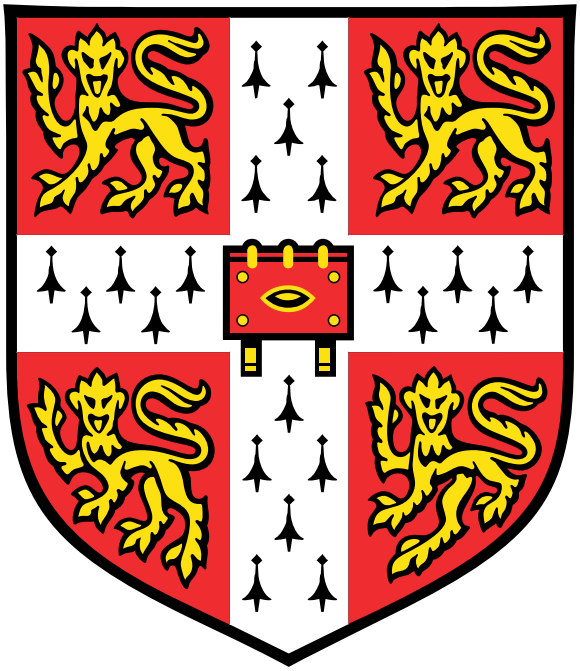
Parameter Optimization using high-dimensional Bayesian Optimization

Bachelor Thesis



David Yenicelik

Department of Computer Science
Swiss Federal Institute of Technology in Zurich

This dissertation is submitted for the degree of

June 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. [7] 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. [6] 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 $x \in \mathbf{R}^D$.

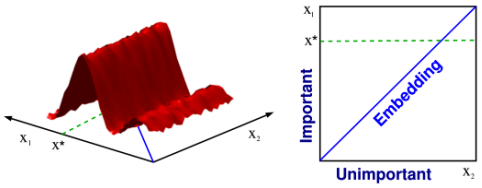


Fig. 1 This function in D=2 dimesions only has d=1 effective dimension. Hence, the 1-dimensional embedding includes the 2-dimensional function's optimizer. It is more efficient to search for the optimum along the 1-dimensional random embedding than in the original 2-dimensional space

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

The main topic of this thesis is how to conduct bayesian optimization in high dimensional domains. Although high dimensional domains can be defined to be between 100 and 1000 dimensions, we will primarily focus on optimization problem that occur in 5 to 30 dimensions. As such, we focus on practical solutions to common problems that can be solved without owning a supercomputer (although extensions to higher dimensions are just as well possible).

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Background

1.1 Bayesian Optimization in high dimensions

Many of today's problems can be boiled down to some flavour of black box optimization. Such problems include neural architecture search, hyper-parameter search for neural networks, parameter optimization for electron accelerators, or drone parameter optimization using safety constraints (CITE Johannes).

Bayesian optimization methods are a class of sequential black box optimization methods, where we learn a surrogate function surface using a Gaussian prior, and a Gaussian likelihood function. Combining the prior and the likelihood results in the Gaussian Posterior, which we then can be used as a surface over which we try to optimize.

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 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).

However, as machine learning algorithm, 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 loose effectiveness in higher dimensions, this work deals with methods that work in higher dimensions.

1.2 Gaussian Processes

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:

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

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.

1.2.1 Derivation of the Gaussian Process Formula

The prior for the Gaussian Process it the following (assuming a 0-mean-prior).

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

Because u is a random variable, it's probably distribution is given by a normal distribution (as determined by the Gaussian Process):

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

Now we go over to observing some data $X = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where y has some noise term ε such that $y = u(x) + \varepsilon$. We assume that ε is normally distributed around 0 with σ standard deviation. This means that the likelihood takes on the following form:

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

Now we want to derive the posterior of the Gaussian Process, given the likelihood and the prior. We use Bayes rule to derive this

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

At this point, we want to create a surrogate embeddings, that predicts y_* for any possible given x_*. We refer to this as the surrogate response surface.

We use the following formula to derive the posterior distribution for a given predictor y_*.

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

To numerically calculate this, one uses the Matrix Inversion Lemma (CITE Murphy Chapter 4.3 110-111)

1.3 Acquisition Functions

Given the above formula for the posterior mean μ and the poster variance σ^2, Bayesian Optimization uses an acquisition function to optimize over. We quickly present some of the most common acquisition functions.

