UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR ROBOTIK
UND KOGNITIVE SYSTEME

HIBO: Hierarchical Acquisition Functions for Bayesian Optimization

HIBO: Hierarchische Akquisitionfunktionen für Bayessche Optimierung

Masterarbeit

verfasst am
Institut für Robotik und Kognitive Systeme

im Rahmen des Studiengangs
Informatik
der Universität zu Lübeck

vorgelegt von
Franz Johannes Michael Werner

ausgegeben und betreut von
Prof. Dr. Elmar Rückert

mit Unterstützung von
M.Sc. Nils Rottmann

Lübeck, den 14. November 2019

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde in dieser oder ähnlicher Form noch keiner Prüfungskommission vorgelegt.

Franz Johannes Michael Werner

Zusammenfassung

Bayessche Optimierung ist eine mächtige Methode zur Optimierung von Black-Box Funktionen, mit unbekannten Ableitungen und hohen Auswertungskosten. Anwendungen können beispielsweise in dem Bereich der Robotik, der Animationsgestaltung oder dem Entwurf von Molekülen gefunden werden. Allerdings skaliert Bayessche Optimierung nicht in höhere Dimensionen, wenn mehr als 20 Parameter optimiert werden müssen. Die vorliegende Arbeit stellt mit HIBO einen neuen hierarchischen Algorithmus in Bereich der hochdimensionalen Bayesschen Optimierung vor. Der Algorithmus nutzt eine automatische Featuregenerierung. Die Feature werden zur Konditionierung der Parameter verwendet, um eine schnellere Optimierung zu ermöglichen. Die Performanz von HIBO wird mit bereits existierenden Erweiterungen für hochdimensionale Bayessche Optimierung auf drei gebräuchlichen Benchmarkfunktionen verglichen. Zusätzlich wird eine Airhockey Simulation verwendet, um die Leistungsfähigkeit in aufgabenbezogenen Szenarien zu untersuchen. Die durchgeführten Experimente zeigen, dass HIBO, unabhängig von der Dimensionalität des Problems, ähnliche Ergebnisse erzielt, wie der grundlegende Algorithmus der Bayesschen Optimierung. Daher ist HIBO nicht in der Lage Bayessche Optimierung in höheren Dimensionen zu ermöglichen.

Abstract

Bayesian Optimization is a powerful method to optimize black-box derivative-free functions, with high evaluation costs. For instance, applications can be found in the context of robotics, animation design or molecular design. However, Bayesian Optimization is not able to scale into higher dimensions, equivalent to optimizing more than 20 parameters. This thesis introduces HIBO, a new hierarchical algorithm in the context of high dimensional Bayesian Optimization. The algorithm uses an automatic feature generation. The features are used to condition the parameters, to enable faster optimization. The performance of HIBO is compared to existing high dimensional extensions of Bayesian Optimization on three common benchmark functions. Additionally, an air hockey simulation is used to examine the capability in a task-oriented setting. The conducted experiments show that HIBO performs similar to the basic Bayesian Optimization algorithm, independent from the dimensionality of the given problem. Hence, the proposed HIBO algorithm does not scale Bayesian Optimization to higher dimensions.

Acknowledgements

I would like to express my gratitude to Prof. Dr. Elmar Rückert and to Nils Rottmann, from the Institute for Robotics and Cognitive Systems at University Lübeck, for the useful comments, remarks and engagement through the process of this master thesis. Furthermore I would like to thank my partner Lena Rieckmann for proofreading this thesis and her support throughout the whole process of this thesis and my studies at University Lübeck. Finally I would like to thank my parents, for making possible and supporting my entire scientific education.

Contents

List of abbreviations

ANN	Artificial Neural Network
BO	Bayesian Optimization
DBO	Bayesian Optimization with Dropouts
EI	Expected Improvement
GP	Gaussian Process
HIBO	Hierarchical Acquisition Functions for Bayesian Optimization
ICM	Intrinsic Coregionalization Model
MBO	Manifold Bayesian Optimization
mGP	Manifold Gaussian Process
MTGP	Multi Task Gaussian Process
PCA	Principal Component Analysis
PI	Probability of Improvement
REMBO	Bayesian Optimization with Random Embedding
UCB	Upper Confidence Bound

1

Introduction

In recent years, robots were able to increasingly support humans, by taking care of various tasks. For example humanoid robots are responsible for transporting plates on construction sides [23], or service robots gain the capability to help cleaning up a kitchen [47]. An important factor is the ability to manipulate objects. Since there is a large variety in size, form and firmness, gripping objects is challenging. An example for a gripper, suitable for service robot tasks, is the Schunk SVH five-finger hand [38], which can be seen in Fig. 1.1. The hand consists of 9 servo motors and 20 joints. The position of the hand is controlled by a PID controller. In order to perform movements and solving tasks, such a robot has to sense its environment and decide which action has to be performed next. Thereby, the decision process is mostly parameterized, since hard coded solutions are extremely complex and are not able to adapt to new situations. An example are the parameters of the PID controller for the Schunk SVH five-finger hand, with which predetermined waypoints can be approached. Hence, finding proper parameters, leading to optimal strategies for solving given tasks, is an important and challenging task in robotics.

One popular approach for finding optimal parameters is reinforcement learning [42], with policy and non-policy variations. Within a non-policy setting, the robot performs an action, based on the current state and receives a reward. By maximizing the reward an optimal action for a given state can be learned. Policy reinforcement learning requires the robot to perform actions, leading to a reward in the future. Therefore, the parameters of the

Figure 1.1: The Schunk SVH five-finger hand. Image from [41].

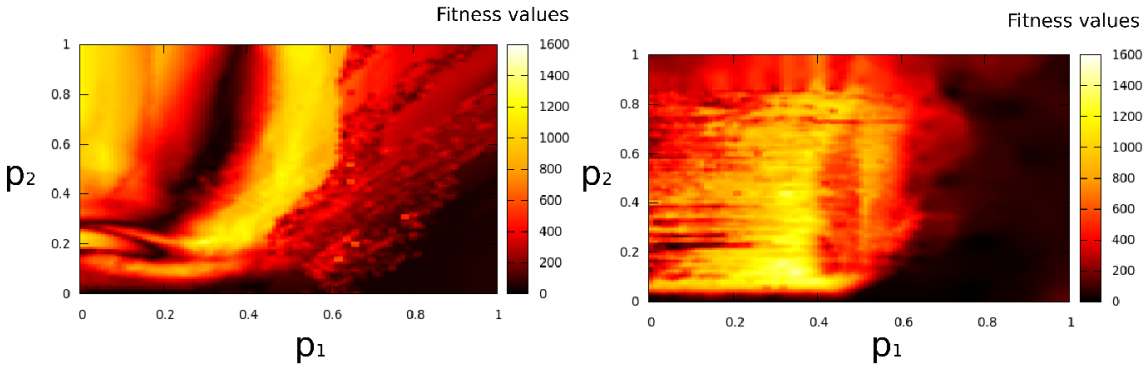


Figure 1.2: (left) Exhaustive fitness landscape in a simulation. (right) Interpolation of the fitness landscape in reality based on about 5500 evaluations. The parameters p_1 and p_2 are used to control the legs of a quadrupedal walking robot. The fitness is the distance (mm) traveled by the robot within 10 seconds. Image from [24].

policy, leading to corresponding actions, have to be learned. For example, catching a ball with a 5-finger hand requires multiple actions, summarized as moving towards the ball and closing the hand. The reward is only provided in the end, when the ball is caught. By repeating the task, the robot will learn proper policy parameters, solving the task according to receiving optimal reward. Another approach are evolutionary algorithms, imitating basic evolutionary processes, like mutation and recombination [36]. A pool, called population, of genetic representations of the parameters or the decision process are applied to the given task. A fitness function measures the performance. By modifying and combining representations with high fitness values, a new generation is created. Repeating the process of measuring the fitness and creating new generations, leads to better solutions over time. Both approaches have in common, that the robot has to repeatedly perform the task, in order to learn the task.

Performing the tasks can be realized in several ways: with a real environment and robot, in a simulation or by combining simulation and the execution on a real robot. However, applying parameters, optimized in a simulation, to a real world situation, often does not lead to corresponding behaviors [5]. This is called reality gap. Due to differences in sensing and actuation, parameters can work well in a simulation, but fail in the real world. Additionally, the physics of a simulation may allow solutions, that can not occur in the real world, especially in a dynamic environment [28]. An example for the reality gap can be seen in Fig. 1.2, where the performances of a controller in a simulation and in reality are compared. The simulation computes high fitness values for some solutions, failing in reality. In addition, there are solutions that achieve good results in reality, but are marked as bad by the simulation. Hence, to ensure valid solutions, the optimization of the parameters has to be performed in the real world. This requires more human resources and time, compared to simulated experiments. The experimental setup has to be prepared for each parameter evaluation and the experiment itself has to be supervised. Additionally, replacements for the robot itself are required, as inappropriate parameters can cause damage. All in all, testing a set of parameters is an expensive process. Therefore, it is desirable, to find proper parameters within a few evaluations.

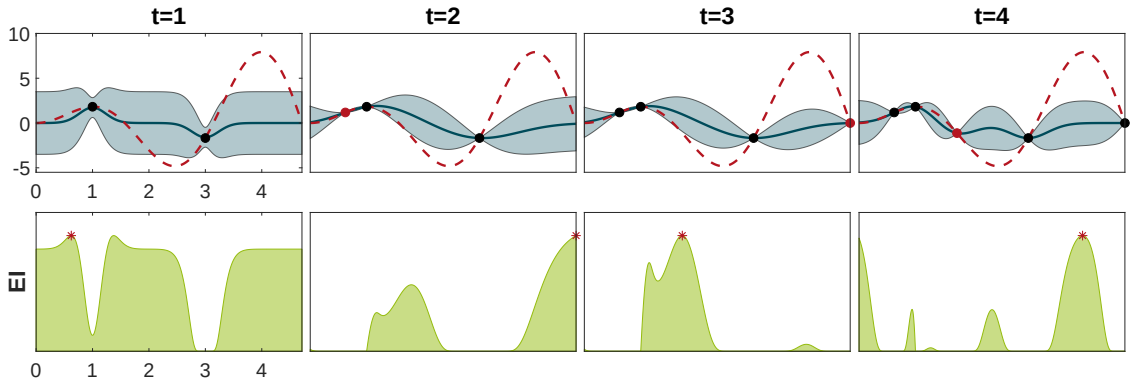


Figure 1.3: BO applied on a one-dimensional function $f(x) = \sin(x) \cdot x$ (dashed red line). Black dots marks previous evaluations of the parameter x and the red dot the position of the last evaluation. The blue line and area shows the belief over the target function created by the model. The green area shows the acquisition function, determining the next value for the parameter x, marked by the red asterisk.

For finding optimal solutions within few iterations, Bayesian Optimization (BO) can be applied. BO is a machine-learning-based method, for black-box derivative-free global optimization [12]. BO can be applied on value based problems but also to policy optimization by optimizing the parameters of the policy. Based on previous evaluations, a model of the objective function is build. This model is used by an acquisition function, which suggests a new parameter set for evaluation. By repeatedly updating the model, the prediction for good parameters is improved. In Fig. 1.3 an exemplary progress of BO on a one-dimensional function is shown. BO is not only restricted to robotics, like in [2, 8, 30, 32], but can be applied in various domains, like animation design [4], molecular design [17] or environmental monitoring [31]. Most successful applications of BO have about 20 or less parameters to optimize [12, 43]. However, optimization problems can easily exceed the number of 20 parameters. Learning the parameters of the PID controller on the SVH hand, results in 27 parameters in total, since controlling a single motor requires three parameters. This can be extended by additional parameters. For example when gripping objects, different materials requires different forces applied to the object. Additionally, the posture of the hand plays an important role. Another example is the design of molecules regarding certain properties. In [17] the encoding of the molecules consists of up to 120 characters.

This leads to research, where BO is extended, in order to perform well in higher dimensions. If an appropriate amount of data is available, an auto encoder can be trained as in [17]. However, since the evaluation of parameters is associated with high costs, access to such data is generally not possible. This leads to approaches which only rely on the few evaluations performed during the optimization process. In [50, 51] the optimization is performed in a lower dimensional embedding. A randomly created matrix is used to project the embedding into the parameter space. The work in [29], inspired by the Dropout algorithm for neural networks, optimizes only a smaller and randomly chosen subset of parameters per iteration. In [34] the hyper parameters for a neural net are jointly learned with the model of the BO.

On the feature space, created by the neural net, a acquisition function is used to compute a new feature. Using a Multi Output Gaussian Process the feature is mapped back into the parameter space.

Within this thesis, a new approach, called Hierarchical Acquisition Functions for Bayesian Optimization (HIBO), is examined. Two acquisition functions are used in a hierarchical manner. In a first step, an acquisition function is applied on a feature space. In a second step, the parameter and the feature space are combined. The feature, computed by the first acquisition function, is then used to condition the search space, while applying a second acquisition function on the combined space. This approach has been successfully tested with a handcrafted feature generation and additional modifications of the BO algorithm. Therefore, this thesis addresses the automatic feature generation for the HIBO algorithm. Additionally, the performance of HIBO alone is investigated.

Thesis Organization

In Chapter 2 basic methods and techniques are covered. This includes Artificial Neural Networks, Bayesian Optimization, Gaussian Processes and extensions of Gaussian Processes. Extensions of Bayesian Optimization, including HIBO, are introduced in Chapter 3. The HIBO algorithm will be compared to BO and three other extensions, called Bayesian Optimization with Random Embedding (REMBO), Bayesian Optimization with Dropouts (DBO) and Manifold Bayesian Optimization (MBO). Chapter 4 describes and evaluates experiments to investigate the performance of the algorithms. At first three common benchmark functions are used to compare the algorithms. Additionally, an air hockey simulation is used, to learn to score a goal from a fixed point. In Chapter 5 the results are discussed and suggestions for future work are given. In the end, the thesis is summarized in Chapter 6.

Mathematical Notation

In the following, the mathematical notation used throughout this thesis is introduced.
Scalar values are written as lower-case letters (e.g. a, s). Vectors are depicted as small bold lower-case letters (e.g. $\boldsymbol{a}, \boldsymbol{\mu}$) and matrices as bold capital letters (e.g. $\boldsymbol{A}, \boldsymbol{\Sigma}$). A single scalar value from a matrix placed in row i and column j is written as $(\boldsymbol{A})_{ij}$. The identity matrix is written as $\boldsymbol{I}$ where the size is chosen appropriate to the usage.

Sets are written as blackboard bold (e.g. $\mathbb{D}$). Please note that labels for number categories, like $\mathbb{R}$, keep there general meaning. Spaces are denoted as calligraphic capital letters (e.g. $\mathcal{X}$).

A function $\mu(x)$ can be shortly written as μ_x. The exponential function is denoted with $\exp(x)$ or e^x with $\exp(x) = e^x$. The variance of a variable x is written as $\text{var}(x)$, the covariance between the variables x and y as $\text{cov}(x, y)$ and the expectation value of a variable x as $\mathbb{E}(x)$.

2

Background Methods

This chapter describes the basic methods and techniques, used in this thesis. It starts with a brief introduction to Artificial Neural Networks in Section 2.1, which are used as customizable nonlinear transformation function. Section 2.2 addresses Bayesian Optimization, an efficient approach for optimizing functions with high evaluation costs. In Section 2.3 Gaussian Process Regression is covered, which is used as non-parametric model for Bayesian Optimization. Sections 2.4 and 2.5 introduce extensions of Gaussian Process Regression, Manifold Gaussian Process and Multi Output Gaussian Process respectively. Manifold Gaussian Processs use a feature space created by an Artificial Neural Network and Multi Output Gaussian Processes enable the modeling of functions with multiple output dimensions.

2.1 Artificial Neural Networks

In recent years, Artificial Neural Networks (ANNs) and variations, like Deep Neural Networks or Convolutional Neural Networks [27, 13] were successfully applied to image classification [25], face recognition [52] and speech to text transformation [18]. ANNs are inspired by the functionality of the human brain. Similar to a biological neural net an ANN consists of artificial neurons, which are arranged within a layered structure. Due to parameterized connections between the artificial neurons, the ANN is able to adapt to training data. In the first part artificial neurons are introduced. In the next step ANNs are briefly covered.

