization. We do a quantitative analysis by applying the Gaussian Process algorithms on different, well defined functions and measuring the regret during optimization that it achieves. We do a qualitative analysis by visualizing the predicted surrogate function, and comparing this to the real function. In addition to that, we do a short experiment on whether the subspace identification method can be used to do feature selection, by choosing the projection in such a way, that the most important features will receive the highest matrix weights.

Our main contribution is BORING, an algorithm which is competitive with other state-of-the-art methods in Bayesian Optimization. The main features of BORING are 1.) the possibility to identify the subspace (unlike most other optimization algorithms), and 2.) to provide a much lower penalty to identify the subspace, as optimization is still the main goal.

Table of contents

List of figures

List of tables

Chapter 1

Background

1.1 Bayesian Optimization in high dimensions

Many technical problems can be boiled down to some flavour of black box optimization. Such problems include neural architecture search [9], hyper-parameter search for neural networks, parameter optimization for electron accelerators, or drone parameter optimization using safety constraints [2].

Bayesian optimization methods are a class of sequential black box optimization methods. A surrogate function surface is learned using a Gaussian prior, and a Gaussian likelihood function. Combining the prior and the likelihood results in the Gaussian Posterior, which can then be used as a surface over which optimization can take place, with the help of a chosen acquisition function.

Bayesian optimization is a method that has increasingly gained attention in the last decades, as it requires relatively few points to find an appropriate response surface for the function over which we want to optimize over. It is a sequential model based optimization function, which means that we choose the best point x_i^* given all previous points $x_{i-1}^*, x_{i-2}^*, \ldots, x_0^*$. Given certain acquisition functions, it offers a good mixture between exploration and exploitation from an empirical standpoint.

Bayesian optimization is a method that has increasingly gained popularity, as it shows empirical success in finding a good mixture between exploration the search space (finding new hyper-parameter configurations that outperform all existing configurations), and exploiting currently found best configurations (using the best learned hyper-parameter configurations to

get utility out of the system) [1].

However, as machine learning algorithms, and other problems become more complex, bayesian optimization needs to cope with the increasing number of dimensions that define the search space of possible configurations. Because BO methods lose effectiveness in higher dimensions due to the loss of information through spatial locality, this thesis work explores methods that work in higher dimensions. Finally, we propose a novel method that improves on certain characteristics of the current state-of-the-art.

1.2 Gaussian Processes

Bayesian Optimization aims at using a Gaussian Process as an intermediate representation to find an optimal parameter setting $\mathbf{x}^*$ that maximizes a given utility function f. We assume the response surface to be Lipschitz-continuous.

Assume we have observations $\mathscr{Y} = \{y^{(1)}, \ldots, y^{(N)}\}$, each evaluated at a point $\mathscr{X} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$. The relationship between the observations y and individual parameter settings $\mathbf{x}$ is $y = f(\mathbf{x}) + \varepsilon$ where $\varepsilon \sim \mathscr{N}\left(0, \sigma_n^2\right)$. Any quantity to be predicted has a subscript-star (e.g. y_* is the function evaluation we want to predict).

In it's simplest form, a Gaussian Process is described by the following equation:

$$\begin{pmatrix} y \\ y_* \end{pmatrix} \sim N\left(\mu, \begin{pmatrix} K & K_*^T \\ K_* & K_{**} \end{pmatrix}\right), \tag{1.1}$$

Where μ is a mean function, $K = \text{kernel}(\mathbf{X}, \mathbf{X})$, $K_* = \text{kernel}(\mathbf{x}_*, \mathbf{X})$ and $K_{**} = \text{kernel}(\mathbf{x}_*, \mathbf{x}_*)$. Any new point y_* is predicted given all previously sampled points y by estimating the probability $p(y_*|y) \sim N(K_* K^{-1} y, K_{**} - K_* K^{-1} K_*')$

This, in turn, can be used to build an acquisition function. This acquisition function describes where to best sample points next. Some popular acquisition functions include GP-UCB, Most probable improvement (MPI) and Expected Improvement (EI). The choice of the acquisition function has great influence on the performance of the optimization procedure.

In the following, I provide a short derivation of the core formulae used for Gaussian Processes.

1.2.1 Derivation of the Gaussian Process Formula

The **prior** for the Gaussian Process is the following (assuming the commonly chosen 0-mean-prior).

$$u \sim GP(0, k(x, x')) \tag{1.2}$$

Because u is a random variable following a Gaussian Process distribution, its probably distribution is given by a normal distribution:

$$p(u) = N(0, K) \tag{1.3}$$

Additional data $X = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ can be observed. The Gaussian Process incorporates an error term that takes into consideration possible noise in the measurement of the experiment. Thus, y has some noise term ε such that $y = u(x) + \varepsilon$. By the central limit theorem, commonly the assumption is held that ε is normally distributed around 0 with σ_s standard deviation. Given the sampled datapoints, and the inherent noise that these datapoints have, the **likelihood** of the Gaussian Process can be represented as follows:

$$p(y|x, u) = N(u, \sigma_s^2 I) \tag{1.4}$$

Given the prior and likelihood of the Gaussian Process, the **posterior** of the Gaussian Process can be derived by simple application of Bayes rule.

$$p(u|x, y) = \frac{p(y|x, u)p(u)}{p(y|x)} \qquad = N(K(K + \sigma^2 I)^{-1} y, \sigma^2 (K + \sigma^2 I)^{-1} K) \tag{1.5}$$

From the above posterior, we now want to predict for an arbitrary x_* the function value y_*. Predicting y_* for every possible x_* in the domain results in the surrogate response surface that the GP models.

We assume that the value y_* we want to predict is also distributed as a Gaussian probability distribution. Because the y_* that we want to predict relies on all the values collected in the past (which are again normally distributed), the probability distribution can be modelled as jointly Gaussian.

$$\begin{pmatrix} y \\ y_* \end{pmatrix} \sim N\left(\mu, \begin{pmatrix} K & K_*^T \\ K_* & K_{**} \end{pmatrix}\right), \tag{1.6}$$

To compute this equation, we use the results from Murphy's textbook [12] pages 110 to 111 to do inference in a joint Gaussian model.

