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 $\mathscr{Y} = \{y^{(1)}, \dots, y^{(N)}\}$, each evaluated at a point $\mathscr{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$. 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 $x \in \mathbf{R}^D$.

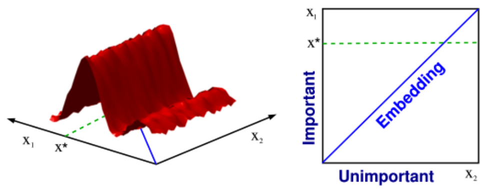

Fig. 1 This function in D=2 dimesions only has d=1 effective dimension. Hence, the 1-dimensional embedding includes the 2-dimensional function's optimizer. It is more efficient to search for the optimum along the 1-dimensional random embedding than in the original 2-dimensional space

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

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Background

1.1

Lorem Ipsum is simply dummy text of the printing and typesetting industry (see

1.2 Solvers

BFGS: https://stats.stackexchange.com/questions/284712/how-does-the-l-bfgs-work

Chapter 2

Existing Approaches

This section will cover approaches taken so far to solve the problem. We will present algorithms that solve the approach, and the dataset that the algorithm was evaluated on, as well as their effectiveness.

2.1 Using a projection matrix

2.1.1 Active learning of linear subspace

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

Steps include: 1.) Create a probability distribution over possible embeddings to learn R (Laplace approximation). 2.) Use this to create a probability distribution over f. 3.) Perform active selection over all possible points.

3.) is done using 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.

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.

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 .

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

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)

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

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

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 Exploiting additive structures within the function

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 usin ghte 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.

2.2.2 Ideas

We could have a function sth like: $f(x) = g(Ax_1) + g(Bx_2) + ...$ as this does not assume that the dimensions are directly correlated. We could also use PCA or anything similar to choose the axis (i.e. $A, B, C..$)

2.3 Other 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.

2.3.3 Ideas

I think optimizing the hyper-parameters of Gaussian processes, and also the kernel is a valueable insight not much explored.

2.4 Datasets for benchmarking

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.

Enumeration

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed vitae laoreet lectus. Donec lacus quam, malesuada ut erat vel, consectetur eleifend tellus. Aliquam non feugiat lacus. Interdum et malesuada fames ac ante ipsum primis in faucibus. Quisque a dolor sit amet dui malesuada malesuada id ac metus. Phasellus posuere egestas mauris, sed porta arcu vulputate ut.

Chapter 3

Experiments

3.1 Biggest competitors

Some main competitors are:

3.1.1 REMBO

3.1.2 Linear embeddings for Gaussian processes

And now I begin my third chapter here . . .

And now to cite some more people **? ?**]

3.1.3 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.

2D to 1D : For this function, we use a simple Parabola which is embedded in a 2D space.

5D to 1D : For this function, we use a simple Parabola which is embedded in a 5D space.

5D to 2D : For this function, we use a simple Camelback function which is embedded in a 5D space.

10D to 5D : For this function, we use the 5D Rosenbrock functoin which is embedded in a 10D space.

3.1.4 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

Implementing methods

4.1 Difficulties when implementing models

When implementing models, the following difficulties may arise:

- Overflows and underflow are often occuring as a reusult of high dimesionality
- Making sure the numerical examples are correct
- Making sure the differentiations are correct (use finite differences for matrices in this example)
- Write theoretical tests of the entire results
- test individual parts
- Test if the expectation

  $$E[f(Ax) - f_hat(A_hatx)]$$

  approaches zero
- Visualize the individual functions
- write a very simple test case of a parabola, and check where the points project to
- Performance of the implementation might not be optimal, as the design is only clear after implementation

4.2 Family of matrices

It turns out that there is a family of matrices that all project to the same space. As such, we need better measures to

- choose the matrix that we will take amongst all initialized tries (should be in the paper)
-

4.3 Methods from

And now I begin my third chapter here ...

And now to cite some more people **? ?**]

4.3.1 First subsection in the first section

... and some more

Chapter 5

Extensions

5.1 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.

5.2 Additive Stiefel projections - Our proposed improvement to existing methods

We propose a novel method which is based on additive GPs, where we use different kinds of kernels.

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

$$f(x) \sim g_1(A_1x_1)+\sum_i g_i(A_ix_i) \tag{5.2}$$

where x_1 is the subvector of the input x that includes the active subspace. And each x_i is a subvector of the input x which is part of the nullspace.

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 $g_{i=2,\ldots i_{max}}$ 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?

5.2.1 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?

5.3 Random addition of REMBO

Another model we potentially propose is again based on additive GP's, where we use again a mixture of different kernels.

We model the function by the following term:

$$f(x) \sim \sum_i g_i^1(A_i^1 x) + \sum_i g_i^2(A_i^2 x) + \sum_i g_i^3(A_i^3 x) + \ldots + \sum_i g_i^{d_{max}}(A_i^{d_{max}} x) \tag{5.3}$$

The idea is that this model can capture the majority of all the possible substructures of a matrix up to a certain dimension d_{max}. We can randomly take and leave behind certain terms from within the above expression. We define this number to be ε, where we leave out one term with probability ε. We can then train the above substructure easily, as the random projection matrices may be fixed.

—- Downsides - this is basically "a random version to find additive substructures within the subspace X".

In minization, we can minize both to best fit the predicted mean (on a test set), and the predicted variance over all values.

–> Possible extension to the Projection pursuit regression?

5.4 Additive variance reducing GP

For this technique, we must detect the number of possible cliques. We don't need to know which variables go together, however, rather, how many times we get a maximal clique.

We can use the EPFL junction tree thing to find how many maximal cliques we have. Then we spawn that many terms with random matrices, but first project them all onto a lower subspace. How do we find out how many active dimensions each of these values has? Or do we assume that each such clique already is in a lower dimension.

Besides fitting a set of test data well through the predicted mean function, we also wish to minimize the

5.5 DimRed

$$f(Ax) + \sum_{i \in Z} G_i(A^{\perp}x)_i) \tag{5.4}$$

where we get the nullspace $A^{\perp}$ of A by taking any two random basis vectors, that are orthogonal to the first few columns of A. This then looks as the following:

Assume we have the following orthogonal A. By definition, this matrix is in the stiefel manifold. Furthermore, this matrix projects any vector to the right of the matrix onto it's columnspace. The columnspace is described by the vectors a_1 and a_2.

$$A = \begin{bmatrix} \vdots & \vdots & & \vdots \\ a_1 & a_2 & ... & a_{d_e} \\ \vdots & \vdots & & \vdots \end{bmatrix} \tag{5.5}$$

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 5.5. 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:

$$S = \begin{bmatrix} A & V \end{bmatrix} \tag{5.6}$$

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?).

Now to the heart of the algorithm. We want to calculate g_i and f within that 5.4, 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.

Why this algorithm?

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!

5.6 Why ours

REMBO has a probaility of failing. This probability is not high, but may result in very, very bad results. Our model gives an approximate best solution given the data, which means that it always performs better than REMBO (because of the built-in restarts).

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

How to install LaTeX

Mac OS X

MacTeX - TeX distribution

1. Download the file from
   https://www.tug.org/mactex/
2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.