2.1.1 Artificial Neuron

In 1949 McCulloch and Pitts [33] published a mathematical approach to model biological neurons. Thereby the neurons complexity is distinctly reduced and also some components and processes of a biological neuron are not considered. A schematic representation of a biological neuron with its basic components can be seen in Fig. 2.1. The model of McColloch and Pitts captures the following processes within a biological neuron. Input signals are weighted by the efficiency of their synapses. Based on accumulated signals and a trigger threshold an output signal emerges. Physical and chemical processes are not considered by the model. The model consist of an input vector $\mathbf{x}$, weights $\mathbf{w}$, a threshold θ and an

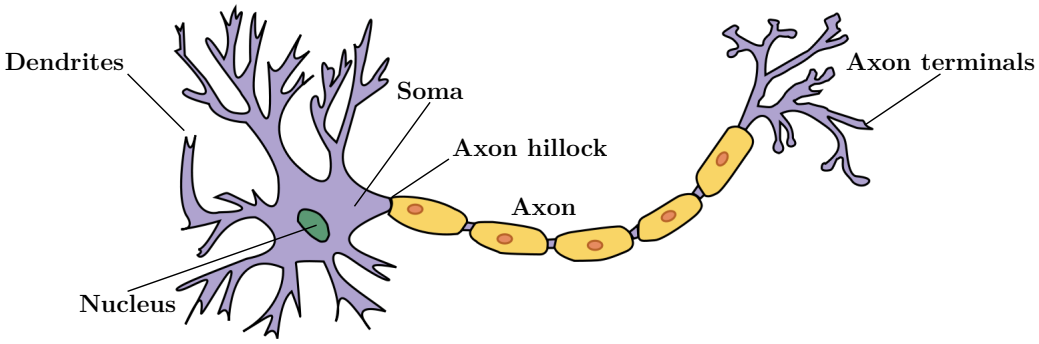


Figure 2.1: Schematic representation of a biological neuron from [10]. A neuron consists of the soma with the nucleus, dendrites, the axon and axon terminals. Within the nucleus the signals received via the dendrites are processed. If a certain voltage level is exceeded, an action potential emerges in the axon hillock. It is then headed via the axon towards the axon terminals. The axon terminals are linked to the dendrites of other neurons by synapses. Due to different synapse efficiencies, the signals arriving at the dendrites have different effects on the behavior of the corresponding neuron.

activation function φ. The output y of a single neuron is computed as

$$y = \varphi\left(\mathbf{w}^T \cdot \mathbf{x} - \theta\right).$$

Common choices for the activation function φ are the Rectified Linear Unit

$$\mathrm{relu}(x) = \max(0, x)$$

and sigmoid functions like

$$\mathrm{sig}(x) = \frac{e^x}{1 + e^x},$$

as shown in Fig. 2.2.

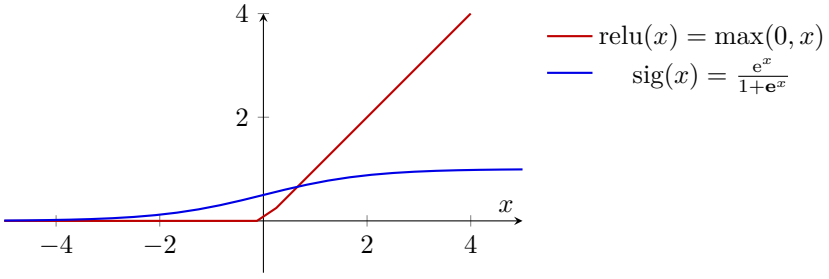


Figure 2.2: Plots of the Rectified Linear Unit and a sigmoid function in the range of $[-5, 5]$.

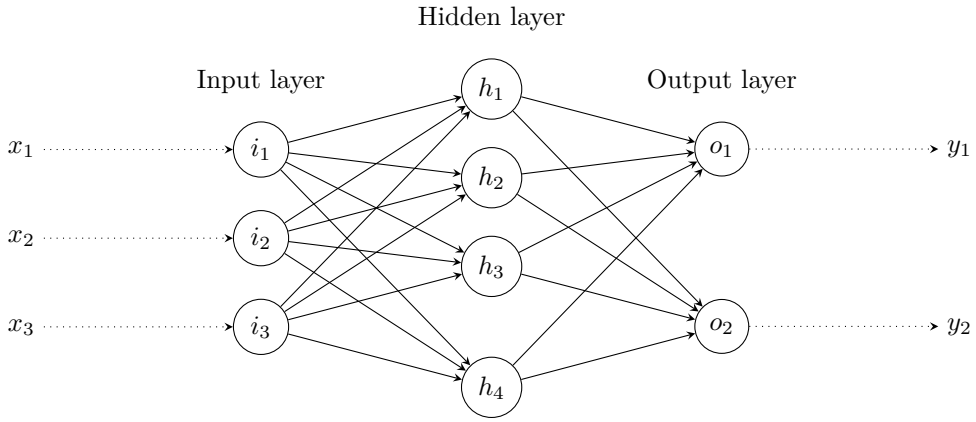


Figure 2.3: Schematic representation of a simple ANN. The network consists of an input layer with three neurons (i_1, i_2 and i_3), a hidden layer with four neurons (h_1, h_2, h_3 and h_4) and an ouput layer with two neurons (o_1 and o_2). Mathematically this ANN is a function $f : \mathbb{R}^3 \to \mathbb{R}^2$, $(x_1, x_2, x_3) \mapsto (y_1, y_2)$.

2.1.2 Artificial Neural Network

An ANN consists of multiple connected artificial neurons. The arrangement of the artificial neurons corresponds to a graph, where the neurons are the nodes and the synapses are the edges. Typically an ANN is structured in layers. Every ANN consists of an input and an output layer. Between both layers an arbitrary number of hidden layers can be placed. Within this thesis feed forward ANNs with fully connected layers are used. In feed forward ANNs with fully connected layers, a neuron is connected to all neurons of the previous layer. Additionally, the output of a neuron can only be passed to a following layer and not to the own or previous layers. In Fig. 2.3, a simple ANN with three input neurons, four hidden neurons and two output neurons is shown. An ANN is parameterized by the weights and thresholds of its artificial neurons.

Very common applications of ANNs are classification and regression. Both applications need a dataset of inputs $\mathbf{x}$ and target outputs or labels $\mathbf{s}$. After propagating the input $\mathbf{x}$ through the ANN the output $\mathbf{y}$ can be compared to the labels $\mathbf{s}$. The gradient of the error function with respect to the parameters can be used to update the parameters. This is called backpropagation [39]. This approach requires a large dataset of known inputs and target outputs. Within this thesis ANNs are applied in the context of Bayesian Optimization where functions with high evaluation costs are optimized. Thus, in a typical case there is no access to a huge amount of data. As a consequence backpropagation can not be used to tune the parameters of an ANN. Instead the parameters are integrated into a Gaussian Process, which will be introduced in Section 2.4.

2.2 Bayesian Optimization

Consider the optimization of an objective function $f : \mathcal{X} \to \mathbb{R}$:

$$\boldsymbol{x}^* = \text{argmax}_{\boldsymbol{x} \in \mathcal{X}} \ f(\boldsymbol{x}),$$

where $f_{\boldsymbol{x}}$ has the following properties [12]. Due to relatively high costs to obtain the value of $f_{\boldsymbol{x}}$, the number of evaluations is limited. This is the case for evaluations performed by human, time consuming tasks or monetary expensive evaluations. $f_{\boldsymbol{x}}$ is a black-box-function. Thus, there is no knowledge about structural properties of $f_{\boldsymbol{x}}$ which could simplify the optimization. Additionally, only $f_{\boldsymbol{x}}$ can be observed. Knowledge about the derivatives of $f_{\boldsymbol{x}}$ is not accessible. Due to this properties, neither gradient based methods nor grid- or randomsearch can be applied. A solution is given by Bayesian Optimization.

Bayesian Optimization (BO) is a machine-learning-based optimization approach, designed for black-box derivative-free global optimization [12]. It consists of two main components: a probabilistic model and an acquisition function. The model is used to build a prior belief over the possible objective function $f_{\boldsymbol{x}}$. The model is sequentially updated with new results of evaluating the true objective function for known parameters. The acquisition function uses the prior belief to determine the next candidate for evaluating the objective function. For each evaluation in the optimization process we store the observed point $\boldsymbol{x} \in \mathcal{X}$ and the obtained objective value $y = f(\boldsymbol{x})$ in a dataset $\mathbb{D} = \{\mathcal{X}, \mathbb{R}\}$. Initially the objective function is evaluated at a small number of randomly chosen points. In following iterations the model is updated based on all observations in $\mathbb{D}$. In a second step the acquisition function uses the model to find a new point $\boldsymbol{x}$ where the objective function $f_{\boldsymbol{x}}$ is evaluated. The observation is then stored in $\mathbb{D}$ as well. See Alg. 2.1 for the pseudo-code of this framework.

Typically the parameter space $\mathcal{X}$ is often a compact subset of $\mathbb{R}^d$. But also more unusual search spaces that involve categorical or conditional inputs, or even combinatorial search spaces with multiple categorical inputs are suitable to BO [43]. BO has been successfully applied to a variety of tasks. For example in robotics BO has been used to learn gates of a dynamic bipedal walker [8] or quadruped robots [30], for trajectory planing in the context of robot soccer [2], or to train a humanoid robot to balance an inverted pendulum [32]. Other applications are for example in pharmaceutical product development [40], environmental monitoring [31], sensor networks[14], rail network optimization [20] or animation design [4]. However, most successful applications require only 20 or less parameters to be optimized [12, 43].

One can choose from different approaches to model the objective function. Next to Gaussian Process (GP) regression, Sparse spectrum Gaussian Processes and Random Forest have been successfully applied in the context of BO [43]. Within this thesis GP regression is used to model the objective function. In the following section common choices for acquisition functions will be discussed.

2.2.1 Acquisition Functions

Acquisition functions are used to compute the next parameters for evaluating the objective function. Therefor an acquisition function $\alpha : \mathcal{X} \to \mathbb{R}$ is maximized. Unlike the unknown

Algorithm 2.1: Pseudocode Bayesian Optimization. The goal is to find an almost optimal parameter $\mathbf{x}^*$ for the blackbox function f with N evaluations in total. At first the dataset $\mathbb{D}$ is initialized with n_0 entries. In a second step new solutions are computed using the acquisition function α and the dataset is updated.

algorithm *Bayesian Optimization*
input $f_{\mathbf{x}}$ // target function
n_0 // number of initial observations
N // total number of evaluations
output $\mathbf{x}^*$ // optimized parameter
// Initialize dataset with n_0 observations
for $n = 1$ *to* n_0 *do*
  $\mathbf{x}_n \leftarrow random(\mathcal{X})$
  $\mathbb{D}_n \leftarrow \{\mathbb{D}_{n-1}, (\mathbf{x}_n, f_{\mathbf{x}}(\mathbf{x}_n))\}$
// Optimize with Bayesian Optimization
for $n = n_0 + 1$ *to* N *do*
  $\mathbf{x}_n \leftarrow \text{argmax}_{\mathbf{x}\in\mathcal{X}}\ \alpha(\mathbf{x}; \mathbb{D}_{n-1})$
  $\mathbb{D}_n \leftarrow \{\mathbb{D}_{n-1}, (\mathbf{x}_n, f_{\mathbf{x}}(\mathbf{x}_n))\}$
$\mathbf{x}^* \leftarrow \text{argmax}_{\mathbf{x}\in\mathcal{X}}\ \alpha(\mathbf{x}; \mathbb{D}_N)$

function $f_{\mathbf{x}}$, the evaluation of α is inexpensive, which enables differed approaches to maximize the acquisition function [4, 12]. The sample at $\text{argmax}_{\mathbf{x}\in\mathcal{X}}\ \alpha(\mathbf{x}; \mathbb{D})$ is suggested for the next evaluation. BO aims to find a global optimum with a small number of evaluations. Thus, the trade-off between exploration and exploitation in the search space $\mathcal{X}$ has to be considered. This section presents common choices of α, enabling different strategies. Common ground of the approaches is that they require a model that predicts a mean value and a corresponding variance for a given $\mathbf{x} \in \mathcal{X}$. $\mu_n(\mathbf{x})$ and $\sigma_n(\mathbf{x})$ are the predicted mean and variance computed with GP regression for a certain parameter $\mathbf{x}$ given the current dataset $\mathbb{D}_n$.

Probability of Improvement

An improvement-based approach is Probability of Improvement (PI) [26]. The idea is to measure the probability that a point $\mathbf{x}$ leads to an improvement upon the incumbent target $f(\mathbf{x}^+)$ where $\mathbf{x}^+ = \text{argmax}_{\mathbf{x}\in\mathcal{X}}\ f(\mathbf{x})$. Using the Gaussian posterior distribution of the modeled function $f(\mathbf{x})$ the probability can be computed as

$$\alpha_{\text{PI}}(\mathbf{x}; \mathbb{D}_n) := \mathbb{P}\left[f(\mathbf{x}) \geq f(\mathbf{x}^+)\right] = \Phi\left(\frac{\mu_n(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma_n(\mathbf{x})}\right),$$

where Φ is the normal cumulative distribution function. α_{PI} can be seen as an exploration strategy, since points with a high probability to be slightly larger than $f(\mathbf{x}^+)$ are favored over points with possible larger gains but with more uncertainty, expressed by σ_n. The first section in Fig. 2.4 shows how PI behaves on a one-dimensional toy problem.

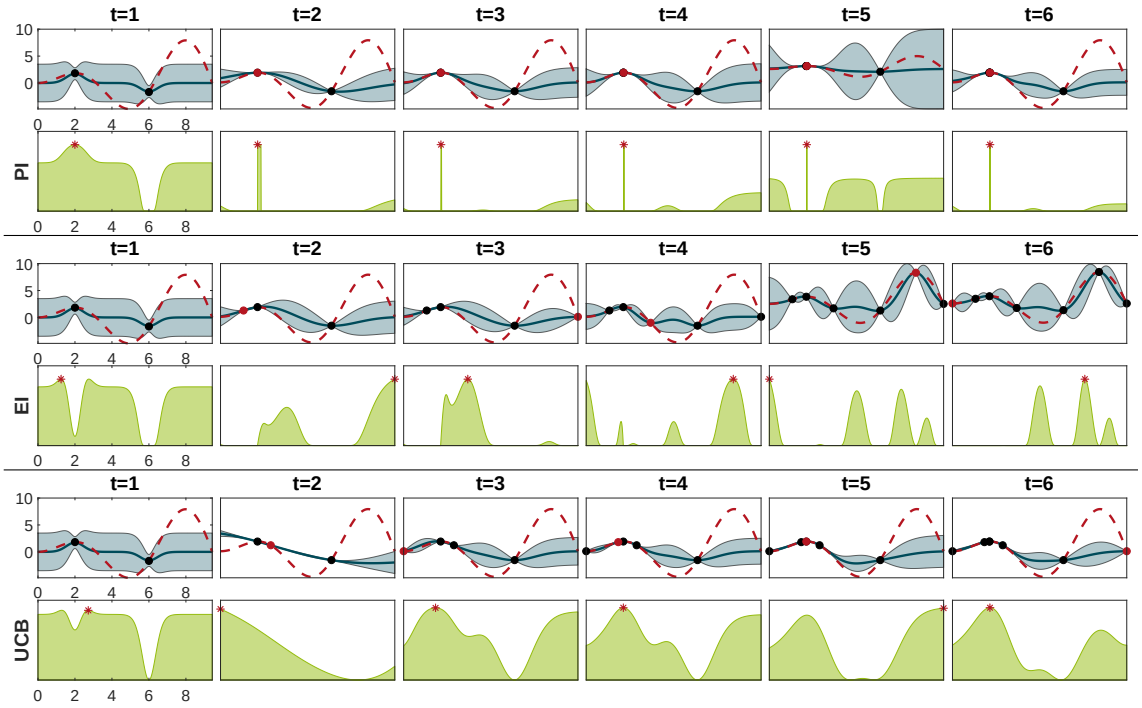


Figure 2.4: Comparison of the acquisition functions PI, EI and UCB on a one-dimensional toy problem. For each acquisition function BO is applied to the target function $f(x) = \sin(x) \cdot x$ with 5 iterations. In all cases the function is initially evaluated at $x_1 = 2$ and $x_2 = 6$. In the upper row of each section the objective function f_x is shown as a red dotted line and the predictive mean as a solid blue line. Black dots are marking previous evaluated points where the red dot marks the last evaluated point. In the lower row the values of the acquisition functions are shown. The maximum of the acquisition function is marked by a red asterisk.

Expected Improvement

Instead of only considering the probability of improvement, Expected Improvement (EI) [35] also considers the magnitude of improvement that a point can possible provide. This is achieved with

$$\alpha_{\text{EI}}(\mathbf{x}; \mathbb{D}_n) := \begin{cases} (\mu_n(\mathbf{x}) - f(\mathbf{x}^+)\Phi(Z) + \sigma_n(\mathbf{x})\Phi(Z) & \text{if } \sigma_n(\mathbf{x}) > 0 \\ 0 & \text{if } \sigma_n(\mathbf{x}) = 0, \end{cases}$$
$$Z = \frac{\mu_n(\mathbf{x}) - f(\mathbf{x}^+)}{\sigma_n(\mathbf{x})}, \tag{2.1}$$

where Φ is the normal cumulative distribution function and Φ the standard normal probability density function. The second section of Fig. 2.4 shows the application of EI on a one-dimensional toy problem.