1.3 Acquisition Functions

Given the above formula for the posterior mean μ and the poster variance σ^2, Bayesian Optimization makes use of an acquisition function. The following is a summary of the most popular acquisition functions in current literature.

1.3.1 Upper Confident Bound (UCB)

[15] show a proof, that for a certain tuning of the parameter β, the acquisition function has certain asymptotic regret bounds.

The upper confidence bound allows the user to control exploitation and exploration through a parameter $\beta > 0$, which can be chosen as specified in [15] , to offer regret bounds. In addition to that, GP-UCB shows state-of-the-art empirical performance [4].

$$UCB(x) = \mu(x) + \sqrt{\beta}\sigma(x) \tag{1.7}$$

Here, the functions μ and σ are the predicted mean and variance of the Gaussian Process Posterior.

1.3.2 Probability of Improvement (PI)

The (maximum) probability of improvement [3] always selects the points where the mean plus uncertainty is above the maximum explored function threshold. The downside to this policy is that this leads to heavy exploitation. However, the rate of exploitation can be controlled by a parameter $\xi > 0$.

$$PI(x) = P(f(x) \geq f(x^+) + \xi) \tag{1.8}$$

$$= \Phi(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}) \tag{1.9}$$

1.3.3 Expected Improvement (EI)

As an improvement to the maximum probability of improvement, the expected improvement takes into consideration not only the probability that a point can improve the maximum found

so far. But that the EI also takes into account the magnitude by which it can improve the maximum function value [3]. As in MPI, one can control the rate of exploitation by setting the parameter $\xi > 0$, which was introduced by [11].

$$EI(x) = \begin{cases} (\mu(x) - f(x^+) - \xi)\Phi(Z) + \sigma(x)\phi(Z) & \text{if } \sigma(x) > 0 \\ 0 & \text{if } \sigma(x) = 0 \end{cases} \tag{1.10}$$

$$\tag{1.11}$$

where

$$Z = \frac{\mu(x) - f(x^+) - \xi}{\sigma(x)} \tag{1.12}$$

and where ϕ denotes the PDF, and Φ denotes the CDF of the standard normal distribution respectively.

Given one of the above acquisition functions, one can then use an optimizer such as *LBFGS* or monte carlo sampling methods, to find an approximate global maximum of the respective function. The combination of Gaussian Processes and Acquisition function together result in a Bayesian Optimization algorithm, which has a prior assumption about the function to be learned, and uses data-samples to create a likelihood to further refine the posterior of the initial function assumption.

1.4 Resources

For the experiments and code basis, most of our functions rely on the Sheffield Machine Learning - GPy library [7]. In addition to that, the febo framework developed by Johannes Kirschner from the Learning and Adaptive Systems group at ETH Zurich.

Chapter 2

Related Work

This section covers current methods solving Bayesian Optimization in high dimensions that are considered state-of-the-art. This is not an exhaustive review. For each algorithm, I discuss the effectiveness with regards to the dataset or function that the respective paper does the evaluation on.

Conventions that I use is the following:

1. f is the real function to be approximated / optimized.
2. g and any subscripted or superscripted derivative of g is a component that one uses to approximate f.
3. Anything that has a "hat" on (caret symbol on f is $\hat{f}$), refers to an empirical estimate.

I will focus on three categories of Bayesian Optimization algorithms: Algorithms that make use of a projection matrix, algorithms that, algorithms that exploit additive substructures and "additional approaches" that are uncategorised.

2.1 Projection matrix based algorithms

Generally, for this family of algorithms, the approximation is a function of $f(x) \sim g(Ax)$, where the properties and identification of A are algorithmic specific.

2.1.1 Active learning of linear subspace

[6] The assumption of this algorithm is that f depends only on $x := uR^T$ with $R \in \mathbf{R}^{d \times D}$, $u \in \mathbf{R}^d$ and where $d << D$. The algorithm learns a projection matrix R and the surrogate

Algorithm 1 Simultaneous active learning of functions and their linear embeddings (pseudocode) :: Active learning of linear subspace [6]

Require: d, D; kernel κ, mean function μ; prior $p(R)$
  $X \leftarrow \emptyset$
  $Y \leftarrow \emptyset$
  while budget not depleted **do**
    $q(R) \leftarrow \text{LAPLACEAPPROX}(p(R|X,Y,\kappa,\mu))$
    $q(f) \leftarrow APPROXMARGINAL(p(f|R), q(R))$
    $x_* \leftarrow OPTIMIZEUTILITY(q(f), q(R))$
    $y \leftarrow OBSERVE(f(x_*))$
    $X \leftarrow [X; x_*]$
    $Y \leftarrow [Y; y_*]$
  end while
  return $q(R), q(f)$

function$g(u)$, with $f(x) \sim g(u)$.

The **Laplace approximation** for R is using the mode of the probability distribution $\log P(R|D)$ as a mean, and the covariance is taken as the inverse Hessian of the negative logarithm of the posterior evaluated at the mean of the distribution. Together, this describes the probability distribution $p(R|X,Y,\kappa,\mu)$, where μ is the mean function, and κ is the covariance function.

The **approximate marginal** subroutine is a novel method proposed in the paper that integrates over the different parameters in the paper. This marginal approximation does a local expansion of $q(x_*|\theta)$ to $p(x_*|D,\theta)$.

The **optimization of utility** (choice of next best point) is done using Bayesian Active Learning by disagreement, where the utility function is the expected reduction in the mutual information, as opposed to uncertainty sampling reducing entropy, which is not well defined for all values.

The metrics used in this paper are negative log-likelihoods for the test points, and the mean symmetric kullback leiber divergence between approximate and true posteriors. The proposed method always outperforms the naive MAP method for the presented functions close to a factor of 2 for both loss functions. Tests are conducted on a real, and synthetic dataset with up to $D = 318$ and selecting $N = 100$ observations.