1.3.1 Upper Confident Bound (UCB)

(CITE Krause Srinivas) about how acquisition function is guaranteed to find best config after awhile.

The upper confidence bound allows the user to control exploitation and exploration through a parameter $\beta > 0$, which can be chosen as CITE PAPER, to offer optimality guarantees.

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

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 always selects the point which has the highest potential to maximise the function. The downside to this policy is that this leads to exploitation, which 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)

"CITE A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning"

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

$$\tag{1.11}$$

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

Given one of the above acquisition functions, we then use an optimizer such as $L-BFGS$, 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 datasamples to create a likelihood to further refine the posterior of the function assumption.

1.4 Resources

(CITE https://github.com/SheffieldML/GPy) We will use "GPy: A Gaussian process framework in python" and "CITE https://github.com/SheffieldML/GPyOpt" by the Sheffield Machine Learning Group. In addition to that, the febo framework developed by Johannes Kirschner from the Learning and Adaptive Systems group at ETH Zurich. We use pytest to write tests (CITE).

Chapter 2

Related Work

This section will cover approaches taken so far to solve the problem. We will present algorithms that solve the approach. We will shortly discuss the effectiveness of this algorithm based on the dataset that the authors work on.

Some convention that we use is the following:

1. f is the function that we want to approximate
2. g and any subscripted or superscripted derivative of g is a component that we use to approximate f.
3. Anything that has a "hat" on (caret symbol $\hat{}$), refers to the empirical estimate.

We will focus on the three points:

2.1 Projection matrix based algorithms

We start with enumerating some existing algorithms that are based on projecting the optimization domain. Generally, for this family of algorithms, there is always a term $f(x) \sim g(Ax)$ present, where the properties and generation of A are algorithmic specific.

2.1.1 Active learning of linear subspace

[3] Assume f depends only on $x := uR^T$ with $R \in \mathbf{R}^{d \times D}$ where $d << D$. Learn an algorithm that learns $g(u) = f(x)$ and R.

The proposed algorithm takes the following steps to learn g and R:

Algorithm 1 Simultaneous active learning of functions and their linear embeddings (pseudocode) :: Active learning of linear subspace CITE GARNETT 2013

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 $q(R), q(f)$

1. Create a probability distribution over possible embeddings to learn R (Laplace approximation).

2. We use the calculated embeddings to create a posterior probability distribution over f.

3. Perform active selection over all possible points.

The choice of next point is done using Bayesian Active Learning by disagreement, where the utility function is the expected reduction in entropy (equal to the mutual information), as opposed to uncertainty sampling, which simply minimizes the entropy.

The metrics used in this paper are negative log-likelihoods for the test points, and the mean SKLD (nat) between approximate and true posteriors. The proposed method always outperforms the naive MAP method. 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

[1] Assume 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 $f(x) = g(Ax)$. Assume $g \in \mathscr{C}^2$. Assume that $B = \mathbf{B}^D(1+\varepsilon)$. We want to maximize $f : B \implies [0,1]$.

The SI-BO algorithm has a two-step approach: 1.) subspace identification. 2.) Bayesian Optimization on the learned subspace.

Algorithm 2 The SI-BO algorithm CITE DJOLONGA2013

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

2.1.3 Random embeddings (REMBO)

[8] 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

[7] Assume $f(x) \approx g(\mathbf{W}^T y)$ 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.

A more detailed description:
The quantities of interest are:

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

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

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

We further assume that the response surface has the form of

$$f(x) \approx g(W^T x)$$

.

Because all matrices yield identical approximations, one can focus on orthogonal approximations.

For the family of orthogonal matrices of dimension $d \times D$, we write $\mathbf{W} \in V_d(\mathbb{R}^D)$. This quantity is also known as the Stiefel manifold.

The paper focuses on identifying the low dimensional map $g(\dot{})$, the effective dimensionality d and the orthogonal matrix W.

We determine the hyperparameters by optimizing over the following loss function, where Q are the input samples, t are the corresponding output samples, θ are the ... , and t are the .

$$\mathscr{L}(\theta, s_n; Q, t) = \log p(t|Q, \theta, s_n)$$

where we find the individual variables by maximizing this log-likelihood.