Confidence bound criteria

For minimizing a function Cox et al. [9] provided an algorithm called „Sequential Design for Optimization“, which selects points for evaluating based on a lower confidence bound

$$\alpha_{\text{LCB}}(\mathbf{x}; \mathbb{D}_n) = \mu_n(\mathbf{x}) - \kappa\sigma_n(\mathbf{x}),$$

with a hyperparameter $\kappa \geq 0$, given a random function model. To adapt this algorithm to maximization of a acqusition function, one can use the Upper Confidence Bound (UCB) criteria

$$\alpha_{\text{UCB}}(\mathbf{x}; \mathbb{D}_n) = \mu_n(\mathbf{x}) + \kappa\sigma_n(\mathbf{x}).$$

With α_{UCB} points where the upper confidence promises high values are preferred. Srinivas et al. [45] suggest setting $\kappa = \sqrt{v\tau_t}$, with $v > 0$ and $\tau_t = 2\log\left(t^{d/2+2} + \pi^2/3\delta\right)$, resulting in

$$\alpha_{\text{GP-UCB}}(\mathbf{x}; \mathbb{D}_n) = \mu_n(\mathbf{x}) + \sqrt{v\tau_t}\sigma_n(\mathbf{x}).$$

The behavior of UCB on a toy sample can be seen in the third section of Fig. 2.4.

2.3 Gaussian Process Regression

GP regression is a non-parametric, statistical method to model functions, which are mapping a multi dimensional space $\mathcal{X} \subseteq \mathbb{R}^D$ to a one-dimensional value. Thereby it learns a transformation $F : \mathcal{X} \mapsto \mathcal{Y}$ which aims to model a function $f : \mathcal{X} \mapsto \mathcal{Y}$, with $\mathcal{Y} \subseteq \mathbb{R}$. A GP is completely specified by its mean function $\mu_0(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ [37], which are defined as

$$\begin{aligned} \mu_0(\mathbf{x}) &= \mathbb{E}\left[f(\mathbf{x})\right] \\ k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[\left(f(\mathbf{x}) - \mu_0(\mathbf{x})\right)\left(f(\mathbf{x}') - \mu_0(\mathbf{x}')\right)\right]. \end{aligned} \tag{2.2}$$

A GP can be used to sample functions f from a prior with mean μ_0 and covariance $\mathbf{K}$ computing [37]

$$\begin{aligned} \mathbf{u} &\sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \\ f &= \mu_0 + \mathbf{L}\mathbf{u}, \end{aligned}$$

where $\mathbf{L}$ is the lower triangular matrix obtained from the Cholesky decomposition of $\mathbf{K} = \mathbf{L}\mathbf{L}^T$. To ensure numerical stability one can add $\sigma\mathbf{I}$ to the covariance $\mathbf{K}$, with a noise variance $\sigma > 0$. In addition, GP regression computes samples drawn from a posterior given training points $X \subset \mathcal{X}$ and target values $\mathbf{y} = f(X)$, with $y_i \in \mathbb{R}$. For GP regression the joint distribution of known function values $\mathbf{y}$ and test outputs $\mathbf{y}_*$ is obtained by

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right), \tag{2.3}$$

where $\mathbf{y} = f(X)$ are the known function values for the training points $X \subset \mathcal{X}$, $\mathbf{y}_*$ are the predictions for the test points $X_* \subset \mathcal{X}$. The elements of $\mathbf{m}$ and $\mathbf{m}_*$ are defined as $m_i = \mu_0(\mathbf{x}_i)$ and $m_{*i} = \mu_0(\mathbf{x}_{*i})$, for $\mathbf{x}_i \in X$ and $\mathbf{x}_{*i} \in X_*$. The covariance matrices $\mathbf{K} = k(X, X)$, $\mathbf{K}_* = k(X, X_*) = k(X_*, X)^T$ and $\mathbf{K}_{**} = k(X_*, X_*)$ are based on a chosen kernel function k. Common choices for k will be discussed later in this section.

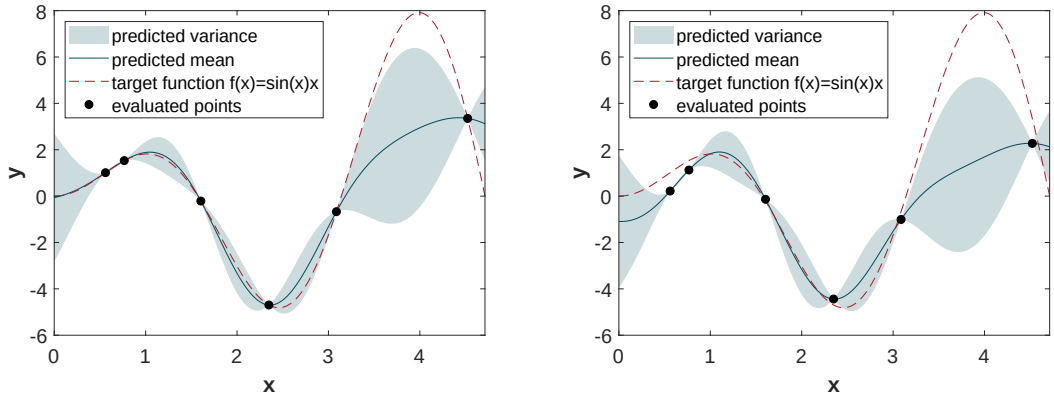


Figure 2.5: An example for GP regression applied on the one-dimensional function $f(x) = \sin(x) \cdot x$ in the range of $[0, 3\pi]$. The target function is marked with the red dashed line. For the left image the target function is noise free. In the right image the evaluation of target function is disturbed with a noise level of $\sigma_n = 0.5$. Thxe x-values for the evaluation (black dots) are drawn randomly but are identical in both examples. The predictive mean is drawn as a blue line and the predictive variance is visualized with the bluish filled area.

By conditioning the prior to only contain functions which agree to the training points, the prediction for a single data point $\mathbf{x}_* \in X_*$, using GP regression, can be computed as [37]

$$\begin{aligned} \mu(\mathbf{x}_*) &= \mu_0(\mathbf{x}_*) + \mathbf{k}_*^T \mathbf{K}^{-1}(\mathbf{y} - \mathbf{m}) \\ \sigma(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*, \end{aligned} \tag{2.4}$$

with $\mathbf{k}_* = k(\mathbf{x}_*, X)$ as the covariance between $\mathbf{x}_*$ and the training input X. The predictive mean $\mu(\mathbf{x}_*)$ represent the prediction at the given point $\mathbf{x}_*$ and the predictive variance $\sigma(\mathbf{x}_*)$ can be interpret as the uncertainty for that prediction. In Eq. (2.3), it is assumed that the observation values $\mathbf{y}$ are exact. However, in general the observation is corrupted by noise, such that $\mathbf{y} = f(X) + \epsilon$, with a Gaussian Noise $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Therefore, the prior of the observed data changes to $\text{cov}(\mathbf{y}) = \mathbf{K} + \sigma_n^2 \mathbf{I}$, with a observation variance σ_n. Now the joint distribution of the observed data and the function values can be written as

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}_* \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ \mathbf{m}_* \end{bmatrix}, \begin{bmatrix} \mathbf{K} + \sigma_n^2 \mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix} \right).$$

And the prediction using GP regression is then computed as

$$\begin{aligned} \mu(\mathbf{x}_*) &= \mu_0(\mathbf{x}_*) + \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} (\mathbf{y} - \mathbf{m}) \\ \sigma(\mathbf{x}_*) &= k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^T \left(\mathbf{K} + \sigma_n^2 \mathbf{I}\right)^{-1} \mathbf{k}_*. \end{aligned} \tag{2.5}$$

An example for the results of GP regression on a one-dimensional function, with noise free and noise disturbed evaluation is shown in Fig. 2.5. Rasmussen et al. [37] provide a practical implementation of the GP regression with zero mean, which can be seen in Alg. 2.2. Instead of the matrix inversion, Cholesky decomposition is used, since it is faster and numerically more stable. The prediction is based on the mean function $\mu_0(\mathbf{x})$ and

Algorithm 2.2: Pseudocode Gaussian Process Regression with zero mean

algorithm *Gaussian Process Regression*
input X // training inputs
$\mathbf{y}$ // targets
k // covariance function
σ_n^2 // noise level
X_* // test inputs
output $\boldsymbol{\mu}$ // predictive mean at X_*
$\boldsymbol{\sigma}$ // predictive variance at X_*
$L \leftarrow \text{cholesky}(\mathbf{K} + \sigma_n^2 \mathbf{I})$
$\boldsymbol{\alpha} \leftarrow L^T \backslash (L \backslash \mathbf{y})$
for $\mathbf{x}_*$ *in* X_* *do*
$\mathbf{k}_* \leftarrow k(\mathbf{x}_*, X)$
// predictive mean
append $\mathbf{k}_*^T \boldsymbol{\alpha}$ *to* $\boldsymbol{\mu}$
// predictive variance
$\mathbf{v} \leftarrow L \backslash \mathbf{k}_*$
append $k(\mathbf{x}_*, \mathbf{k}_*) - \mathbf{v}^t \mathbf{v}$ *to* $\boldsymbol{\sigma}$
end

the covariance function $k(\mathbf{x}, \mathbf{x}')$, which influence the behavior of the predicted function. The mean function $\mu_0(\mathbf{x})$ provides a possible offset [43]. The mean function also offers the possibility to incorporate expert knowledge of the target function, if available. However, usually a constant mean $\mu_0(\mathbf{x}) = m_0$ is used, due to missing knowledge about the mean of the target function. Additionally, the impact of the mean is relatively small, compared to the impact of the kernel function. The covariance function, or kernel function, dictates the structure, like the smoothness and amplitude [43]. For example, if the target function $f_{\mathbf{x}}$ is expected to be a periodic function, one can use a periodic kernel function to mimic this behavior.

2.3.1 Kernel functions

For GP regression stationary kernels are commonly used, which are shift invariant. An example for a stationary kernel is the squared exponential kernel function

$$k_{\text{SE}}(r) = \exp\left(-\frac{r^2}{2l^2}\right), \tag{2.6}$$

where $r = \|\mathbf{x} - \mathbf{x}'\|$ is the distance between the considered points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. This kernel function leads to smooth behavior, as seen in Fig. 2.6. The squared exponential kernel is a special case within the class of Matérn kernels

$$k_{\text{Matérn}}(r) = \frac{2^{1-v}}{\Gamma(v)} \cdot \left(\frac{\sqrt{2v}r}{l}\right)^v \cdot K_v\left(\frac{\sqrt{2v}r}{l}\right),$$

where K_v is a modified Bessel function [1], Γ the gamma function and $l > 0$ a length scale parameter [37]. The squared exponential kernel is derived by using $v \to \infty$. The parameter

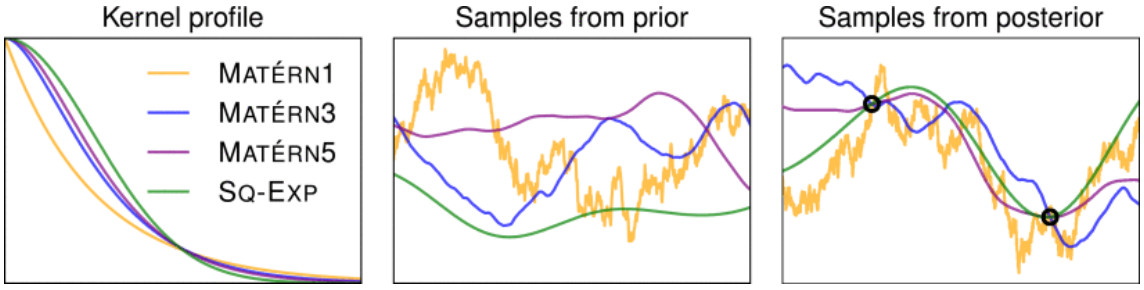


Figure 2.6: (Left): Visualization of various kernel profiles. The horizontal axis represents the distance $r > 0$. (Middle): Samples from GP priors with the corresponding kernels. (Right): Samples from GP posteriors given two data points (black circles). Note the sharper drop in the Matérn1 kernel, which corresponds to the exponential kernel, leads to rough features in the associated samples, while samples from a GP with the Matérn3 and Matérn5 kernels are increasingly smooth. Image from [43]

$v > 0$ controls the smoothness of the functions drawn from the corresponding GP. This function is then $\lfloor v-1 \rfloor$ times differentiable. The influence of v can be seen in Fig. 2.6. The Matérn kernel function becomes simple for a half integer $v = p + \frac{1}{2}$, where p is a positive integer. Within machine learning choosing $v = \frac{3}{2}$ or $v = \frac{5}{2}$ are the most interesting cases:

$$k_{\text{Matérn3}}(r) = \left(1 + \frac{\sqrt{3}r}{l}\right) \exp\left(-\frac{\sqrt{3}r}{l}\right),$$
$$k_{\text{Matérn5}}(r) = \left(1 + \frac{\sqrt{5}r}{l} + \frac{5r^2}{3l^2}\right) \exp\left(-\frac{\sqrt{5}r}{l}\right).$$

For $v = \frac{1}{2}$ the exponential kernel

$$k_{\text{E}}(r) = \exp\left(\frac{-r}{l}\right)$$

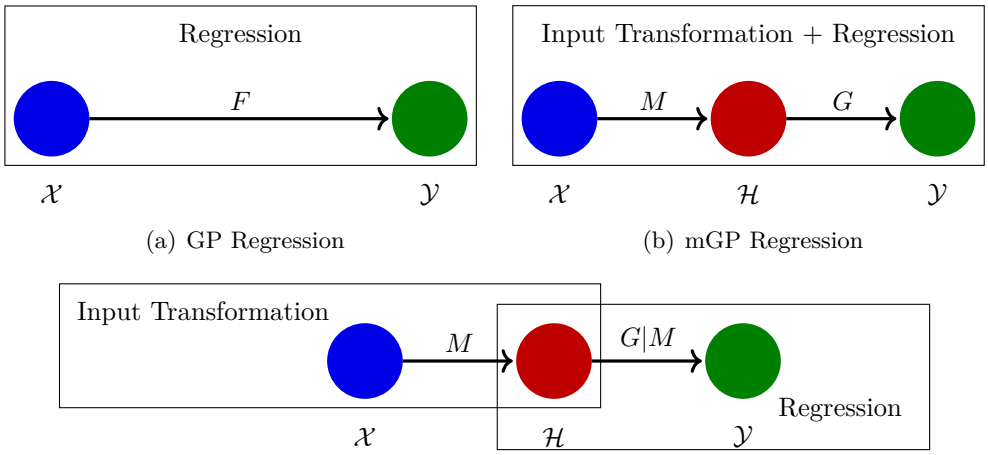
is received, where the GP becames very rough, as seen Fig. 2.6.

2.3.2 Hyperparameter tuning

The kernel and potentially the mean function are parameterized. In order to find good parameters the log marginal likelihood, given by

$$\begin{aligned}\log(p(\mathbf{y}|X,\boldsymbol{\theta})) = &-\frac{1}{2}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}})^T\left(\mathbf{K}_{\boldsymbol{\theta}}+\sigma_n^2\mathbf{I}\right)^{-1}(\mathbf{y}-\mathbf{m}_{\boldsymbol{\theta}}) \\ &-\frac{1}{2}\log\left|\mathbf{K}_{\boldsymbol{\theta}}+\sigma_n^2\mathbf{I}\right| - \frac{n}{2}\log 2\pi,\end{aligned} \tag{2.7}$$

can be maximized. The right hand term in Eq. (2.7) can be split into three parts [43]. The first part quantifies how good the model fits the data. The second term quantifies the complexity of the model. Smoother covariance matrices will have smaller determinants and therefore lead to lower complexity penalties. The last term is a linear function of the number of data points, indicating that the likelihood decreases with a larger dataset.



(a) GP Regression

(b) mGP Regression

(c) Seperatly learnd input transformation M and regression

Figure 2.7: Visualization of the differences of using GP Regression (a) and mGP Regression (b) to fit a function $F : \mathcal{X} \mapsto \mathcal{Y}$. GP Regression maps directly from input space $\mathcal{X}$ to the observation space $\mathcal{Y}$. mGP jointly learns a input transformation $M : \mathcal{X} \mapsto \mathcal{H}$ and a regression from the feature space $\mathcal{H}$ to the output space $\mathcal{Y}$. Please note that the input transformation M is not pre-trained as in (c), where the regression is conditioned by M. Images based on [7].