2.1.2 High dimensional Gaussian bandits

[4] This model assumes that there exists a function $g : \mathbf{R}^k \implies [0,1]$ and a matrix $A \in \mathbf{R}^{d \times D}$ with orthogonal rows, such that $f(x) \sim g(Ax)$. Assume $g \in \mathscr{C}^2$.

Algorithm 2 The SI-BO algorithm [4]

Require: $m_X, m_\Phi, \lambda, \varepsilon, k$, oracle for the function f, kernel κ
  $C \leftarrow m_X$ samples uniformly from $\mathbb{S}^{d-1}$
  for $i \leftarrow 1$ to m_X **do**
    $\Phi_i \leftarrow m_\Phi$ samples uniformly from $\{-\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}\}^k$
  end for
  $y \leftarrow$ (compute using Equation 1 – INSERT Eq 1 here, or create a summary of all important equations here)
  select z_i according to a UCB acquisition function, evaluate f on it, and add it to the datasamples found so far

The SI-BO algorithm consists of a subspace identification step, and an optimization step using GP-UCB.

The **subspace identification** is treated as a low-rank matrix recovery problem as presentedin [**?**].

2.1.3 Random embeddings (REMBO)

[18] Let $x \in \mathbb{R}^D$ and $y \in \mathbb{R}^d$. Assume, that $f(x) = f(Ax)$. We can generate $A \in \mathbb{R}^{D \times d}$ by randomly generating this matrix.

2.1.4 Applications to high-dimensional uncertainty propogation

Because the algorithm we propose uses this method to identify the subspace projection matrix, I will discuss this algorithm in more detail.

[16] This algorithm assumes that $f(x) \sim g(\mathbf{W}^T y)$ where $\mathbf{W} \in \mathbb{R}^{D \times d}$ and $D >> d$. $\mathbf{W}$ is orthogonal.

This algorithm does not require gradient-information. This makes it easier to implement, and more robust to noise according to the authors of this paper.
The GP nosie variance, kernel parameters and $\mathbf{W}$ can be found iteratively. The main idea of the algorithm is to fix the kernel parameters and the GP nosie variance, and identify $\mathbf{W}$. Then, we fix $\mathbf{W}$ and train over the kernel parameters and the GP nosie variance. This procedure is repeated until the change of the log-likelihood between iterations is below some ε_l, or if the

maximum number of steps of optimization is reached. We repeat this procedure .

I now proceed with a more detailed description of the algorithm. The quantities of interest are

$$\mu_f = \int f(x)p(x)dx \tag{2.1}$$

$$\sigma_f^2 = \int (f(x) - \mu_f)^2 p(x)dx \tag{2.2}$$

$$f \sim p(f) = \int \delta(f - f(x))p(x)dx \tag{2.3}$$

The authors argue that w.l.o.g., one can regard only on orthogonal matrices, as this does not restrict the search space of projection matrices, the main reason being that no projection subspace is neglected by doing this. Here, the family of orthogonal matrices of dimension $d \times D$ is denoted by $\mathbf{W} \in V_d(\mathbb{R}^D)$. This quantity is also known as the **Stiefel manifold**, where d is the found effective dimension of the function, and D is the real input dimension to the function.

Kernel used

As in the paper, I use the Matern-32 kernel function. This function has two input vectors a and b.

$$K(a,b,\theta) = s^2 \left(1 + \sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) exp\left(-\sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) \tag{2.4}$$

Because I want to avoid numerical testing and implementation, I use the derivative as provided in the GPy library. The $s, l_1, \ldots, l_l$ are hyper-parameters of the kernel. The concatenated vector of all these kernel hyperparameters is denoted by θ.

The only modification made to this kernel is the additional parameter W:

$$k_{AS} : \mathbb{R}^D \times \mathbb{R}^D \times V_d(\mathbb{R}^D) \times \phi -> \mathbb{R} \tag{2.5}$$

where the kernel has the form

$$k_{AS}(x,x';W,\phi) = k_d(W^T x, W^T x';\phi) \tag{2.6}$$

Step 1.: Determine the active projection matrix W

. In this step, the algorithm optimizes $W \in V_d(\mathbb{R}^D)$ while keeping the kernel hyperparameters ϕ and the GP noise variance s_n fixed.

To simplify calculations later, we define the function, where all parameters but W are fixed as F. The other parameters are determined from previous runs, or are freshly sampled:

$$\begin{aligned} F(W) &:= \mathscr{L}(W,s_n;X,y) && (2.7)\\ &= \log p(y|X,W,s_n) && (2.8)\\ &= -\frac{1}{2}(y-m)^T(K+s_n^2 I_N)^{-1}(y-m) - \frac{1}{2}\log|K+s_n^2 I_N| - \frac{N}{2}\log 2\pi && (2.9)\\ & && (2.10)\end{aligned}$$

where $\phi, s_n; X, y$ are fixed and m is the prior mean function, which is 0 in our specific case.

To optimize over the loss function, the algorithm defines the derivative of F with regards to each individual element of the weights-matrix:

$$\begin{aligned} \nabla_{w_{i,j}} F(W) &:= \nabla_{w_{i,j}} \mathscr{L}(W,s_n;X,y) && (2.11)\\ &= \frac{1}{2}\mathrm{tr}\left[\{(K+s_n^2 I_N)^{-1}(y-m)\left((K+s_n^2 I_N)^{-1}(y-m)\right)^T - (K+s_n^2 I_N)^{-1}\}\nabla_{w_{i,j}}(K+s_n^2 I_N)\right] && (2.12)\end{aligned}$$

both these functions depend on the kernel K, and it's derivative $\nabla_{w_{i,j}} K$.