The authors introduce a new kernel function

$$k_{AS} : \mathbb{R}^D \times \mathbb{R}^D \times V_d(\mathbb{R}^D) \times \phi - > \mathbb{R} \tag{2.4}$$

where the kernel has the form

$$k_{AS}(x, x'; W, \phi) = k_d(W^T x, W^T x'; \phi) \tag{2.5}$$

Within the paper, we use the following kernel function

$$1 \tag{2.6}$$

Optimize $W \in V_d(\mathbb{R}^D)$ and keep ϕ and s_n^2 fixed

For this subsection, redefine the loss function as (ϕ are the hyperparameters of the covariance function):

(We instantiate θ with W)

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

where $\phi, s_n; X, y$ are fixed and m is the prior mean function.

The derivative of this function F with respect to the weights-matrix is:

$$\nabla_W F(W) := \nabla_W \mathscr{L}(W, s_n; X, y) \tag{2.11}$$
$$= \frac{1}{2}\mathrm{tr}\left[\{(K + s_n^2 I_N)^{-1}(y-m)\left((K + s_n^2 I_N)^{-1}(y-m)\right)^T - (K + s_n^2 I_N)^{-1}\}\nabla_W (K + s_n^2 I_N)\right] \tag{2.12}$$

both these functions depend on the kernel K, and it's derivative $\nabla_W K$.

We use the 32-Matern kernel function (for two vectors a and b) with derivative:

$$K(a, b, \theta) = s^2 \left(1 + \sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) exp\left(-\sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) \tag{2.13}$$

$$\nabla_W K(a, b, \theta) = s^2 \left(1 + \sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) exp\left(-\sqrt{3}\sum_{i=1}^{l} \frac{(a_i - b_i)^2}{l_i}\right) \tag{2.14}$$

where $s, l_1, \ldots, l_l$ are hyper-parameters included within θ

We optimize over this non-convex optimization problem by using a gradient-descent technique that optimizes over the Stiefel manifold (class of orthogonal matrices).

Identification of active subspace dimension

2.2 Algorithms that exploit additive substructures

We turn our attention to algorithms that assume that the function 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

[2] 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

[5] 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

[4] 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

We will now draw our attention to understanding what the current best models lack. We will enumerate the methods 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:** Not purely robust due to the chance that no suitable subspace will be found. Furthermore, empirically, the choice of the optimization domain heavily affects the duration and effectiveness of the optimization. We have found that the proposed solution to choose $\left[-\sqrt{d}, \sqrt{d}\right]^d$ does not offer a well enough domain solution, even for simple problems (such as 2d embedded functions in 5d spaces)
- **Identification of subspace:** In specific domain settings, such as optimization with safety constraints, or general reproducibility in higher domains, requires the subspace to be identified. REMBO is an implicit optimizer, in that it does not find any subspace. Identifying the active subspace may be a requirement for certain applications.
- **Probability of failure:** Although the authors propose that restarting REMBO multiple times would allow for a good optimization domain to be found, they do not propose an explicit way of how to achieve this.

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:** We don't always have access to the gradients for datasets. Especially in high dimensions, this would require a very, very high number of datapoints per dimension. It would also require these points to be evenly enough distributed, such that the gradients can be effective estimated at many points.
- **Robustness to noise:** According to (CITE TRIPATHY), this method is also quite sensitive to noisy observations. This makes this method not very robust.

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

Tripathy solution argues that it is more robust to real-world noise. They also do not rely on gradient information from the data used for optimization. Tripathy allows for a noise-robust way to identify the active subspace.

- **Duration of optimization:** However, in practice tripathy's solution takes a long time, especially if the dimensions, or the number of datapoints are high (due to the high number of matrix multiplications). Especially in easier problems, it is often desirable not to wait a few hours (but rather a few minutes, or do a precomputation) to find a next best candidate for a point.
- **Efficienty:** Tripathy's solution practically 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 adding heuristics for example), would be desirable.
- **Insensitive to small pertubations:** Tripathy's model - although finding an active subspace, completely neglects the other dimensions which could allow for small perturbations to allow for an increase the global optimum value. There are simple methods that allow for optimization in these domains, and it may be helpful to incorporate such solutions into an existing algorithm.

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 (somehow). Still need to think about this.