2.4 Manifold Gaussian Process

In [7], Calandra et al. published the Manifold Gaussian Process (mGP), which jointly learns a transformation from a multi dimensional input space $\mathcal{X} \subseteq \mathbb{R}^D$ into a multi dimensional feature space $\mathcal{H} \subseteq \mathbb{R}^d$, where $d < D$ and a GP Regression from the feature space $\mathcal{H}$ to the observation space $\mathcal{Y} \subseteq \mathbb{R}$. The general goal is to find representations of the input data in a way that they are helpful for the regression task. The performance of GP Regression depends on the kernel function k. By choosing k, assumptions about the smoothness of the target function are made. However, finding an appropriate kernel function k is challenging for certain kind of functions, like highly non-linear functions. By transforming the input space $\mathcal{X}$ into the feature space $\mathcal{H}$ the restriction given by the kernel k can be satisfied more likely. Within the classical regression setting, like GP Regression a transformation $F : \mathcal{X} \mapsto \mathcal{Y}$ is learned. Instead of applying a GP on the input space $\mathcal{X}$, mGP decomposes the transformation F into $F = G \circ M$, with a deterministic and parameterized function $M : \mathcal{X} \mapsto \mathcal{H}$ and a GP Regression $G : \mathcal{H} \mapsto \mathcal{Y}$. The difference is also illustrated in Fig. 2.7. Using $M(\mathbf{X})$ as input for a GP, mGP is equivalent to a GP for a function $F : \mathcal{X} \mapsto \mathcal{Y}$ with a covariance function defined as

$$\tilde{k}(\mathbf{x}_i, \mathbf{x}_j) = k(M(\mathbf{x}_i), M(\mathbf{x}_j)).$$

Therefor the kernel $\tilde{k}$ only operates on the d-dimensional feature space. Thus, the prediction for a test input $\mathbf{x}_*$ using a mGP is similar to the prediction of a GP in Eq. (2.5) and is

obtained by

$$\mu(M(\mathbf{x}_*)) = \mu_0(M(\mathbf{x}_*)) + \tilde{\mathbf{k}}_*^T \left(\tilde{\mathbf{K}} + \sigma_n^2 \mathbf{I}\right)^{-1} (\mathbf{y} - \mathbf{m})$$
$$\sigma(M(\mathbf{x}_*)) = \tilde{k}(M(\mathbf{x}_*), M(\mathbf{x}_*)) - \tilde{\mathbf{k}}_*^T \left(\tilde{\mathbf{K}} + \sigma_n^2 \mathbf{I}\right)^{-1} \tilde{\mathbf{k}}_*,$$

with $\tilde{\mathbf{K}} = \tilde{k}(X, X)$, $\tilde{\mathbf{k}}_* = \tilde{k}(\mathbf{x}_*, X)$ and $m_i = \mu_0(M(x_i))$ for each $x_i \in X$ where μ_0 is defined as in Eq. (2.2). Training a mGP is similar to training a GP. The parameters $\boldsymbol{\theta}_M$ and $\boldsymbol{\theta}_G$ for the transformations M and G are jointly learned using the log marginal likelihood, like in Eq. (2.7). Since the Kernel also depends on the transformation M the log marginal likelihood changes to

$$\begin{aligned} \log(p(\mathbf{y}|X, \boldsymbol{\theta}_{mGP})) = &-\frac{1}{2}(\mathbf{y} - \mathbf{m}_{\boldsymbol{\theta}_{mGP}})^T \left(\tilde{\mathbf{K}}_{\boldsymbol{\theta}_{mGP}} + \sigma_n^2 \mathbf{I}\right)^{-1} (\mathbf{y} - \mathbf{m}_{\boldsymbol{\theta}_{mGP}}) \\ &- \frac{1}{2} \log \left|\tilde{\mathbf{K}}_{\boldsymbol{\theta}_{mGP}} + \sigma_n^2 \mathbf{I}\right| - \frac{n}{2} \log 2\pi, \end{aligned} \tag{2.8}$$

with $\boldsymbol{\theta}_{mGP} = [\boldsymbol{\theta}_M, \boldsymbol{\theta}_G]$. In [7], the transformation M is implemented as a multi-layer ANN, as introduced in Section 2.1. Thereby two different activation functions are considered. Using the log-sigmoid function

$$\text{logsig}(x) = 1/(1 + e^{-x}) \tag{2.9}$$

results into a non-linear transformation M, while the identity function

$$\text{id}(x) = x$$

creates a linear version.

2.5 Multi Output Gaussian Process

GP Regression can only be used to model a single valued function $f : \mathcal{X} \to \mathbb{R}$. A simple approach to extend GP Regression to multi valued functions is to tread each output dimension as a single function and model those functions with separate GPs independently. However, this approach ignores that the different dimensions can be correlated. Formally, a multi dimensional function $f : \mathcal{X} \to \mathbb{R}^D$ is now considered. In general with Multi Output GP, a prediction $\mathbf{g}$ is then drawn from a vector valued GP, such that $\mathbf{g} \sim \mathcal{GP}(\mathbf{m}, \mathbf{K})$, where $\mathbf{m} = \{m_m\}_{m=1}^D$ is a vector of mean functions and $\mathbf{K}$ is a positive valued matrix, which is based on a kernel function k, capturing the correlation between dimensions and data points. The formulation of $\mathbf{K}$ can lead to different model assumptions [34]. This sections covers the Intrinsic Coregionalization Model (ICM) and the Multi Task Gaussian Process (MTGP) approach, which both emerged from different research-fields and are closely related. Both methods are using the Kronecker product $\otimes$ to build $\mathbf{K}$. For a given $m \times n$ matrix $\mathbf{A}$ and a $p \times r$ matrix $\mathbf{B}$ the Kronecker product $\otimes$ is defined as

$$\mathbf{A} \otimes \mathbf{B} = (a_{ij} \cdot B) = \begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix},$$

resulting in a matrix of size $mp \times nr$.

2.5.1 Intrinsic Coregionalization Model

The Intrinsic Coregionalization Model (ICM) [16, 49] uses linear combinations of GP samples to compute the output functions

$$g_m(\mathbf{x}) = \mathbf{a}_m^T \mathbf{u}(\mathbf{x}),$$

for $m = 1, \ldots, D$ where $\mathbf{u}(\mathbf{x})$ is a P-dimensional vector, containing P functions values randomly drawn from the GP prior $\mathcal{GP}(m(\cdot), k(\cdot,\cdot))$, such that $u_i(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x},\mathbf{x}))$. ICM is parameterized by $\mathbf{a}_m \in \mathbb{R}^P$, which are learned from the data. The coefficients and the output functions can be combined to $\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_D] \in \mathbb{R}^{D \times P}$ and $\mathbf{g}(\mathbf{x}) = \mathbf{A}\mathbf{u}(\mathbf{x})$. The covariance is then computed as

$$\operatorname{cov}(\mathbf{g}(\mathbf{x}), \mathbf{g}(\mathbf{x}')) = \mathbf{B} k(\mathbf{x}, \mathbf{x}'),$$

where $\mathbf{B} = \mathbf{A}\mathbf{A}^T \in \mathbb{R}^{D \times D}$ is a positive semi-definite coregionalization matrix. The covariance matrix $\mathbf{K}$ can be written as

$$\mathbf{K} = \mathbf{B} \otimes \overline{\mathbf{K}}, \tag{2.10}$$

where $\left(\overline{\mathbf{K}}\right)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$.

2.5.2 Multi Task Gaussian Process

In [3] Bonilla et al. published the Multi Task Gaussian Process (MTGP), where a task is equal to an output dimension of f. Let there be N training points $\mathbf{x}_1, \ldots, \mathbf{x}_N$, the vector $\mathbf{y} = (y_{11}, \ldots, y_{N1}, y_{12}, \ldots, y_{N2}, \ldots, y_{1D}, \ldots, y_{ND})^T$ contains all target values for the D tasks, where y_{il} is the response for the l-th task on the i-th input $\mathbf{x}_i$. To specify the inter-task similarities a positive semi-definite matrix $\mathbf{K}^f \in \mathbb{R}^{D \times D}$ is learned. The correlation between tasks is then induced by placing a GP prior over the output functions $\{f_i\}$ with

$$\operatorname{cov}(f_i(\mathbf{x}), f_j(\mathbf{x}')) = \left(\mathbf{K}^f\right)_{ij} k(\mathbf{x}, \mathbf{x}'),$$

where k is a kernel function over the inputs. Therefor the kernel matrix $\mathbf{K}$ is

$$\mathbf{K} = \mathbf{K}^f \otimes \overline{\mathbf{K}},$$

where $\left(\overline{\mathbf{K}}\right)_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The prediction of the output values g_l can be computed with the standard GP formulas (see Eq. (2.4) and Eq. (2.5)). Assuming a GP with zero mean, leads to a mean prediction g_l for task l computed as

$$\begin{aligned} g_l(\mathbf{x}_*) &= \left(\mathbf{k}_l^f \otimes \overline{\mathbf{k}}_*\right)^T \Sigma^{-1}\mathbf{y}, \text{ with} \\ \mathbf{\Sigma} &= \mathbf{K}^f \otimes \overline{\mathbf{K}} + \sigma^2 \mathbf{I}_D \otimes \mathbf{I}_N, \end{aligned}$$

where $\mathbf{k}_l^f$ is the l-th column of $\mathbf{K}^f$ and $\mathbf{k}_*$ is the vector of covariances between the test points $\mathbf{x}_*$ and the training points. $\mathbf{I}_D$ and $\mathbf{I}_N$ are a $D \times D$ and a $N \times N$ identity matrix respectively. Therefore, $\mathbf{\Sigma}$ is a $DN \times DN$ matrix. Bonilla et al.[3] showed that for the case of noise-free observations ($\sigma^2 = 0$), there is no inter-task transfer, since the matrix $\mathbf{K}^f$ is no longer relevant for the task prediction.

ICM and MTGP are closely related. Both approaches uses a parameterized matrix, which aims to captures the relation between the output dimensions of the target function f. ICM emerges from a geostatistic research field and with MTGP the field of multi-task learning in the context of GP is addressed. The observation model of MTGP correspond to an ICM model with two processes: one for f and a noise process [3], since MTGP integrates the noise σ^2. In both cases a matrix is learned, $\mathbf{A}$ and $\mathbf{K}^f$ respectively, which then is combined with the kernel matrix $\overline{\mathbf{K}}$ by the Kronecker product. For both approaches the parameters can be jointly learned with the kernel parameters, by maximizing the marginal likelihood [3, 34].

3

Bayesian Optimization in High Dimensional Spaces

BO is a method to tune parameters according to a target function. However, standard BO only performs well for up to 20 parameters that have to be optimized [12, 43]. However, a many problems require the optimization of more than 20 parameters, e.g. finding molecules with certain properties as in [17]. Thus, research in order to adapt BO to perform well in high dimensional settings emerges. This is a difficult and unsolved problem: to ensure a global optimum is found, a good coverage of the parameter space $\mathcal{X}$ is required. The number of evaluations needed to cover $\mathcal{X}$ increases with an increasing number of dimensions d [43].

This chapter covers different approaches within this research domain. All methods have in common that they are reducing the dimension by creating a lower dimensional space. In the following sections an optimization problem $f : \mathcal{X} \to \mathcal{Y}$ with a parameter space $\mathcal{X} \subseteq \mathbb{R}^D$ and $\mathcal{Y} \subseteq \mathbb{R}$ is considered, where D is assumed to be high. Additionally, the approaches discussed within this chapter, define a lower dimensional space $\mathcal{H} \subseteq \mathbb{R}^d$, with $d < D$. In Section 3.1 Bayesian Optimization with Dropouts is introduced, which is inspired by the Dropout algorithm for neural networks. Section 3.2 covers Bayesian Optimization with Random Embedding using a randomly created embedding and a mapping into the input space $\mathcal{X}$ to optimize the target function. Section 3.3 introduces Manifold Bayesian Optimization which combines mGP and Multi-Output GP to create a feature space $\mathcal{H}$ and a corresponding reconstruction of the input space $\mathcal{X}$. In Section 3.4 a new extension of BO, called Hierarchical Acquisition Functions for Bayesian Optimization, is introduced. Thereby the feature space $\mathcal{H}$ is used to condition the input space $\mathcal{X}$ during the optimization process.

Next to the approaches mentioned above, there exits additional work which is not further considered within this thesis. For example in [22] it is assumed that the objective function is the sum of lower dimensional functions and BO is applied on sub spaces of $\mathcal{X}$. In [48] Principal Component Analysis (PCA) is used to randomly select a subset of dimensions which are optimized. However, PCA requires an appropriate number of training data to perform well, which is not always accessible in the context of BO. If the transitions from and to the feature space $\mathcal{H}$ are known, $\mathcal{H}$ can be incorporated into the kernel of the underling GP. However, this requires expert knowledge or a suitable amount of data to train a auto encoder neural net like in [17].

Algorithm 3.1: Pseudocode Bayesian Optimization with Dropouts.

algorithm *DBO*
input $f_{\mathbf{x}}$ // target function
N // number of iterations
output $\mathbf{x}^*$// optimized parameter
for $n = 1$ *to* N *do*
randomly select d dimensions
$\mathbf{x}_n^d \leftarrow \text{argmax}_{\mathbf{x}^d \in \mathcal{X}^d}\ \alpha(\mathbf{x}^d; \mathbb{D}_{n-1})$
$\mathbf{x}_n^{D-d} \leftarrow$ fill_in_strategy()
$\mathbf{x}_n \leftarrow [\mathbf{x}_n^d, \mathbf{x}_n^{D-d}]$
$\mathbb{D}_n \leftarrow \{\mathbb{D}_{n-1}, (\mathbf{x}_n, f_{\mathbf{x}}(\mathbf{x}_n))\}$
randomly select d dimensions
$\mathbf{x}^{*d} \leftarrow \text{argmax}_{\mathbf{x}^d \in \mathcal{X}^d}\ \alpha(\mathbf{x}^d; \mathbb{D}_N)$
$\mathbf{x}^{*D-d} \leftarrow$ fill_in_strategy()
$\mathbf{x}^* \leftarrow [\mathbf{x}^{*d}, \mathbf{x}^{*D-d}]$

3.1 Bayesian Optimization with Dropouts

Dropout was introduced in [46] as a technique to improve training of artificial neural networks. The general idea is to drop randomly chosen artificial neurons and their corresponding weights while propagating training data through the net. Dropouts can be used to decrease the chance of overfitting and leads to more generalization in an artificial neural network [46]. Motivated by the dropout algorithm for artificial neural networks, Li et al. [29] introduced dropout for high dimensional BO. Their algorithm is therefore called Bayesian Optimization with Dropouts (DBO). At each iteration d out of D dimensions are chosen randomly. The optimization of the acquisition function in the current iteration is only performed on the d selected dimensions. The space containing the d selected dimensions is denoted as $\mathcal{X}^d$ and the space with the left out dimensions as $\mathcal{X}^{D-d}$, such that $\mathcal{X} = [\mathcal{X}^d, \mathcal{X}^{D-d}]$. In Alg. 3.1, the pseudocode of DBO can be seen. In order to evaluate the objective function, values for the left out dimension are necessary. Therefor Li et al. provide three fill-in strategies to create $\mathbf{x}^{D-d} \in \mathcal{X}^{D-d}$. With *Dropout-Random* a random value within the domain of $\mathcal{X}^{D-d}$ is used, such as

$$\mathbf{x}^{D-d} \sim \mathcal{U}(\mathcal{X}^{D-d}),$$

where $\mathcal{U}$ is a uniform distribution. This strategy does not work effectively with a large number of influential dimensions. But since $D-d$ dimension have to be filled in, it can still lead to improvements. *Dropout-Copy* copies the required values from the best solution so far:

$$\begin{aligned} \mathbf{x}^+ &= \text{argmax}_{\mathbf{x} \in \mathbb{D}_n} f_{\mathbf{x}}(\mathbf{x}) \\ \mathbf{x}^{D-d} &= \left(\mathbf{x}^+\right)^{D-d}, \end{aligned}$$

where $\mathbf{x}^+$ is the variable of the best found function value after n iterations. With this approach the currently incumbent solution is improved with each iteration. That leads to

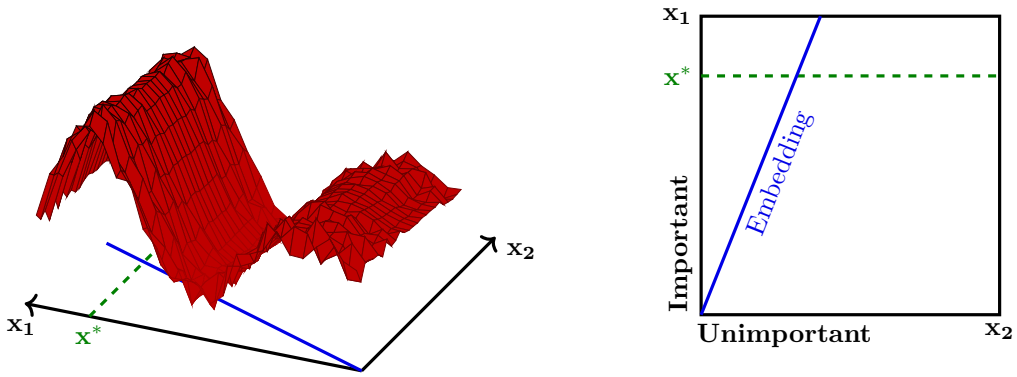


Figure 3.1: This function in $D = 2$ dimensions only has $d_e = 1$ effective dimension: the vertical axis indicated with the word important on the right hand side figure. Hence, the one-dimensional random embedding (blue line) includes the two-dimensional function's optimizer (green dashed line). It is more efficient to search for the optimum along the one-dimensional random embedding than in the original two-dimensional space. Based on [51].

the risk of getting stuck in a local optimum. This is solved by *Dropout Mix*, which combine both previous strategies. With a probability p random values are chosen. Otherwise the values are copied from the best solution so far. Based on the results in [29] it is recommended to use Dropout-Copy or in high dimensions Dropout-Mix with a small p, like $p = 0.1$ or $p = 0.2$.