To optimize over F, a more sophisticated algorithm is used, that resembles iterative hill-climbing algorithms. First, the paper defines the function whose output is a matrix in the Stiefel manifold

$$\gamma(\tau;W) = (I_D - \frac{\tau}{2}A(W))^{-1}(I_D + \frac{\tau}{2}A(W))W \tag{2.13}$$

where W is a fix parameter, and τ is the variable which modifies the direction that we move on in the Stiefel manifold and with

$$A(W) = \nabla_W F(W) W - W(\nabla_W F(W))^T \tag{2.14}$$

One iteratively chooses fixed W, and does a grid-search over τ such that at each step, the log-likelihood $\mathscr{L}$ is increased.

Step 2.: Optimizing over GP noise variance and the kernel hyperparameters

We determine the hyperparameters by optimizing over the following loss function, where X are the input values, Y are the corresponding output samples. ϕ is the concatenated vector of the kernel hyperparameters and the GP noise variance.

One keeps the W fixed (either by taking W from the last iteration, or freshly sampling it), and then defines the loss function

$$L(\phi) = \mathscr{L}(W, \phi, \sigma_n; X, y) \tag{2.15}$$

To optimize this loss function, a simple optimization algorithm such as $L-BFGS$ is used to individually maximize each element of the hyperparameter vector with regards to the log-likelihood. This is done for a maximum number of steps, or until the change of improvement becomes marginal.

Additional details

Because initialization is a major factor in this algorithm, these steps are iteratively applied for many hundred steps. There are also many tens or hundreds of restarts to ensure that the search on the Stiefel manifold results in a good local optimum, and does not get stuck on a flat region with no improvement. This algorithm is very sensitive on the initially sampled parameter W.

Identification of active subspace dimension

One cannot know the real active dimension of a problem that one does not know the solution to. As such, the proposed to apply the above algorithms iteratively by increasing the selected active dimension d. The moment where the relative change between the best found matrix between two iterations is below a relative threshold ε_s, the previous active dimension is chosen as the real active dimension.

2.2 Algorithms that exploit additive substructures

Functions with additive substructures can be decomposed into a summation over subfunctions, such that $f(x) \sim g_0(x) + g_1(x) + \dots g_2(x)$ where each g_i may operate only on a subset of dimensions of x.

2.2.1 Independent additive structures within the target function

[5] Assume that $f(x) = \sum_{i=1}^{|P|} f_i(x[P_i])$, i.e. f is fully additive, and can be represented as a sum of smaller-dimensional functions f_i, each of which accepts a subset of the input-variables. The kernel also results in an additive structure: $f(x) = \sum_{i=1}^{|P|} k_i(x[P_i], x[P_i])$. The posterior is calculated using the Metropolis Hastings algorithm. The two actions for the sampling algorithm are 'Merge two subsets', and 'Split one set into two subsets'. k models are sampled, and we respectively approximate $p(f_*|D,x^*) = \frac{1}{k}\sum_{j=1}^{k} p(f(x^*|D,x,M_j))$, where M_j denotes the partition amongst all input-variables of the original function f.

2.3 Additional approaches

2.3.1 Elastic Gaussian Processes

[13] Use a process where the space is iteratively explored. The key insight here is that with low length-scales, the acquisition function is extremely flat, but with higher length-scales, the acquisition function starts to have significant gradients. The two key-steps is to 1.) additively increase the length-scale for the gaussian process if the length-scale is not maximal and if $||x_{init} - x^*|| = 0$. And 2.) exponentially decrease the length-scale for the gaussian process if the length-scale is below the optimum length-scale and if $||x_{init} - x^*|| = 0$.

2.3.2 Bayesian Optimization using Dropout

[10] propose that the assumption of an active subspace is restrictive and often not fulfilled in real-world applications. They propose three algorithms, to iteratively optimize amongst certain dimensions that are not within the d 'most influential' dimensions: 1.) Dropout Random, which picks dimensions to be optimized at random, 2.) Dropout copy, which continuous optimizing the function values from the found local optimum configuration, and 3.) which does method 1. with probability p, and else method 2. The d 'most influential' dimensions are picked at random at each iteration.

Chapter 3

Fields of Improvement

3.1 Shortcomings of current methods

I will enumerate select models from the section "related work", and will shortly discuss what the shortcomings of these models are:

REMBO is a purely optimizational algorithm, which finds optimizations in lower dimensions

- **Suitable choice of the optimization domain:** REMBO is not purely robust, as there is a considerable chance that no suitable subspace will be found. Empirically, the choice of the optimization domain heavily affects the duration and effectiveness of the optimization. I have found that the proposed optimization domain $\left[-\sqrt{d}, \sqrt{d}\right]^d$ is not well chosen for smaller environments, such as the Camelback function embedded in 5 dimensions. In any case, this is a very sensitive hyperparameter
- **Identification of subspace:** In some settings, including optimization with safety constraints, knowing the subspace that the model projects to is advantageous. REMBO is an implicit optimizer, in that it does not find any subspace, but optimizes through a randomly sampled matrix.
- **Probability of failure:** REMBO has a relatively high probability of failure. The authors propose that restarting REMBO multiple times would allow for a good optimization domain to be found, which leads to interleaved runs.

Active subgradients can be a viable option if we have access to the gradients of the problem, from which we can learn the active subspace projection matrix in the manner by using that gradient matrices.

- **Access to gradients:** For optimization algorithms, the function we want to optimize over is usually a black-box function. Practically, most black-box functions don't offer access to gradient information. To approximate the gradients using sampled points, this would require a high number of datapoints per dimension. In addition to that, these points would have to be evenly distributed, such that the gradients can be effectively estimated for more than one region.
- **Robustness to noise:** According to [16], methods that approximate gradients and use this gradient information to create subspace projectsion are very sensitive to noise. Depending on the application, this can make the algorithm ineffective as it is not robust to small variations in the response surface.

Given the nature of real-world data, approximating the active subspace using the gradients of the data-samples is thus not a robust, and viable option.

Tripathy's method argues that it is more robust to real-world noise. It also does not rely on gradient information of the response surface. Tripathy's method allows for a noise-robust way to identify the active subspace.