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

- choose the matrix that we will take amongst all initialized tries (should be in the paper)
- Test if the expectation
$$E[f(Ax) - f_{hat}(A_{hat}x)]$$
decreases / approaches zero (for methods that identify a projection matrix).
- check if the test log-likelihood decreases
- Check if the l2-loss to the real surface decreases.

3.2.1 Synthetic Datasets

5 dimensional function with 2 dimensional linear embedding It is easy to test snythetic datasets, as we can evaluate these functions at any point, and immediately get a regret value. We will have one test function for different scenarios.

2D to 1D : For this function, we use 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 simpler possibilities, i.e. that the complexity does not sacrifice finding simple models.

5D to 2D : For this function, we use a simple 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 : For this function, we use a simple Camelback function which is enclosed in a sinusoidal function (with decreasing amplitude), which is embedded in a 5D space. This is the most important dataset, as it measures if small perturbations are covered by the more complicated model (our proposed model).

10D to 5D : For this function, we use the 5D Rosenbrock functoin which is embedded in a 10D space. To check how the algorithms cope with higher dimensions. This checks if our model is able to expand to higher dimensions.

3.2.2 Real Datasets

It is more difficult to test algorithms on real datasets, as we cannot quite test a metric such as regret, but rather have to use log likelihood or something similar to test how well our algorithm adapts on a test set, given a training 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. We will first present that algorithm, and then points 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**EMBO 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 we wish to approximate, and any subsequent function annotated by g refers to a component of the right hand side, which approximates the function f.

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

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 generate $A^{\perp}$ using Algorithm 2.

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

We will no proceed with describing the algorithm in more detail.

4.1.1 Algorithm Description

General Idea of the algorithm

We propose a novel method which is based on additive GPs, where we use different kinds of kernels. We want to calculate g_i and f within that 4.1, such that the maximum likelihood of the data we have accumulated so far. 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 REMBO to acquire new points (as this is an efficient form of random sampling). We take $\sqrt{d}$ as the dimensionality of the space Y in which we search (heuristic!!).

1. Calculate the active subspace using the algorithm from
2. Calculate an appropriate pair of vectors $v_1, \ldots, v_{n-q}$, where each vector is orthonormal to every other vector in A.
3. Maximize the GP for each individual expression of the space orthogonal to A (as given by Vx) individually.

This fights the curse of dimensionality, as we can freely choose $q \geq d_e$ to set the complexity of the second term. This, thus allows for 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**
  $A, d \leftarrow$ Calculate active subspace projection using Algorithm 2 from the paper by Tripathy.
  $A^{\perp} \leftarrow$ Generate passive subspace projection using Algorithm 3.
  TODO: CHECK DIMENSIONS IN THE CODE!!! (If concat is in the correct dimension!)
  $Q \leftarrow$ colwiseConcat($[A, A^{\perp}]$)
  $gp \leftarrow GP(\text{dot}(Q^T, X), Y)$
  kernel $\leftarrow$ activeKernel + $\sum_i^q$ passiveKernel$_i$ {For this one, maybe be more explicit with how to set the kernels (as each kernel addresses a different number of dimensions)}
  $x_* \leftarrow \text{argmax}_x$ acquisitionFunction$(dot(Q^{\perp}, x))$ using Equations (REFERENCE HERE)

  Add x_* to X and $f(x_*)$ to Y.
end while
return $q(R), q(f)$

We find the active projection matrix using the following algorithm, which is identical to the procedure described in "CITE TRIPATHY". We then generate a matrix $A^{\perp}$ by using the following procedure.

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 we have found the maximal embedding 4.2. Then we have found the active subspace, which is characterizable through it's first few column vector $a_1, a_2, ..., a_{d_e}$. However, as said before, 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 can be characterized by the set of vectors, that are all orthogonal to all other column vectors in A, i.e. the space orthogonal to the projection of A.

As such, we define the entire span of the given vectorspace as:

$$S = \begin{bmatrix} A & V \end{bmatrix} \tag{4.3}$$

where V describes the matrix that is orthogonal to the columnspace of A. For this, V consists of any basis vectors that are orthogonal to all other vectors in A.

We can generate these vectors by taking a random vector, and applying gram schmidt. We repeat the above procedure if the norm of the resulting vector is smaller than a given threshold. (One could also maybe use fourier, then add another fourier axis, and then go back again to have a nice basis?).