3.2 Bayesian Optimization with Random Embedding

Wang et al. [50, 51] present an algorithm called Bayesian Optimization with Random Embedding (REMBO). They assume that the objective function f has an effective dimensionality d_e, where the effective dimensionality is defined as follows:

Definition 3.1 (effective dimensionality). A function $f : \mathbb{R}^D \to \mathbb{R}$ is said to have **effective dimensionality** d_e, with $d_e < D$, if there exits a linear subspace $\mathcal{T}$ of dimension d_e such that for all $\mathbf{x}_\top \in \mathcal{T} \subset \mathbb{R}^D$ and $\mathbf{x}_\perp \in \mathcal{T}^\perp \subset \mathbb{R}^D$, we have $f(\mathbf{x}) = f(\mathbf{x}_\top + \mathbf{x}_\perp) = f(\mathbf{x}_\top)$, where $\mathcal{T}^\perp$ denotes the orthogonal complement of $\mathcal{T}$. We call $\mathcal{T}$ the **effective subspace** of f and $\mathcal{T}^\perp$ the **constant subspace**.

This means that the value of f only changes along the coordinates $\mathbf{x}_\top$ and remains constant along the coordinates $\mathbf{x}_\perp$. A visual example for the effective and constant subspace can be seen in Fig. 3.1. Wang et al. [50, 51] also give the proof of Theorem 3.2.

Theorem 3.2. *Assume we are given a function $f : \mathbb{R}^D \to \mathbb{R}$ with effective dimensionality d_e and a random matrix $\mathbf{A} \in \mathbb{R}^{D \times d}$ with independent entries sampled according to $\mathcal{N}(0, 1)$ and $d \geq d_e$. Then, with probability 1, for any $\mathbf{x} \in \mathbb{R}^D$, there exists a $\mathbf{h} \in \mathbb{R}^d$ such that $f(\mathbf{x}) = f(\mathbf{A}\mathbf{h})$.*

Following Theorem 3.2 the function $f(\mathbf{A}\mathbf{h})$ can be optimized in a lower dimensional space, instead of being optimized in higher dimensions. If an optimizer $\mathbf{x}^* \in \mathcal{X}$ exists, then there

Algorithm 3.2: Pseudocode Bayesian Optimization with Random Embedding.

algorithm *REMBO*
input $f_{\mathbf{x}}$ // target function
N // number of iterations
$\mathcal{H}$ // Embedding
output $\mathbf{x}^*$ // optimized parameter
$\mathbf{A} \leftarrow random(D, d; \mathcal{N}(0,1))$
for $n = 1$ *to* N *do*
$\mathbf{h}_n \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \mathbb{D}_{n-1})$
$\mathbb{D}_n \leftarrow \{\mathbb{D}_{n-1}, (\mathbf{A}\mathbf{h}_n, f_{\mathbf{x}}(\mathbf{A}\mathbf{h}_n))\}$
$\mathbf{h}^* \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \mathbb{D}_N)$
$\mathbf{x}^* \leftarrow \mathbf{A}\mathbf{h}^*$

is an optimizer $\mathbf{h}^* \in \mathcal{H}$ with $f(\mathbf{x}^*) = f(\mathbf{A}\mathbf{h}^*)$. $\mathbf{A}$ acts as a mapping form $\mathcal{H}$ to $\mathcal{X}$ and is required to be constant during the optimization. Based on Theorem 3.2 REMBO first draws a random embedding (given by $\mathbf{A}$) and then performs BO in the embedded space $\mathcal{H}$, as shown in Alg. 3.2. Unlike BO, the dataset $\mathbb{D}_n = \{\mathcal{H}, \mathcal{Y}\}$ within REMBO contains the previous visited embeddings and the corresponding values of $f_{\mathbf{x}}$. Like the parameter space $\mathcal{X}$ the embedded space $\mathcal{H}$ has to be bounded. Therefor, it is important how the bounds for $\mathcal{H}$ are chosen. Locating the optimum within $\mathcal{H}$ is easier if $\mathcal{H}$ is small, but if $\mathcal{H}$ is chosen too small it may not contain the global optimizer, which is visualized in Fig. 3.2. Wang et al. [50, 51] were able to show that for a box-constrained $\mathcal{X}$ and an optimizer $\mathbf{x}^* \in \mathcal{X}$, with a probability of at least $1 - \epsilon$, there exits an optimizer $\mathbf{h}^* \in \mathcal{H}$ with

$$\|\mathbf{h}^*\|_2 \ \leq\ \frac{\sqrt{d_e}}{\epsilon}\|\mathbf{x}_*\|_2$$

and $f(\mathbf{A}\mathbf{h}^*) = f(\mathbf{x}^*)$. For a box constrain of $[-1, 1]^D$ the upper bound can be written as

$$\|\mathbf{h}_*\|_2 \ \leq\ \frac{\sqrt{d_e}}{\epsilon}\sqrt{d_e} \ =\ \frac{d_e}{\epsilon}.$$

Therefore the bounds of $\mathcal{H}$ have to be chosen, in a way that a hyper sphere with center $\mathbf{0}$ and radius $\frac{d_e}{\epsilon}$ is contained by $\mathcal{H}$. If $\mathbf{A}\mathbf{h}$ is outside of $\mathcal{X}$, $\mathbf{A}\mathbf{h}$ has to be projected into $\mathcal{X}$. This is also visualized in Fig. 3.2. Wang et al. choose $\mathcal{H} = [-\sqrt{d}, \sqrt{d}]^d$ for all there experiments.

3.3 Manifold Bayesian Optimization

Manifold Bayesian Optimization (MBO) [34] is a encoder-decoder approach, combining a mGP, for the encoder part and the modeling of the objective function, with a Multi-Output GP for the decoder part. Therefor mGP is used to learn a transformation $M : \mathcal{X} \mapsto \mathcal{H}$ from the input space $\mathcal{X}$ into a feature space $\mathcal{H} \subseteq \mathbb{R}^d$, also called manifold. Depending on its implementation the transformation M can be a linear, but also a non-linear function. The transformation M is jointly learned with a GP Regression $G : \mathcal{H} \mapsto \mathcal{Y}$. Like BO, MBO

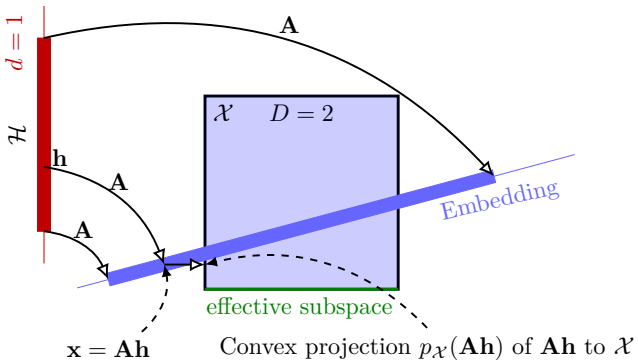
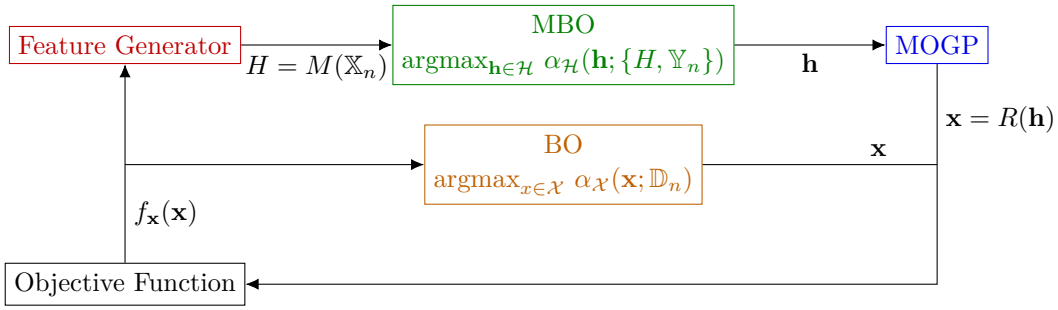


Figure 3.2: Embedding from d = 1 into D = 2. The blue box illustrates the 2D constrained space $\mathcal{X}$, while the thicker red line illustrates the 1D constrained space $\mathcal{H}$. Note that if $\mathbf{Ah}$ is outside $\mathcal{X}$, it is projected onto $\mathcal{X}$. The set $\mathcal{H}$ must be chosen large enough so that the projection of its image $\mathbf{A}\mathcal{H}$, onto the effective subspace (vertical green axis in this diagram) covers the vertical side of the box. Based on [51].

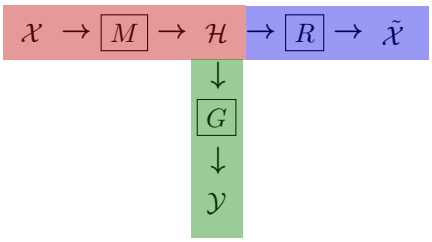
Algorithm 3.3: Pseudocode Manifold Bayesian Optimization.

algorithm *MBO*
input $f_{\mathbf{x}}$ // target function
N // number of iterations
output $\mathbf{x}^*$ // optimized parameter
for $n = 1$ *to* N *do*
$\mathbf{h}_n \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \{M(\mathbb{X}_{n-1}), \mathbb{Y}_{n-1}\})$ // apply BO on feature space
$\mathbf{x}_n \leftarrow R(\mathbf{h}_n)$ // map feature to parameter space
$\mathbb{X}_n \leftarrow \{\mathbb{X}_{n-1}, \mathbf{x}_n\}$
$\mathbb{Y}_n \leftarrow \{\mathbb{Y}_{n-1}, f_{\mathbf{x}}(\mathbf{x}_n)\}$
$\mathbf{h} \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \{M(\mathbb{X}_N), \mathbb{Y}_N\})$
$\mathbf{x}^* \leftarrow R(\mathbf{h})$

stores the n previous evaluated parameters $\mathbb{X}_n$ and the corresponding values of $f_{\mathbf{x}}$ within the dataset $\mathbb{D}_n = \{\mathbb{X}_n, \mathbb{Y}_n\}$. In standard BO the parameters for the next evaluation are gained by maximizing the acquisition function $\alpha_{\mathcal{X}} : \mathcal{X} \mapsto \mathbb{R}$. Instead of optimizing on the parameter space, MBO uses the lower dimensional feature space $\mathcal{H}$ and GP Regression, created by the mGP. Therefore, MBO maximizes the acquisition function $\alpha_{\mathcal{H}} : \mathcal{H} \mapsto \mathbb{R}$. This results into a feature $\mathbf{h} = \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha_{\mathcal{H}}(\mathbf{h}; \{M(\mathbb{X}_n), \mathbb{Y}_n\})$, with $M(\mathbb{X}_n)$ as the feature representation of $\mathbb{X}_n$. Multi-Output GP is then used to learn a transformation $R : \mathcal{H} \mapsto \tilde{\mathcal{X}}$ from the manifold back into the parameter space. The next parameter to evaluate $f_{\mathbf{x}}$ is computed with $\tilde{\mathbf{x}} = R(\mathbf{h})$. This process is visualized in Fig. 3.3. Additionally, the pseudocode of MBO can be seen in Alg. 3.3. Similar to [7], Moriconi et al.[34] implement M as a multi layer ANN. In their experiments an architecture of $D - 20 - d$ is used, with a single hidden layer of 20 units. For the transformation R ICM is used. The parameters for the neural net, GP



(a) MBO compared to BO



(b) Transformations in MBO. Based on [34].

Figure 3.3: Visualization of MBO. In (a) the general process of MBO is visually compared to BO (orange). In (b) the relation between the transformation within MBO is visualized. In total three transformations $M : \mathcal{X} \mapsto \mathcal{H}, G : \mathcal{H} \mapsto \mathcal{Y}$ and $R : \mathcal{H} \mapsto \tilde{\mathcal{X}}$ are jointly learned. The transformation M (red) creates a manifold $\mathcal{H}$. Using a GP G, BO is applied on the feature space to find a new feature optimizing the target function (green). The red and green area correspond to a mGP solving a regression task. In a last step a Multi-Output GP R is used to map the feature back into the input space (blue).

and ICM are jointly trained by maximizing a joint log marginal likelihood, given by

$$\begin{aligned}\log(p(\mathbf{y}, \tilde{\mathbf{X}}|X, \boldsymbol{\theta})) = &-\frac{1}{2}\mathbf{y}^T\left(\tilde{\mathbf{K}} + \sigma_n^2\mathbf{I}\right)^{-1}\mathbf{y} - \frac{1}{2}\log\left|\tilde{\mathbf{K}} + \sigma_n^2\mathbf{I}\right| \\ &- \frac{1}{2}\mathbf{x}_V^T\mathbf{K}^{-1}\mathbf{x}_V - \frac{1}{2}\log|\mathbf{K}|\,,\end{aligned}$$

with the covariance matrix of the mGP $\tilde{\mathbf{K}}$ as defined in Eq. (2.8) and the kernel matrix of the ICM $\mathbf{K}$ as in Eq. (2.10). The vector $\mathbf{x}_V$ is obtained by stacking all parameters within $\mathbb{X}_n$ and $\mathbf{y}$ is the vectored representation of $\mathbb{Y}_n$. The reconstructed parameter space is symbolized with $\tilde{\mathbf{X}}$. By jointly learning the parameters, the weights and biases of the mGP are not only optimized for the regression task, but also consider the reconstruction of the input space. Since ICM and MTGP are closely related, the decoder part within this thesis is implemented with a MTGP.

3.4 Hierarchical Acquisition Functions for Bayesian Optimization

Hierarchical Acquisition Functions for Bayesian Optimization (HIBO) aims to condition the search space using features. The feature generation can incorporate expert knowledge. An example can be the learning of controller parameters for a certain task in robotics. During testing proposed parameter settings, properties like velocities or positions can be observed and used as features. However, there are cases where the creation of handcrafted features is not achievable. Therefore, in this thesis the simultaneously learning of the features in the context of HIBO is considered and a mGP is used to generate the feature space $\mathcal{H}$. The mGP provides two transformations. The features are created by the transformation $M : \mathcal{X} \mapsto \mathcal{H}$, which can be implemented as an ANN. The second transformation $G : \mathcal{H} \mapsto \mathcal{Y}$ is a GP solving a regression task from the features to the target values of the objective function. The dataset $\mathbb{D}_n = \{\mathbb{X}_n, \mathbb{Y}_n\}$, consist of the n previously evaluated parameters $\mathbb{X}_n$ and the corresponding return values of the objective function $\mathbb{Y}_n = f_{\mathbf{x}}(\mathbb{X}_n)$.
In order to condition the search space, two acquisition functions are used in a hierarchical manner. At first an acquisition function $\alpha_{\mathcal{H}} : \mathcal{H} \mapsto \mathbb{R}$ is applied on the feature space, resulting in a feature potentially optimizing the objective function

$$\mathbf{h}_n = \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha_{\mathcal{H}}(\mathbf{h}; \{M(\mathbb{X}_{n-1}), \mathbb{Y}_{n-1}\}),$$

where $\alpha_{\mathcal{H}}$ uses the GP G from the mGP. The parameter search space is then conditioned, by concatenating $\mathcal{X}$ with the feature $\mathbf{h}_n$, resulting in $\mathcal{Z} = [\mathcal{X}, \mathbf{h}_n] \subset \mathbb{R}^{D+d}$. To receive the next parameter $\mathbf{x}_n$ for evaluation, a second acquisition function $\alpha_{\mathcal{Z}} : \mathcal{Z} \mapsto \mathbb{R}$ is used, resulting in

$$\mathbf{z}_n = \text{argmax}_{\mathbf{z}\in\mathcal{Z}}\ \alpha_{\mathcal{Z}}(\mathbf{z}; \{[\mathbb{X}_{n-1}, M(\mathbb{X}_{n-1})], \mathbb{Y}_{n-1}\}).$$

The first D elements of $\mathbf{z_n}$ are then used as $\mathbf{x}_n$. In the end, the dataset $\mathbb{D}_n = \{\mathbb{D}_{n-1}, (\mathbf{x}_n, f_{\mathbf{x}}(\mathbf{x}_n))\}$ is updated. The pseudocode for HIBO can be seen in Alg. 3.4 and a visualization of the process within HIBO is in Fig. 3.4. Please note that the parameters of the GPs, used by the acquisition functions $\alpha_{\mathcal{H}}$ and $\alpha_{\mathcal{Z}}$, are not jointly learned, since the GP for $\alpha_{\mathcal{Z}}$ is conditioned by $M(\mathbb{X}_{n-1})$. The parameters for the mGP $\boldsymbol{\theta}_{mGP}$, including the transformation M and the GP regression G are learned by optimizing a log marginal likelihood like in Eq. (2.8) with

$$\begin{aligned}\log(p(\mathbf{y}_n|\mathbb{X}_n, \boldsymbol{\theta}_{mGP})) = &-\frac{1}{2}(\mathbf{y}_n - \mathbf{m}_{\boldsymbol{\theta}_{mGP}})^T \left(\tilde{\mathbf{K}}_{\boldsymbol{\theta}_{mGP}} + \sigma_n^2\mathbf{I}\right)^{-1} (\mathbf{y}_n - \mathbf{m}_{\boldsymbol{\theta}_{mGP}}) \\ &-\frac{1}{2}\log\left|\tilde{\mathbf{K}}_{\boldsymbol{\theta}_{mGP}} + \sigma_n^2\mathbf{I}\right| - \frac{n}{2}\log 2\pi,\end{aligned}$$

where $\mathbf{y}_n$ is a vectorized version of $\mathbb{Y}_n$. The GP Regression G is defined by the mean function $\mathbf{m}_{\boldsymbol{\theta}_{mGP}}$ and the kernel $\tilde{\mathbf{K}}_{\boldsymbol{\theta}_{mGP}}$, which are both influenced by the transformation M. The GP used for the second acquisition function $\alpha_{\mathcal{Z}}$ is trained, similar to Eq. (2.7), with

$$\begin{aligned}\log(p(\mathbf{y}_n|\left[\mathbb{X}_n, M(\mathbb{X}_n)\right], \boldsymbol{\theta})) = &-\frac{1}{2}(\mathbf{y}_n - \mathbf{m}_{\boldsymbol{\theta}})^T \left(\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right)^{-1} (\mathbf{y}_n - \mathbf{m}_{\boldsymbol{\theta}}) \\ &-\frac{1}{2}\log\left|\mathbf{K}_{\boldsymbol{\theta}} + \sigma_n^2\mathbf{I}\right| - \frac{n}{2}\log 2\pi.\end{aligned}$$

This GP is defined by the mean function $\mathbf{m}_{\boldsymbol{\theta}}$ and the kernel $\mathbf{K}_{\boldsymbol{\theta}}$.