- **Duration of optimization:** In practice, Tripathy's method takes a long time, especially if the dimensions, or the number of data-points are high. This is due to the high number of matrix multiplications. Especially for easier problems, it is often desirable if the running time of optimizing for the next point is in a few minutes or seconds, rather than hours.
- **Efficiency:** In practice, Tripathy's method relies on a high number of restarts. From our observations, the number of steps to optimize the orthogonal matrix becomes relevant as the number of dimensions grow. Given the nature of accepting any starting point, it does not allow for a very efficient way to search for the best possible projection matrix. A more efficient way to search all possible matrices - by incorporating heuristics for example - would be desirable.
- **Insensitive to small perturbations:** Although Tripathy's model finds an active subspace, it completely neglects other dimensions which could allow for small perturbations to allow for an additional increase the global optimum value. Although we

can control to what extent small perturbations should be part of the active subdimension, one usually wants to choose a significant cutoff dimension, but still incorporate additional small perturbations without sacrificing the effectiveness of the projection.

3.2 Method of measuring improvements

In the following sections, we will discuss and show how we can improve on the shortcomings of the above methods. Because practicality is important in our method, we will both use synthetic functions to measure the efficiency of our method, but also real datasets. For real datasets, we want to see if the **???** log likelihood improves. I will use a real dataset of an electron accelerator. I will increasingly train the GP model on a number of data-points. As more data-points are added to the GP, the log likelihood of the test points should have the tendency to increase.

Besides that, we offer the following possibilities to check how well our model does.

- Test if the expectation
$$E[f(Ax) - \hat{f}(\hat{A}x)]$$
decreases / approaches zero (for methods that identify a projection matrix).
- Check if the test log-likelihood decreases for functions that are provided by a finite number of data-points.
- Check if the regret for the methods at hand are competitive, for synthetic functions.

3.2.1 Synthetic Datasets

5 dimensional function with 2 dimensional linear embedding One can evaluate synthetic functions at any point, and immediately get a regret value. The following synthetic functions cover different use cases.

2D to 1D : A simple Parabola which is embedded in a 2D space. This function is supposed to check that the complexity of the model does not hinder discovering simpler structures - in other words, the model complexity should still allow for finding simpler embeddings and functions.

5D to 2D : The Camelback function which is embedded in a 5D space. This checks if simple models can be found within certain, higher dimensional spaces.

5D to 2D : A simple exponential function which is enclosed in a sinusoidal function (with decreasing amplitude), which is embedded in a 5D space. This function measures if small perturbations are covered by the more complicated model (our proposed model).

3.2.2 Real Datasets

It is more difficult to test algorithms on real datasets, as one cannot test on a metric such as regret. However, one can train the GP on a training set, and then compare the log-likelihood on a test set.

SwissFEL dataset ... and some more

Chapter 4

Model Design and Extensions to the state of the art

Given the fields of improvements in the above section, we now propose an algorithm which addresses the majority of the issues mentioned in the above section. I will first present that algorithm, and then point out, as to why each individual concern is addressed.

4.1 The BORING Algorithm

We propose the following algorithm, called BORING. **BORING** stands for **B**ayesian **O**ptimization using **R**andom and **I**de**N**tifyable subspace **G**eneration.

The general idea of boring can be captured in one formula, where f stands for the real function that one wishes to approximate, and any subsequent function annotated by g refers to a component of the right hand side.

$$f(x) \approx g_0(Ax) + \sum_{i \in \mathbb{Z}^+}^{q} g_i(A^{\perp}x)_i) \tag{4.1}$$

Where the following variables have the following meaning

- A is the active subspace projection (an element of the stiefel manifold) learned through our algorithm, using Algorithm 1
- $A^{\perp}$ is an matrix whose subspace is orthonormal to the projection of A. We randomly generate $A^{\perp}$ using Algorithm 2.

- The subscript i in the right additive term denotes that we view each output dimension of $dot(A^{\perp}, X)$ as independent to the other output dimensions.

I will now proceed with a more detailed description.

4.1.1 Algorithm Description

Overview

We propose a novel method which is based on additive GPs and an active subspace projection matrix. We use different kinds of kernels. We want to calculate g_i and A within 4.1, such that the log-likelihood of the data we have accumulated so far is maximized. Because the term can be written as a sum of expressions, we can maximize each summand individually, which will lead to maximizing the entire expression (after we have calculate the active subspace).

The following few steps are applied after a "burn-in" phase, in which we use random sampleing to acquire new points. We do a random sampling as we want an unbiased dataset from which we can identify the active subspace projection matrix.

In simple terms, the algorithm proceeds as follows:

1. Pick the first n samples using random sampling. Do not apply UCB or any acquisition function to select the next best point yet. From the collected points, approximate the active projection matrix A using algorithm 1 from [16].

2. Generate a basis that are bi-orthonormal to every element in A. Concatenating these basis vectors $v_1, \ldots, v_{n-q}$ amongst the column-dimension gives us the passive projection matrix $A^{\perp}$.

3. Maximize the GP for each individual expression of the space within A, and parallel to that also orthogonal to A (as given by $A^{\perp}x$) individually.

This fights the curse of dimensionality, as we can freely choose $q \geq d_e$ to set the complexity of the second term while the first term still allows for creating proximity amongst different vectors by projecting the vectors onto a smaller subspace. The additive terms that get projected onto the passive subspace allows to incorporate smaller perturbations in the space orthogonal to A to occur.

Algorithm 3 BORING Alg. 1 - Bayesian Optimization using BORING

$X \leftarrow \emptyset$
$Y \leftarrow \emptyset$
{Burn in rate - don't look for a subspace for the first 50 samples}
$i \leftarrow 0$
while i < 50 **do**
  $i++$
  $x_* \leftarrow \text{argmax}_x$ acquisitionFunction$(dot(Q^{\perp}, x))$ using standard UCB over the domain of X.
  Add x_* to X and $f(x_*)$ to Y.
end while
while we can choose a next point **do**
  $A, d, \phi \leftarrow$ Calculate active subspace projection using Algorithm 2 from the paper by Tripathy.
  $A^{\perp} \leftarrow$ Generate passive subspace projection using Algorithm 3.
  $Q \leftarrow$ colwiseConcat($[A, A^{\perp}]$)
  $gp \leftarrow GP(\text{dot}(Q^T, X), Y)$
  kernel $\leftarrow$ activeKernel + $\sum_i^q$ passiveKernel$_i$
  $x_* \leftarrow \text{argmax}_x$ UCB$(dot(Q^{\perp}, x))$
  Add x_* to X and $f(x_*)$ to Y.
end while
return $A, A^{\perp}$

Where ϕ are the optimized kernel parameters for the activeKernel. The active projection matrix using the following algorithm, which is identical to the procedure described in [16]. The generation of the matrix $A^{\perp}$ is described next.