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 i < 50 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 we decompose the function into multiple additive components, we need to adapt the mean and variance computation accordingly, as is described in (CITE THE EPFL PAPER).

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

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

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

4.2 Additive Stiefel projections - Our proposed improvement to existing methods

We model the function f, which we approximate through functions g_i as follows:

We propose the following training method:

1. We first approximate the function

$$f(x) \sim g_1(A_1, x)$$

. We decide the dimensionality of the matrix A_1 by deciding a 'cutoff dimension' as proposed by Tripathy et al. in their Algorithm 4. g_1 is a gaussian process, and A_1 is a learned matrix.

2. We then expand this approximated term to

$$f(x) \sim g_1(A_1, x) + g_2(A_2, x)$$

. In this step, g_1 and A_1 are kept fix. We now learn the function g_2 using a gaussian process, and A_2 using a matrix optimization process. We must make sure that A_1 and A_2 do not map onto the same subspace!

3. We repeat the above procedure until we have the following expression:

$$f(x) \sim g_1(A_1 x_1) + \sum_i g_i(A_i x_i)$$

, where we refer to g_1 as the leading term, and $g_{i=2,...i_{max}}$ as the follow-up terms.

The question now is: Does this structure implicitly find additive substructures between variables, and how is this better than actively finding substructures?

4.2.1 What is it better than actively finding substructures

. Actively finding substructures does find relations between different variables. But it does not account for the weight each variable has within the subsubstruce, which then needs to be accounted by.

We will refer to a smaller function that does not influence the real function f by $O(g_n)$.

We are now interested if it covers some edge cases.

We separate the algorithm into two parts:

1.) Is the model capable of learning this structure 2.) How could training result in this structure, and are the chances big enough?

–> 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:

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

where x_0, x_1 are constants.

5.3.2 Finding a matrix when we apply a polynomial kernel onto the input first (feature selection)

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:

$$f\left(W\begin{bmatrix}(x-x_0)^2\\(x-x_1)^2\end{bmatrix}\right)\approx g\left(x\right) \tag{5.2}$$

where x_0, x_1 are constants.

5.3.3 Subspace identification

References

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

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

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

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

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

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

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

[8] 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. [3] Synthetic in-model data matching the proposed model, with $d = 2,3$, and $D = 10,20$.
2. [3] (Synthetic) Braning function, $d = 2$, hidden in a higher dimensional space $D = 10,20$.
3. [3] Temperature data $D = 106$ and $d = 2$.
4. [3] Communities and Crime dataset $d = 2$, and $D = 96$.
5. [3] Relative location of CT slices on axial axis with $d = 2$ and $D = 318$.
6. [1] (Synthetic) Random GP samples from 2-dimensional Matern-Kernel-output, embedded within 100 dimensions
7. [1] 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. [8] (Synthethic) $d = 2$ and $D = 1 * 10^9$.
9. [8] $D = 47$ where each dimension is a parameter of a mixed integer linear programming solver.
10. [8] $D = 14$ with d for a random forest body part classifier.
11. [7] (Synthetic) Use $d = 1,10$ and $D = 10$.

12. [7] (Half-synthetic) Stochastic elliptic partial differential equation, where $D = 100$, and an assume value for d of 1 or 2.
13. [7] Granular crystals $X \in \mathbb{R}^{1000 \times 2n_p+1}$, and $y \in \mathbb{R}^{1000}$.
14. [2] (Synthetic) Styblinski–Tang function where D is freely choosable.
15. [2] (Synthetic) Michalewicz function where D is freely choosable.
16. [2] (Simulated) NASA cosmological constant data where $D = 9$.
17. [2] Simple matrix completion with $D = 3$.
18. [5] (Synthetic) Hertmann6d in [0, 1].
19. [5] (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. [5] (Synthetic) Generalized Rosenbrock function $[-5, 10]^d$
21. [5] Training cascade classifiers, with $D = 10$ per group.
22. [5] Optimizing alloys $D = 13$.
23. [4] (Synthetic) Gaussian mixture function
24. [4] (Synthetic) Schwefel's 1.2 function.
25. [**?**] A list of optimiztation test functions can be found here.
26. [**?**] A more comprehensive list of general functions can be found here.