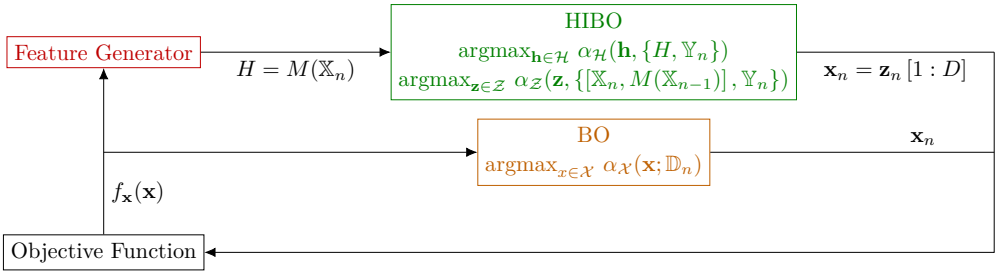


Figure 3.4: Visualization of HIBO compared to BO (orange). A mGP provides the transformation M, which is used to generate features H (red). Based on the mGP a potentially optimal feature $\mathbf{h}_n$ for the objective function is computed, by maximizing $\alpha_{\mathcal{H}}$. The feature $\mathbf{h}_n$ is used to condition the parameter space $\mathcal{X}$. In a second step the acquisition function $\alpha_{\mathcal{Z}}$ is maximized to receive the next parameter $\mathbf{x}_n$ for the evaluation of the objective function (green).

Algorithm 3.4: Pseudocode Hierarchical Acquisition Functions for Bayesian Optimization.

algorithm *HIBO*
input $f_{\mathbf{x}}$ // target function
N // number of iterations
output $\mathbf{x}^*$ // optimized parameter
for $n = 1$ *to* N *do*
$\mathbf{h}_n \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \{M(\mathbb{X}_{n-1}), \mathbb{Y}_{n-1}\})$
$\mathbf{z}_n \leftarrow \text{argmax}_{\mathbf{z}\in\mathcal{Z}}\ \alpha(\mathbf{z}; \{[\mathbb{X}_{n-1}, M(\mathbb{X}_{n-1})], \mathbb{Y}_{n-1}\})$
$\mathbf{x}_n \leftarrow \mathbf{z}_n\,[1:D]$
$\mathbb{X}_n \leftarrow \{\mathbb{X}_{n-1}, \mathbf{x}_n\}$
$\mathbb{Y}_n \leftarrow \{\mathbb{Y}_{n-1}, f_{\mathbf{x}}(\mathbf{x}_n)\}$
$\mathbf{h} \leftarrow \text{argmax}_{\mathbf{h}\in\mathcal{H}}\ \alpha(\mathbf{h}; \{M(\mathbb{X}_N), \mathbb{Y}_N\})$
$\mathbf{z} \leftarrow \text{argmax}_{\mathbf{z}\in\mathcal{Z}}\ \alpha(\mathbf{z}; \{[\mathbb{X}_N, M(\mathbb{X}_{n-1})], \mathbb{Y}_N\})$
$\mathbf{x}^* \leftarrow \mathbf{z}\,[1:D]$

4

Experiments

Within this chapter, the performance of the HIBO algorithm is evaluated and compared to other methods. Therefore HIBO, BO, DBO, REMBO and MBO are applied on different problems. In Section 4.1 three benchmark functions are used to compare the performance of these approaches. The benchmark functions are embedded into higher dimensions. The dimensionality varies, in order to show the capabilities in different high dimensional settings. In Section 4.2 a simple air hockey simulation is used as a more realistic robotic inspired problem. The target is to learn the trajectory of the mallet to score a goal from a fixed position.

For all algorithms the EI acquisition function from Eq. (2.1) is used. The acquisition functions are sampled at 1000 randomly chosen points, to find a point that maximizes the acquisition function. The kernel function of the used GP is implemented with the squared exponential kernel function from Eq. (2.6). The DBO algorithm is used with the *Dropout Mix* fill-in strategy and $p = 0.15$. For REMBO, the embedding is set to $\mathcal{H} = [-\sqrt{d}, \sqrt{d}]^d$, where d is the number of dimensions of the embedding. The projection $\mathbf{A}\mathcal{H}$ is clipped to the range of $[-1, 1]$ and is then scaled to the size of the parameter space $\mathcal{X}$. MBO and HIBO use a mGP with an ANN. The architecture is set to $[D - 20 - d]$, with fully connected layers, where D is the dimensionality of the parameter space and d the size of the feature space. The activation function is implemented with the log-sigmoid function from Eq. (2.9). Therefor the feature generation for MBO and HIBO is non-linear. The reconstruction of the parameter space $\mathcal{X}$ in MBO is implemented with a MTGP. Therefor the implementation by Bonilla et al. is used[1]. The experiments are implemented and executed with Matlab[2]. Tuning the hyper parameters, in equal minimizing the log marginal likelihood is implemented with the fminunc function in Matlab, using the BFGS Quasi-Newton method [6, 11, 15, 44] with a cubic line search procedure.

4.1 Synthetic Benchmark Functions

Within this section, BO and its extensions are applied on three common benchmark functions: Branin function, Schwefel function and Ackley function. To evaluate the perfor-

[1] https://github.com/ebonilla/mtgp
[2] https://www.mathworks.com/products/matlab.html

Table 4.1: Overview of the parameters used for the benchmark experiments. The d_b-dimensional benchmark function is embedded in a D dimensional space. Depending on the algorithm a d-dimensional feature space is created. Each optimization process is performed for n iterations and is repeated N times.

D	d	d_b **Branin**	d_b **Ackley & Schwefel**	n	N
10	2	2	2	20	200
30	5	2	5	30	200
50	10	2	10	30	200

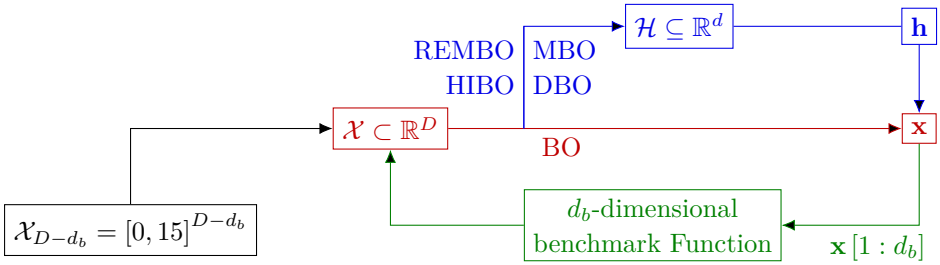
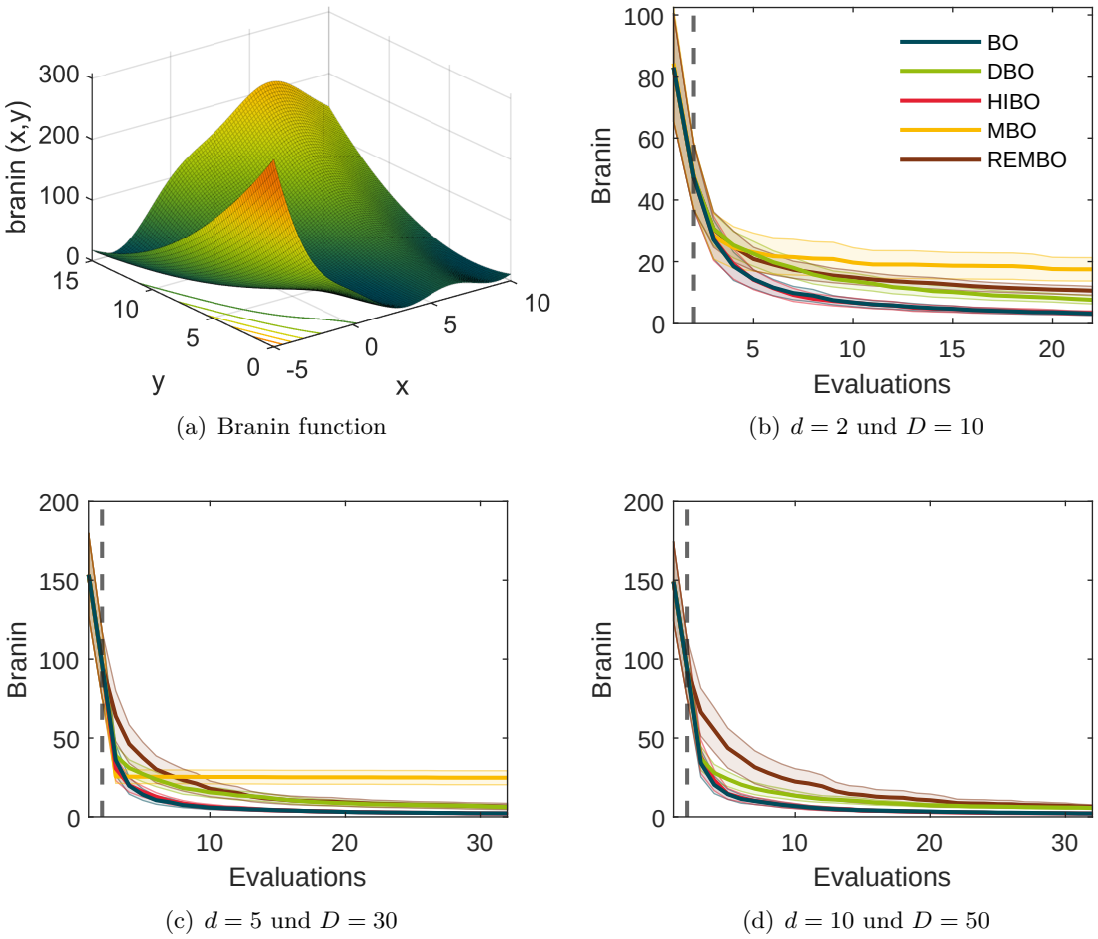


Figure 4.1: A d_b-dimensional benchmark function is used (green). The benchmark function is embedded into a D-dimensional space $\mathcal{X}$, where $D - d_b$ dimension have no influence onto the benchmark function. BO is directly applied to $\mathcal{X}$, while its extensions create a d-dimensional feature space $\mathcal{H}$ (blue). All algorithms return a D-dimensional parameter $\mathbf{x}$. Only the fist d_b of $\mathbf{x}$ are used to evaluate the benchmark function.

mance in different dimensional situations, the following setting is applied. In the case of the Ackley and the Schwefel functions, a d_b-dimensional version is created, resulting in a parameter space $\mathcal{X}_{d_b}$. The Branin function is constrained to $d_b = 2$ dimensions. Now $\mathcal{X}_{d_b}$ is embedded into a higher dimensional parameter space $\mathcal{X} = [\mathcal{X}_{d_b}, \mathcal{X}_{D-d_b}]$, with $\mathcal{X}_{D-d_b} = [0, 15]^{D-d_b} \subset \mathbb{R}^{D-d_b}$. The parameter space $\mathcal{X}$ consists of $D > d_b$ dimensions, but only the first d_b dimensions are used to evaluate the benchmark function. HIBO, MBO, DBO and REMBO compute, a d-dimensional feature space $\mathcal{H} \subset \mathbb{R}^d$, according to the respective procedure. A representation of the setting can be see in Fig. 4.1. Overall three different settings, regarding the dimensionality, are tested. Tab. 4.1 shows the used parameters for each setting. Depending on the dimensional size, the algorithms operate over $n = 20$ or $n = 30$ iterations. In order to ensure statistical stability, the optimization process is repeated $N = 200$ times for each algorithm and benchmark function. Each optimization process is initialized with a dataset $\mathbb{D}$, including 2 samples. For each of the $N = 200$ repetitions, distinguish seed points are randomly chosen. All initial samples are taken from the embedding of the REMBO algorithm and are projected into the parameter space $\mathcal{X}$, using the random projection matrix $\mathbf{A}$ of REMBO. Thereby all algorithms share the same initial points in the dataset. Since the data set is initialized with two samples, the benchmark function is evaluated $n + 2$ times. Due to optimizing the hyper parameters, used by the GPs and ANNs, the five algorithms hold different running times. To optimize a benchmark function with $d = 10$ and $D = 50$, the implementation used for MBO, endures for about 6 hours. Hence, MBO will not be considered in the evaluation for $d = 10$ and $D = 50$.



(a) Branin function

(b) $d = 2$ und $D = 10$

(c) $d = 5$ und $D = 30$

(d) $d = 10$ und $D = 50$

Figure 4.2: (a) Plot of the 2D Branin function in the interval of $x \in [-5, 10]$ and $y \in [0, 15]$. (b)-(d) Mean and $\frac{1}{4}$ standard deviation over 200 trails, optimizing the Branin function. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process.

4.1.1 Branin function

The Branin function is defined as

$$\text{branin}(x, y) = a \cdot (y - bx^2 + cx - r)^2 + s \cdot (1 - t) \cdot \cos(x) + s,$$

where it is recommended to use $a = 1$, $b = \frac{5.1}{4\pi^2}$, $c = \frac{5}{\pi}$, $r = 6$, $s = 10$ and $t = \frac{1}{8\pi}$. A plot of the Branin function can be seen in Fig. 4.2(a). The Branin function is typically minimized in the interval of $x \in [-5, 10]$ and $y \in [0, 15]$. Within this interval the Branin function owns three global optima at $\mathbf{x}_1^* = (-\pi, 12.275)$, $\mathbf{x}_2^* = (\pi, 2.275)$, $\mathbf{x}_3^* = (3\pi, 2.475)$, with $\text{branin}(\mathbf{x}_i^*) = 0.3978873$ [21]. The results are visualized in Fig. 4.2 and data about the best solution found, is provided in Tab. 4.2. In all settings BO and HIBO are able to optimize the Branin function the best. Additionally, both algorithms perform at a same level during the optimization process. However, in average an optimal solution, with value 0 is not found. But the mean value of the solutions found by BO and HIBO varies between

Table 4.2: Mean and standard deviation of the best parameter value on the Branin function, found during the optimization process. The values are computed based on 200 repetitions.

Algorithm	$d = 2$ $D = 10$	$d = 5$ $D = 30$	$d = 10$ $D = 50$
BO	2.97 ± 2.57	2.19 ± 1.74	2.13 ± 1.8
DBO	7.46 ± 5.32	6.18 ± 4.89	5.59 ± 3.6
HIBO	3.14 ± 2.88	2.08 ± 1.61	2.08 ± 1.55
MBO	17.40 ± 15.3	24.71 ± 17.76	
REMBO	10.45 ± 5.77	6.54 ± 8.04	6.34 ± 5.17

2 and 3. The results for REMBO improves with higher dimensions, but is not able to reach the quality of BO and HIBO. The same holds for DBO, whereas DBO performs better in the low dimensional setting, compared to REMBO. Additionally, the results of DBO improve with higher dimensions, as seen in Tab. 4.2. MBO only significantly improves in the beginning. After a few iterations, the quality of the best solutions so far remains equal. Additionally, MBO has the highest variation within the results.