Finding a basis for the passive subspace (a subspace orthogonal to the active subspace)

$$A = \begin{bmatrix} \vdots & \vdots & & \vdots \\ a_1 & a_2 & \dots & a_{d_e} \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{4.2}$$

Given that we choose a maximal lower dimensional embedding (maximising the log-likelihood of the embedding for the given points), some other axes may be disregarded. However, the axes that are disregarded may still carry information that can make search faster or more robust.

To enable a trade-off between time and searchspace, we propose the following mechanism.

Assume an embedding maximizing 4.2 is found. Then the active subspace is characterized by it's column vector $a_1, a_2, ..., a_{d_e}$. We refer to the space spanned by these vectors as the *active subspace*.

However, we also want to address the subspace which is not addressed by the maximal embedding, which we will refer to *passive subspace*. This passive subspace is characterized by a set of vectors, that are pairwise orthogonal to all other column vectors in A, i.e. the vector space orthogonal to the active subspace spanned by the column vectors of A.

As such, we define the span of the active and passive subspace is defined by the matrix:

$$Q = \begin{bmatrix} A & A^{\perp} \end{bmatrix} \tag{4.3}$$

where $A^{\perp}$ describes the matrix that is orthogonal to the columnspace of A. For this, $A^{\perp}$ consists of any set of vectors that are orthogonal to all other vectors in A.

The vectors forming $A^{\perp}$ is generated by taking a random vector, and applying Gram Schmidt. This procedure is repeated for as many orthogonal vectors as we want. The procedure is summarised in Algorithm 3:

Algorithm 4 BORING Alg. 3 - generate orthogonal matrix to A(A, n)

Require: A a matrix to which we want to create $A^{\perp}$ for; n, the number of vectors in $A^{\perp}$.

```
normedA ← normalize each column of A
Q ← emptyMatrix() { The final concatenated Q will be A^⊥. }
for i = 1,...,n do
    i ← 0
    while True do
        i++
        q_i ← random vector with norm 1
        newBasis = apply gram schmidt single vector( [A, Q], q_i )
        if dot(normedA^T, newBasis) ≈ 0 and | newBasis | > 1e − 6 then
            Q ← colwiseConcatenate( (Q, newBasis)
            break
        end if
    end while
end for
return Q
```

Additive UCB acquisition function

Because the function is decomposed into multiple additive components, the computation of the mean and variance needs to be adapted accordingly. Refering to [14], the following method is used.

$$\mu_{t-1}^{(j)} = k^j(x_*^{(j)}, X^j)\Delta^{-1}y \tag{4.4}$$

$$\left(\sigma_{t-1}^{(j)}\right)^2 = k^j(x_*^j, x_*^j) - k^j(x_*^j, X^{(j)})\Delta^{-1}k^j(X^{(j)}, x_*^j) \tag{4.5}$$

where $k(a,b)$ is the piecewise kernel operator for vectors or matrices a and b and $\Delta = k(X,X) + \eta I_n$. A single GP with multiple kernels (where each kernel handles a different dimension of $dot(Q^T, x)$) is used. There are $k^{j=1,\ldots,q+1}$ kernels (the $+1$ comes from the first kernel being the kernel for the active subspace).

Using this information about each individual kernel component results in the simple additive mean and covariance functions, which can then be used for optimization by UCB:

$$\mu(x) = \sum_{i=1}^{M} \mu^{(i)}(x^{(i)}) \tag{4.6}$$

$$\kappa(x, x') = \sum_{i=1}^{M} \kappa^{(i)}(x^{(i)}, x'^{(i)}) \tag{4.7}$$

–> Possible extension to the Projection pursuit regression?

How does our algorithm address the shortcomings from chapter 3?

1. Our algorithm intrinsically uses multiple restarts, firstly proposed as an extension to REMBO. This makes training more reliable.
2. Our algorithm allows to not only optimize on a given domain, but also identify the subspace on which the maximal embedding is allocated on.
3. Our algorithms uses a "burn-in-rate" for the first few samples, which allows for efficient point search at the beginning, and later on switches to finding the actual, real subspace.
4. Our algorithm has faster convergence onto the real subspace, as we use the most promising models during optimization, instead of going through all possible restarts.

(Approximate enhancement of loss by taking sum of last 10 losses - extrapolate, and compare it to the best example so far!)

5. Our algorithm is more accurate, as we don't assume that there is a singular maximal subspace. We also take into consideration that there might be perturbation on lower dimensions!

Chapter 5

Evaluation

5.1 Evaluation Settings

Appendix A presents a list of synthetic functions and real datasets that are used to evaluate the effectiveness of a bayesian optimization algorithm. We will briefly go over some synthetic functions that we will be using for the evaluation of our algorithm. Furthermore, we will shortly discuss the underlying real dataset, which is hyper-parameter-configurations from the SwissFEL x-ray laser.

5.2 Quantitative evaluation

We will conduct experiments in the following settings (as mentioned in some other chapter REEEE):

1. a 2D embedded function in a 3D (synthetic).
2. a 2D embedded function in a 10D (synthetic).
3. a 5D embedded function in a 25D setting (synthetic).
4. A function that has the exact same structure that we propose.
5. SwissFEL data (5D real dimensional domain).

5.3 Qualitative evaluation

5.3.1 Finding a matrix when we apply a polynomial kernel onto the input first

We want to analyse if the gpregression can identify the random matrix, if the input is first put through a polynomial kernel of degree 2.