4.1.2 Schwefel function

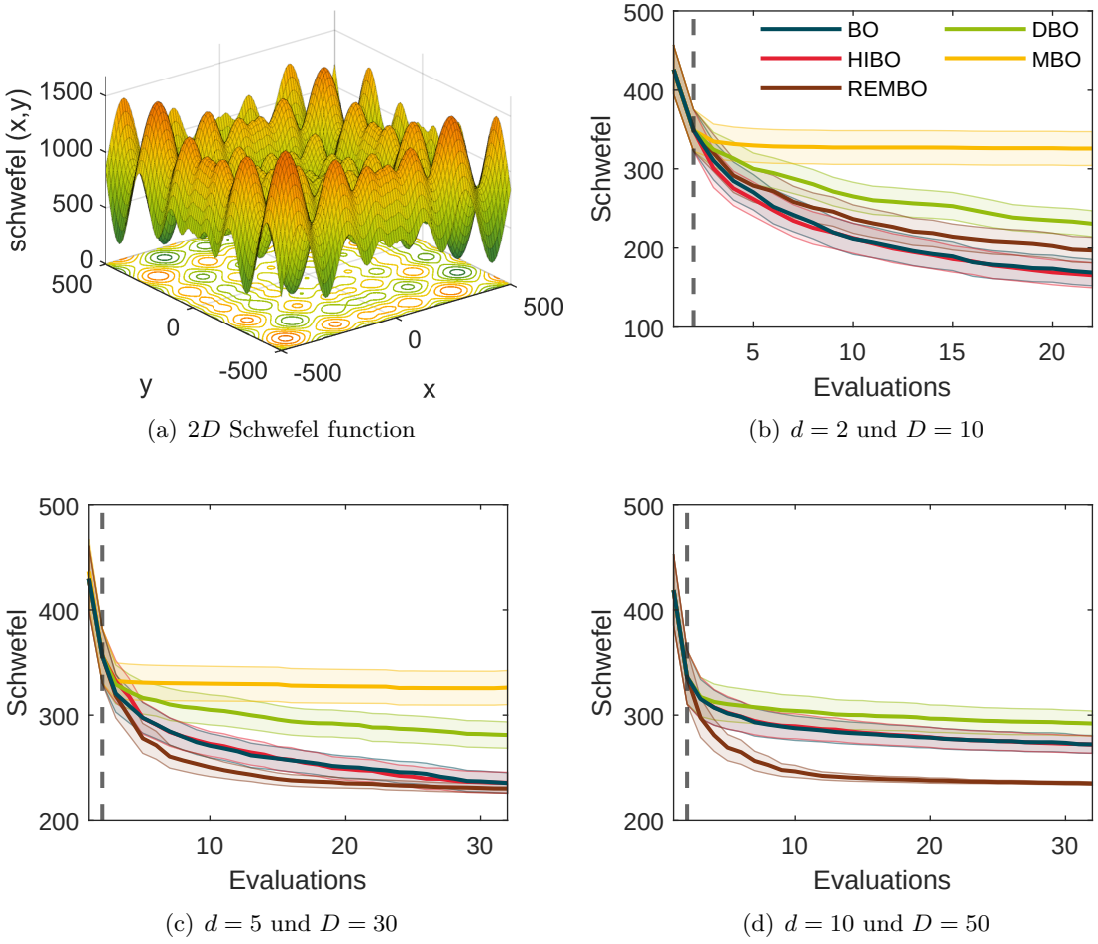
In Literature, the Schwefel function is defined as

$$\text{schwefel}(\mathbf{x}) = 418.983 \cdot d_b - \sum_{i=1}^{d_b} x_i \cdot \sin\left(\sqrt{|x_i|}\right).$$

In order to ensure the same value range for different d_b, the Schwefel function can be scaled to

$$\text{schwefel}(\mathbf{x}) = 418.983 - \frac{1}{d_b} \sum_{i=1}^{d_b} x_i \cdot \sin\left(\sqrt{|x_i|}\right).$$

A plot of the $2D$ Schwefel function can be seen in Fig. 4.3(a). The Schwefel function is typically minimized in the interval of $x_i \in [-500, 500]$. The global minimum of the Schwefel function is at $x_i^* = 420.9687$ for $i = 1, \dots d_b$, with $\text{schwefel}(\mathbf{x}^*) = 0$ [21]. The Schwefel function hold many local optima. Local optima, with similar values to the global optimum, are spread all over the interval. Hence, there are a lot of local optima, promising nearly optimal values. However, only one location is optimal. The results of optimizing the Schwefel function are shown in 4.3 and the mean and standard deviation values of the final solutions are listed in Tab. 4.3. The performance level of BO and HIBO are similar across all settings. In the low dimensional setting, with $d = 2$ and $D = 10$, both algorithms performs best. However, REMBO performs superior to BO and HIBO, when the number of dimensions increases. Overall, the optimization with REMBO obtain the best results in higher dimensions. DBO always achieves results slightly inferior to BO and HIBO. The worst performance is achieved by MBO. Improving solutions are only found in the first iterations. The value level remains equal during the rest of the optimization. For an increasing number of dimensions, all algorithms have in common, that the overall found solutions recede further away from the optimal solution and the corresponding variances decrease.



(a) $2D$ Schwefel function

(b) $d = 2$ und $D = 10$

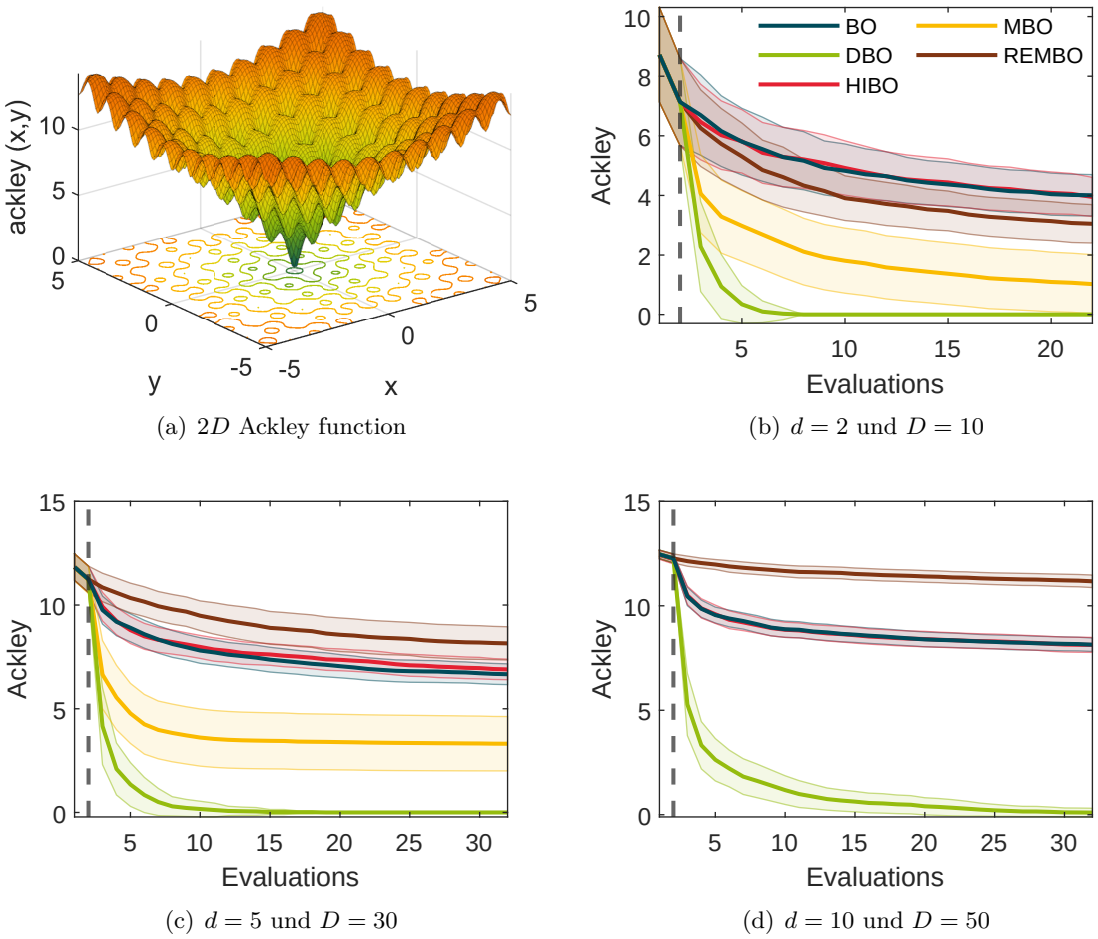
(c) $d = 5$ und $D = 30$

(d) $d = 10$ und $D = 50$

Figure 4.3: (a) Plot of the 2D Schwefel function, in the interval of $x, y \in [-500, 500]$. (b)-(d) Mean and $\frac{1}{4}$ standard deviation over 200 trails, optimizing the Schwefel function. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process.

Table 4.3: Mean and standard deviation of the best parameter value on the Schwefel function, found during the optimization process. The values are computed based on 200 repetitions.

Algorithm	$d = 2$ $D = 10$	$d = 5$ $D = 30$	$d = 10$ $D = 50$
BO	167.66 ± 68.48	234.15 ± 42.59	270.79 ± 38.96
DBO	228.97 ± 67.96	279.66 ± 53.8	290.57 ± 51.89
HIBO	164.24 ± 64.93	234.47 ± 43.04	270.53 ± 37.83
MBO	324.08 ± 88.8	324.59 ± 68.66	
REMBO	195.89 ± 64.73	228.00 ± 23.59	233.73 ± 17.63



(a) $2D$ Ackley function

(b) $d = 2$ und $D = 10$

(c) $d = 5$ und $D = 30$

(d) $d = 10$ und $D = 50$

Figure 4.4: (a) Plot of the 2D Ackley function, in the interval of $x, y \in [-5, 5]$. (b)-(d) Mean and $\frac{1}{2}$ standard deviation over 200 trails, optimizing the Ackley function. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process.

4.1.3 Ackley function

The Ackley function is defined as

$$\text{ackely}(\mathbf{x}) = -a \exp\left(-b \cdot \sqrt{\frac{1}{d_b}\sum_{i=1}^{d_b} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d_b} \cos\left(c \cdot x_i\right)\right) + a + \exp(1),$$

where it is recommended to use $a = 20$, $b = 0.2$, $c = 2\pi$. A plot of the $2D$ Ackley function can be seen in Fig. 4.4(a). The Ackley function is typically minimized in the interval of $x_i \in [-5, 5]$, for $i = 1, \ldots, d_b$. The global minimum lies in $\mathbf{x}^* = \mathbf{0}$, with $\text{ackley}(\mathbf{x}^*) = 0$ [21]. The results of the optimization process on the Ackley function is shown in Fig. 4.4. The mean and standard deviation values of the best solution found, can be seen in Tab. 4.4. Overall DBO performs best. For $d = 2, D = 10$ and $d = 5, D = 30$ the optimum is found in all repetitions. For $d = 10$ and $D = 50$ the results tend towards

Table 4.4: Mean and standard deviation of the best parameter value on the Ackley function, found during the optimization process. The values are computed based on 200 repetitions.

Algorithm	$d = 2$ $D = 10$	$d = 5$ $D = 30$	$d = 10$ $D = 50$
BO	3.99 ± 1.41	6.63 ± 1.1	8.10 ± 0.86
DBO	0.00 ± 0.0	0.00 ± 0.0	0.10 ± 0.42
HIBO	3.93 ± 1.34	6.86 ± 1.1	8.09 ± 0.92
MBO	1.02 ± 1.3	3.30 ± 2.63	
REMBO	3.03 ± 1.3	8.11 ± 1.7	11.12 ± 0.99

the optimum. Presumable, the optimum can be approximate more closely with a higher number of iterations. For BO, HIBO, MBO and REMBO it holds, that with a higher number of dimensions the results recede more from the optimum value. For $d = 2, D = 10$ and $d = 5, D = 30$ MBO outperforms BO, HIBO and REMBO. However, MBO is not able to approximate the optimal solution as good as DBO. Additionally, the results of MBO hold the highest variance. BO and HIBO achieve similar results in all three settings. However, both algorithms do not optimize the Ackley function well. In the low dimensional experiment, REMBO performs superior to BO and HIBO. However, with higher dimensions the performance loss of REMBO is higher. Particularly for $d = 10$ and $D = 50$ the results for REMBO improves only slightly compared to the dataset initialization. Overall the standard deviation is the lowest for $d = 10$ and $D = 50$, except for the DBO algorithm, since it always optimizes the Ackley function in the lower dimensional setting.

4.2 Robotic Air Hockey Task

In this section an air hockey simulation is used to evaluate the performance of the algorithms on a robotic inspired problem. The task is to learn policy parameters which are used to create the trajectory of the mallet. In order to score a goal the puck has to be hit by the mallet, in such a way that the puck reaches the goal area. In Section 4.2.1 the simulation is descried in detail, along with the experimental settings. Section 4.2.2 presents the results of the experiments.

4.2.1 Simulation

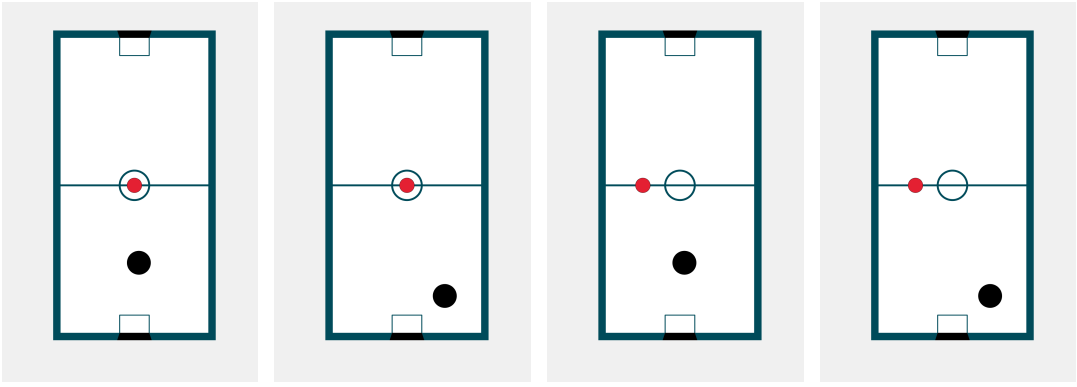
The airhockey simulation consists of an air hockey field with borders, two goals, a mallet and a puck. The measurements of the field, puck and mallet can be seen in Tab. 4.5. The mallet can be moved freely in the lower half of the field. The puck only moves, if it is hit by the mallet. Each rollout of the experiment is limited to 5 seconds. In order to learn a policy, the following cost function is applied

$$c = \sum_{t=1}^{N} (a_m(t) + d_p(t)) - g \cdot 300, \tag{4.1}$$

with N as the number of time steps simulated, $a_m(t)$ as the acceleration of the mallet at time step t and $d_p(t)$ as the distance between the puck and the goal. The value of g is set to 1 if a goal is scored and is 0 if the puck did not reached the goal. Therefor moving the

Table 4.5: The parameters of the air hockey simulation, including sizes, weights and a friction coefficient.

parameter	value
width	1
height	2
goal width	0.2
radius mallet	0.08
radius puck	0.05
weight mallet	0.3
weight puck	0.1
friction coefficient	0.1



(a) Setting 1 (b) Setting 2 (c) Setting 3 (d) Setting 4

Figure 4.5: Visualization of the initial air hockey field for different settings. In setting 1 and 3 the mallet (black) is located at $(0, -0.5)$. For the settings 2 and 4 the mallet starts at $(0.25, -0.75)$. In the settings 1 and 2 the puck (red) is placed in the center $(0, 0)$ and in the settings 3 and 4 the puck is placed off center at $(-0.25, 0)$.

mallet with less acceleration if favored. If the puck moves towards or into the goal the costs of the policy are reduced. Due to the design of the cost function, negative costs for a policy evaluation indicates that a goal is scored. The current position of the mallet $\mathbf{x}(t)$ and the initial position of the puck are known. The initial position of the mallet and the puck remain equal during the optimization process. In order to vary the task, four different settings are defined, where the mallet is initially placed at the point $(0, -0.5)$ or $(0.25, -0.75)$ and the puck starts at $(0, 0)$ or $(-0.25, 0)$. This is visualized in Fig. 4.5.

In a first experiment the policy consists of a P controller, which is used to create the trajectory of the mallet. A P controller uses the error between the current state $\mathbf{x}(t)$ and a target state to compute an action, reducing the error. In this case the P controller is used to control the mallets position. Therefore, two way points $\mathbf{x}_1 = (x_1, y_1)$ and $\mathbf{x}_2 = (x_2, y_2)$ for the mallet and the proportional gain K_P, a parameter for the P controller, are learned.

The correction $\mathbf{u}(t)$ is then computed with

$$\mathbf{u}(t) = K_P \cdot e(t) = K_P \cdot (\mathbf{x}_i - \mathbf{x}(t)),$$

with $\mathbf{x}_i, i \in \{1, 2, 3\}$ as the active way point and $\mathbf{x}_3$ as the initial position of the puck. The new position of the mallet is then updated with

$$\mathbf{x}(t+1) = \mathbf{x}(t) + \mathbf{u}(t).$$

Learning the parameters for the P controller policy is applied to all settings described in Fig. 4.5.