More specifically, the problem looks as follows. We want to approximate the real function f through a gaussian process g, with the following condition:

$$f\left(W\begin{bmatrix}(x-x_0)^2\\(x-x_1)^2\end{bmatrix}\right) \approx g\,(x) \tag{5.1}$$

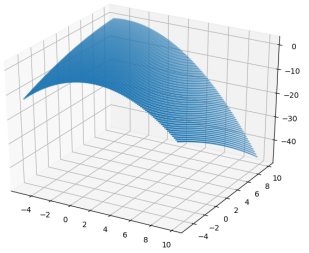
where x_0, x_1 are constants.

5.3.2 Subspace identification

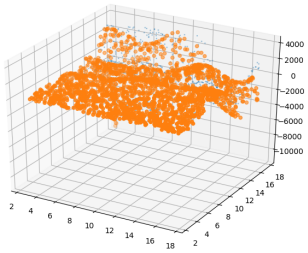
One of the main reasons to use our method is because we allow for subspace identification. We have the following functions:

1. 1D Parabola embedded in a 2D space
2. 2D Camelback embedded in a 5D space
3. 2D Sinusoidal and Exponential function embedded in a 5D space
4. 5D Rosenbrock function embedded in a 10D space

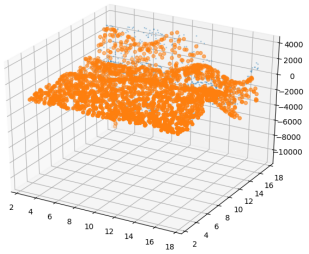
In the following figure, any blue point shows the real function value. Any orange point shows the point that the mean prediction of the Gaussian Process predicted. The function was learned using 100 datapoints. The points shown below were not used for training at any point. The predominantly orange figures are respective surrogate functions of this real function. For each of the methods, the embedding is learned using the respective algorithm.



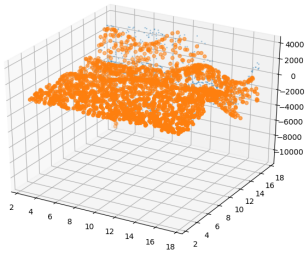
(a) Parabola Original



(b) Parabolanusoidal Boring

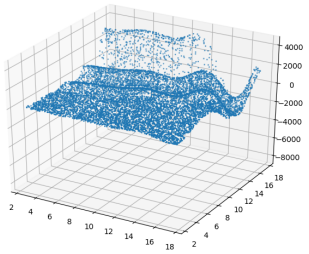


(c) Parabola Tripathy

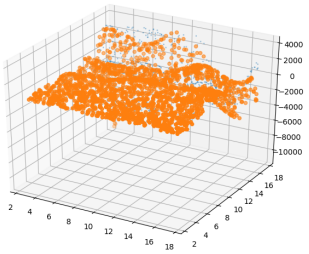


(d) Parabola Rembo

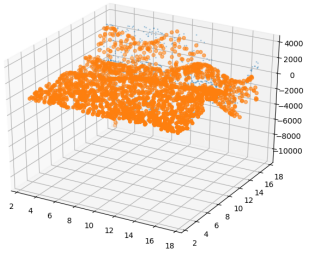
Fig. 5.1 Top-Left: The 1D Parabola which is embedded in a 2D space.



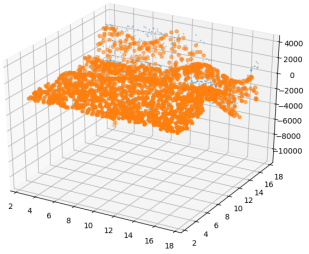
(a) Sinusoidal Original



(b) Sinusoidal Boring

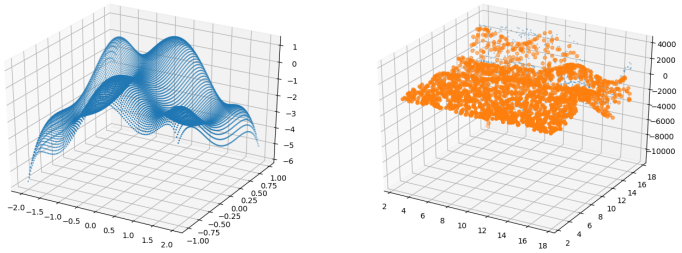


(c) Sinusoidal Tripathy

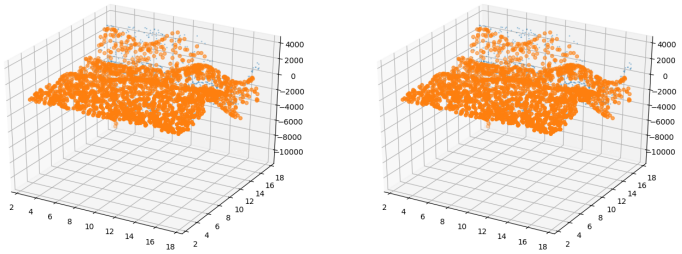


(d) Sinusoidal Rembo

Fig. 5.2 Top-Left: The 2D Sinusoidal-Exponential Function which is embedded in a 5D space.

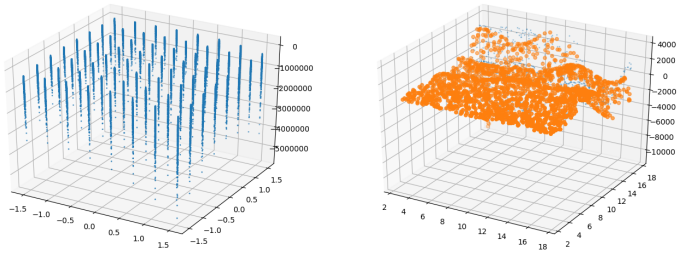


(a) Camelback Original (b) Camelback Boring

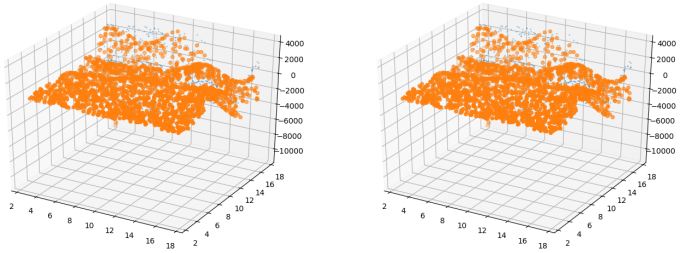


(c) Camelback Tripathy (d) Camelback Rembo

Fig. 5.3 Top-Left: The 2D Camelback Function which is embedded in a 5D space.



(a) Rosenbrock Original (b) Rosenbrock Boring



(c) Rosenbrock Tripathy (d) Rosenbrock Rembo

Fig. 5.4 Top-Left: The 5D Rosenbrock-Exponential Function which is embedded in a 10D space. We visualize the function using the first two principal components (projection using PCA)

References

[1] Abdelrahman, H., Berkenkamp, F., Poland, J., and Krause, A. (2016). Bayesian Optimization for Maximum Power Point Tracking in Photovoltaic Power Plants.

[2] Berkenkamp, F., Turchetta, M., Schoellig, A. P., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Neural Information Processing Systems (NIPS)*.

[3] 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.

[4] Djolonga, J., Krause, A., and Cevher, V. (2013). High-Dimensional Gaussian Process Bandits.

[5] Gardner, J. R., Guo, C., Weinberger, K. Q., Garnett, R., and Grosse, R. (2017). Discovering and Exploiting Additive Structure for Bayesian Optimization.

[6] Garnett, R., Osborne, M. A., and Hennig, P. (2013). Active Learning of Linear Embeddings for Gaussian Processes.

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

[8] Jamil, M. and Yang, X.-S. (2018). A Literature Survey of Benchmark Functions For Global Optimization Problems Citation details: Momin Jamil and Xin-She Yang, A literature survey of benchmark functions for global optimization problems.

[9] Kandasamy, K., Neiswanger, W., Schneider, J., Poczos, B., and Xing, E. (2018). Neural Architecture Search with Bayesian Optimisation and Optimal Transport.

[10] Li, C., Gupta, S., Rana, S., Nguyen, V., Venkatesh, S., and Shilton, A. (2018). High Dimensional Bayesian Optimization Using Dropout.

[11] Lizotte, D. (2008). Practical Bayesian Optimization.

[12] Murphy, K. P. (2012). *Machine Learning, A Probabilistic Perspective*. MIT Press.

[13] Rana, S., Li, C., Gupta, S., Nguyen, V., and Venkatesh, S. (2017). High Dimensional Bayesian Optimization with Elastic Gaussian Process.

[14] Rolland, P., Scarlett, J., Bogunovic, I., and Cevher, V. (2018). High-Dimensional Bayesian Optimization via Additive Models with Overlapping Groups.

[15] Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2009). Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design.

[16] Tripathy, R., Bilionis, I., and Gonzalez, M. (2016). Gaussian processes with built-in dimensionality reduction: Applications in high-dimensional uncertainty propagation.

[17] University, S. F. (2017). Virtual library of simulation experiments: Test functions and datasets - optimization test problems.

[18] Wang, Z., Hutter, F., Zoghi, M., Matheson, D., and De Freitas, N. (2013). Bayesian Optimization in High Dimensions via Random Embeddings.

Appendix A

A.1 Benchmarking functions

The following is a list of synthetic functions and real datasets presented in previous works, where the goal is to evaluate bayesian optimization algorithms.

1. [6] Synthetic in-model data matching the proposed model, with $d = 2,3$, and $D = 10,20$.
2. [6] (Synthetic) Braning function, $d = 2$, hidden in a higher dimensional space $D = 10,20$.
3. [6] Temperature data $D = 106$ and $d = 2$.
4. [6] Communities and Crime dataset $d = 2$, and $D = 96$.
5. [6] Relative location of CT slices on axial axis with $d = 2$ and $D = 318$.
6. [4] (Synthetic) Random GP samples from 2-dimensional Matern-Kernel-output, embedded within 100 dimensions
7. [4] Gabor Filters: Determine visual stimuli that maximally excite some neurons which reacts to edges in the image. We have $f(x) = \exp(-(\theta^T x - 1)^2)$. θ is of size 17x17, and the set of admissible signals is d.
8. [18] (Synthethic) $d = 2$ and $D = 1 * 10^9$.
9. [18] $D = 47$ where each dimension is a parameter of a mixed integer linear programming solver.
10. [18] $D = 14$ with d for a random forest body part classifier.
11. [16] (Synthetic) Use $d = 1,10$ and $D = 10$.

12. [16] (Half-synthetic) Stochastic elliptic partial differential equation, where $D = 100$, and an assume value for d of 1 or 2.
13. [16] Granular crystals $X \in \mathbb{R}^{1000 \times 2n_p+1}$, and $y \in \mathbb{R}^{1000}$.
14. [5] (Synthetic) Styblinski–Tang function where D is freely choosable.
15. [5] (Synthetic) Michalewicz function where D is freely choosable.
16. [5] (Simulated) NASA cosmological constant data where $D = 9$.
17. [5] Simple matrix completion with $D = 3$.
18. [13] (Synthetic) Hertmann6d in [0, 1].
19. [13] (Synthetic) Unnormalized Gaussian PDF with a maximum of 1 in $[-1, 1]^d$ for $D = 20$ and $[-0.5, 0.5]^d$ for $D = 50$
20. [13] (Synthetic) Generalized Rosenbrock function $[-5, 10]^d$
21. [13] Training cascade classifiers, with $D = 10$ per group.
22. [13] Optimizing alloys $D = 13$.
23. [10] (Synthetic) Gaussian mixture function
24. [10] (Synthetic) Schwefel's 1.2 function.
25. [17] A list of optimiztation test functions can be found here.
26. [8] A more comprehensive list of general functions can be found here.