The second policy uses cubic spline curves to generate the trajectory of the mallet. A spline consists of piecewise polynomial curves. The elements of a spline curve are defined by a series of points $x_0, \ldots, x_n$. These points are used to define n intervals $[x_0, x_1], \ldots, [x_{n-1}, x_n]$. At the i-th interval the cubic spline matches a polynomial $P_i(x) = a_i + b_i \cdot x + c_i \cdot x^2 + d_i \cdot x^3$. Therefore, a cubic spline is defined as

$$S(x) = \begin{cases} P_1(x) & , x_0 \leq x < x_1 \\ P_2(x) & , x_1 \leq x < x_2 \\ \vdots & \\ P_n(x) & , x_{n-1} \leq x < x_n. \end{cases}$$

The parameters a_i, b_i, c_i, d_i, with $i = 1, \ldots, n$ are chosen, such that

$$\begin{aligned} P_i(x_i) &= P_{i+1}(x_i), \\ \left.\frac{\partial P_i}{\partial x}\right|_{x=x_i} &= \left.\frac{\partial P_{i+1}}{\partial x}\right|_{x=x_i} \quad \text{and} \\ \left.\frac{\partial^2 P_i}{\partial^2 x}\right|_{x=x_i} &= \left.\frac{\partial^2 P_{i+1}}{\partial^2 x}\right|_{x=x_i} \end{aligned}$$

holds. Therefor a cubic spline is twice continuously differentiable in $[x_0, x_n]$. For the air hockey experiment the policy consists of 10 way points $\mathbf{x}_1, \ldots, \mathbf{x}_{10} \in \mathbb{R}^2$ in the lower half of the game field. Since the initial position of the mallet and the puck are know, a cubic spline curve with $n = 12$ way points is used, where $\mathbf{x}_0$ and $\mathbf{x}_{11}$ are the initial position of the mallet and puck respectively. The trajectory is created by two cubic spline curves, where the path in x and y direction is computed separately. The trajectory is traced within the first two seconds of the simulation. The time is equally distributed over the defined intervals. Therefore, the speed of the mallet is defined by the distance between the current way points. The last three seconds are required to simulate the movement of the puck. The cost function from Eq. (4.1) is modified with a higher reward for scoring a goal, resulting in

$$c = \sum_{t=1}^{N} (a_m(t) + d_p(t)) - g \cdot 500.$$

This is motivated, as the mallet moves faster and due to more way points the acceleration increases.

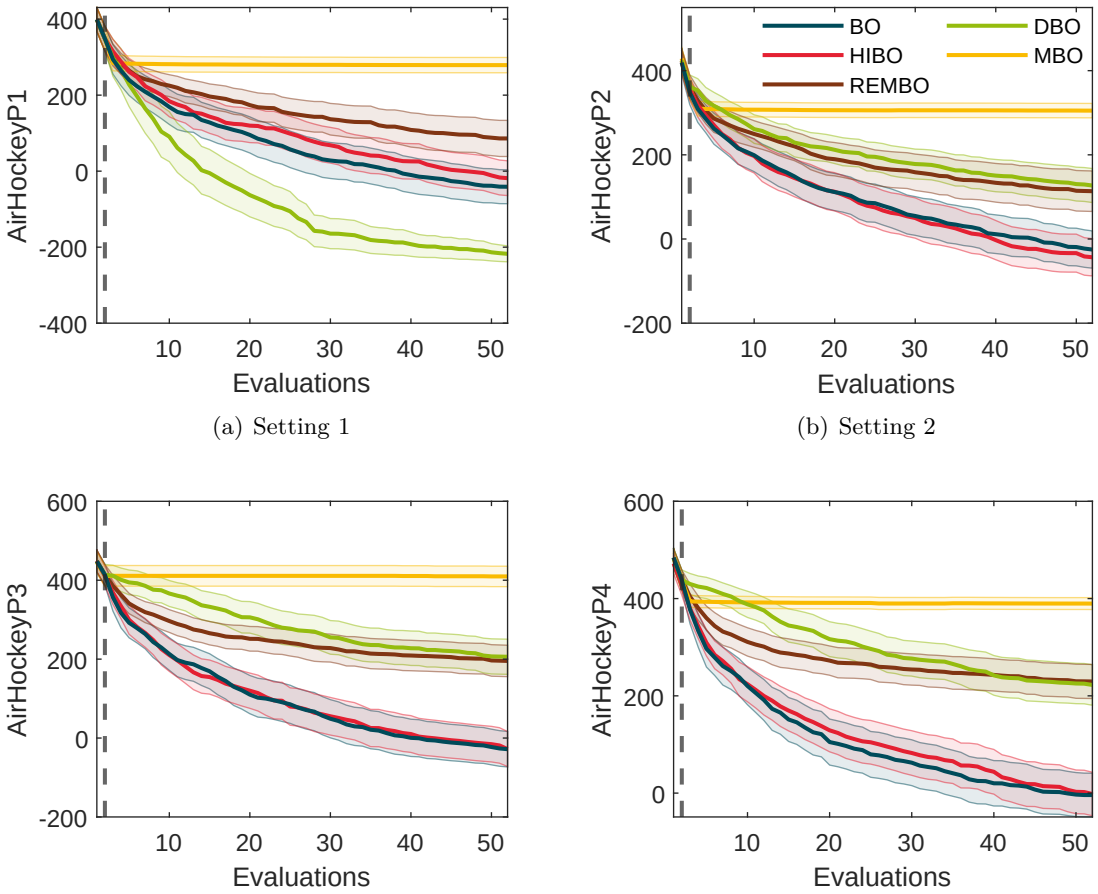
Table 4.6: Mean and standard deviation of the best P controller policy parameters while learning to score a goal in the air hockey simulation. The values are computed based on 200 repetitions.

Algorithm	Setting 1	Setting 2	Setting 3	Setting 4
BO	20.86 ± 130.22	33.72 ± 128.19	32.59 ± 131.62	52.23 ± 130.94
DBO	−120.44 ± 67.67	145.81 ± 124.01	220.43 ± 145.49	234.69 ± 137.05
HIBO	37.33 ± 133.81	21.94 ± 130.17	33.93 ± 129.46	53.38 ± 130.0
MBO	279.58 ± 71.95	304.52 ± 65.32	408.58 ± 102.14	387.74 ± 55.94
REMBO	115.97 ± 145.47	142.36 ± 148.85	208.40 ± 128.32	237.78 ± 114.94

4.2.2 Results

In order to learn the parameters of the policies described above, BO and its extensions execute 50 iterations. The dataset is initialized with 2 initial samples. The experiments include 200 repetitions of the optimization process. Fig. 4.6 shows the results of optimizing the parameters of the P controller policy. The extensions of BO are using a feature space of size $d = 2$. The mean and standard deviation values of the final solutions can be seen in Tab. 4.6. In total, MBO is not able to find proper parameters for the policy. In setting 1, 2 and 4, an improvement is only made in the first iterations. However, within the remaining iterations no improvement is made. In setting 3 no progress is made, compared to the initial dataset. BO and HIBO perform at similar levels. Regarding all settings, both algorithms achieve costs between 20 and 53 in average. In the settings 2, 3 and 4 BO and HIBO receive the lowest cost values, compared to the other algorithms. All this settings differ to setting 1, by the fact that the initial positions of mallet and puck are not directly aligned to the goal. In the first setting DBO minimizes the costs most effectively and also receives negative averaged costs in the end of the optimization, indicating that the task of scoring a goal is solved in most repetitions. However, once the initial position of the mallet and puck do not share the same x-coordinate, like in the remaining settings, the performance of DBO drops. REMBO generates costs between 115 and 238 and is always inferior to BO and HIBO. For setting 2, 3 and 4, REMBO and DBO perform similar, where the averaged costs of REMBO shrink faster in the beginning of the optimization process.

In task oriented robotics, it is mostly not important to find the most optimal parameters. It is sufficient to find a set of parameters that solves the task but is not the optimal solution. Hence, optimizing until a first task solving solution is found, is satisfactory. In the context of the air hockey simulation, a success is defined as scoring a goal. Fig. 4.7 shows the number of evaluations performed by each algorithm until the first success of scoring a goal occurs. In setting 1 DBO is able to find a proper solution within 30 evaluations, for the most cases. In 96.5 % DBO is able to find parameters, leading to a goal. However, in the other settings DBO only yields to working solutions in 13.5 % up to 19.5 % of the repetitions. BO and HIBO are able to find appropriate parameters in 55.5 % up to 65.5 % of the cases. In the first three iterations BO is more likely to gain success, compared to HIBO. However, HIBO catches up in the next iterations, in a way that the performance of HIBO and BO is balanced. In setting 2, 3 and 4 the amount of repetitions, where a proper parameter set is found within 30 evaluations, is distinctly higher for BO and HIBO compared to the other algorithms. REMBO obtains success in only up to 30.5 % of the repetitions. The moment



(a) Setting 1

(b) Setting 2

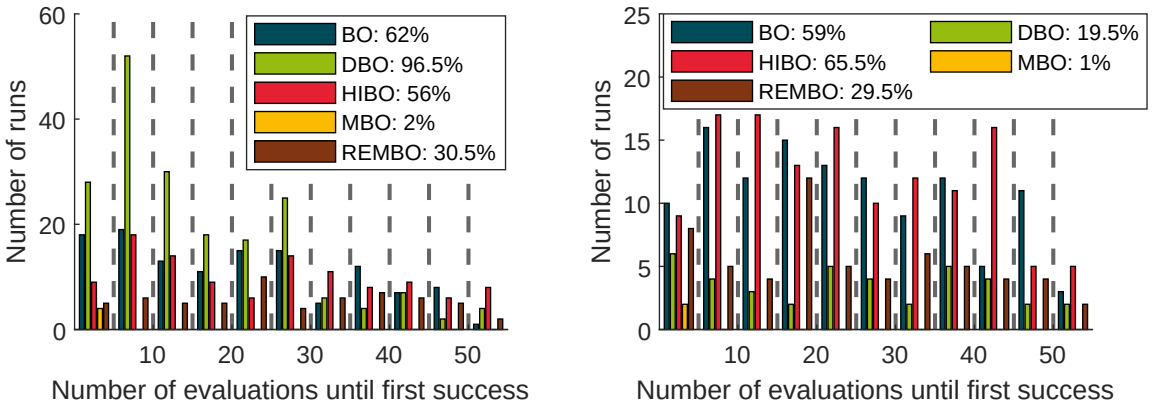
(c) Setting 3

(d) Setting 4

Figure 4.6: (a)-(d) Mean and $\frac{1}{4}$ standard deviation of the cost values over 200 trails, learning the P controller parameters, in order to learn to score a goal according to the initial setting. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process.

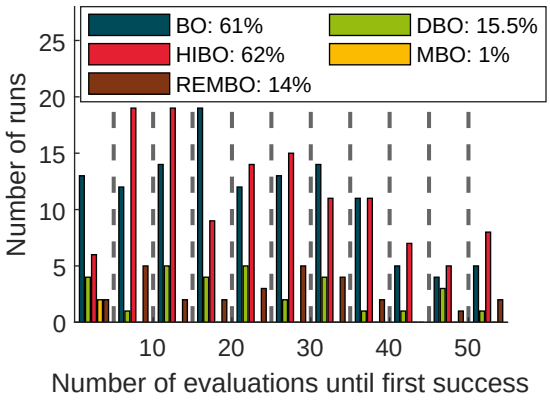
of the first success is mostly equally distributed over the optimization process. The success rate for MBO does not exceed 2 %. Regarding MBO, parameters, leading to a goal, are only found in the first three iterations or are elements of the initial dataset.

Learning the parameters of the spline policy is applied to the first setting, as in Fig. 4.5(a). For DBO, REMBO, MBO and HIBO a feature space of size $d = 5$ is used. For each algorithms 200 repetitions are executed. The results are shown in Fig. 4.8. Overall all algorithms are able to learn parameters which lead to goals. MBO does not perform as well, as the other algorithms, but is able to find proper parameters in 62.5 % of the repetitions. The other algorithms do solve the task in every repetition. BO, REMBO and HIBO achieve similar results, regarding both the averaged mean cost values and the number of evaluations performed until the first success occurs. In most cases task solving parameters are found within the first 20 evaluations. DBO is able to find such parameters within the first 10 eval-

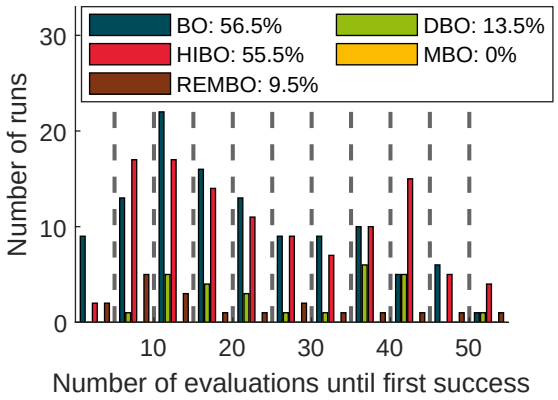


(a) Setting 1

(b) Setting 2

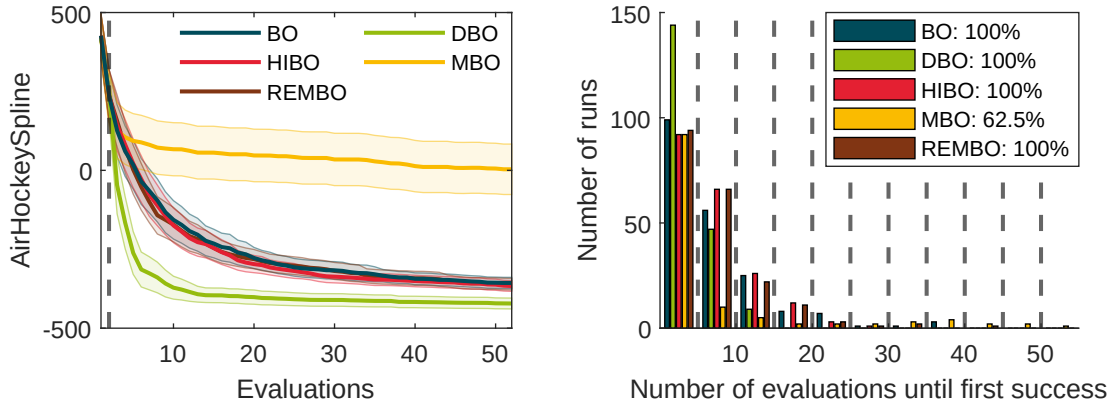


(c) Setting 3



(d) Setting 4

Figure 4.7: (a)-(d) Histograms over the number of evaluations performed on the four different settings until the first success occurs. The histogram uses a bin size of 5, where the first bin consists of the tow initial sample points of the data set and three iterations of the optimization process. The legend shows the percentage of repetitions for each algorithm, where a least one goal is scored. Each algorithm was executed 200 times.



(a) (b)

Figure 4.8: (a) Mean and $\frac{1}{4}$ standard deviation of the cost values over 200 trails learning the spline policy parameters. The vertical line indicates the separation of the data set initialization and the optimization process. The averaged cost values for the best solution are: BO: -354.92 ± 74.95, DBO: -419.73 ± 74.76, HIBO: -364.49 ± 72.79, MBO: 3.37 ± 318.83 and REMBO: -358.87 ± 77.56. (b) Histogram over the number of evaluations until the first success occurs, with a bin size of 5. The legend shows the percentage of repetitions for each algorithm, where a least one goal is scored. Each algorithms was executed 200 times.

uations for the most repetitions. Also the averaged cost values are lower compared to the other algorithms. Generally, the results on learning the spline policy parameters, seems to be better compared to learning the P controller policy parameters. With the spline policy, the mallet does moved faster, since the ten way points have to be approached within the first two seconds of the simulation. Additionally, for the P controller policy the speed of the mallet depends on the proportional gain K_p. Higher speed allows the puck to travel more on the air field, after being hit by the mallet. This is creases the chance that a goal is scored, since the goal is not defended.

5

Discussion and Future Work

This sections discusses the introduced algorithms, the corresponding experimental results and the automatic feature generation of HIBO. Additionally, suggestions for future works are given. In Section 4.1 and Section 4.2 BO and the four high dimensional extensions DBO, HIBO, MBO and REMBO were applied to benchmark functions and an air hockey simulation. Regarding all experiments, there is no algorithm that did perform best in every setting. For each algorithm, except for MBO, there is at least on setting where the algorithm performed best in higher dimensions. BO and HIBO produced the best result for the Branin function. However, the embedded Branin function is restricted to two dimensions. Therefor the experiments on the remaining benchmark functions can be considered to be more difficult. DBO outperformed the other algorithms at the Ackley function, while REMBO optimizes the Schwefel function at best. This indicates, that choosing the best algorithm depends on the given problem.

HIBO

It is noticeable, that HIBO and BO performed similar in every experiment. In some cases a small variation among both algorithms occurs, like in 4.6(b). However, in total the results do not differ significantly. This indicates that the HIBO algorithm does not provide any improvement compared to the classic BO algorithm. One reason can be, that the conditioning of the search space performed by HIBO is not influential enough, in order to guide the optimization of the parameters towards better solutions. Further more, the run time of HIBO is larger since two GPs have to be trained. For example, in the high dimensional benchmark experiments, HIBO lasts about eight times longer compared to BO. Regarding the similar performance of HIBO and BO, combined with the longer run time of HIBO, the proposed version of HIBO is not suitable to replace BO in higher dimension. As mentioned in Chapter 1, HIBO was successfully tested before, with a handcrafted feature generation and additional modifications of the BO algorithm. The experiments within this thesis indicate, that the good performance is derived from the additional modifications, as in this thesis HIBO always performed similar to BO.

As described in Section 3.4, HIBO computes a currently optimal feature. This feature is then used to condition the search space while deploying an acquisition function on the fusion of the parameter space and the feature space. The underlying model has to estimate the objective function on $D + d$ dimensions, where D and d are the size of the parameter

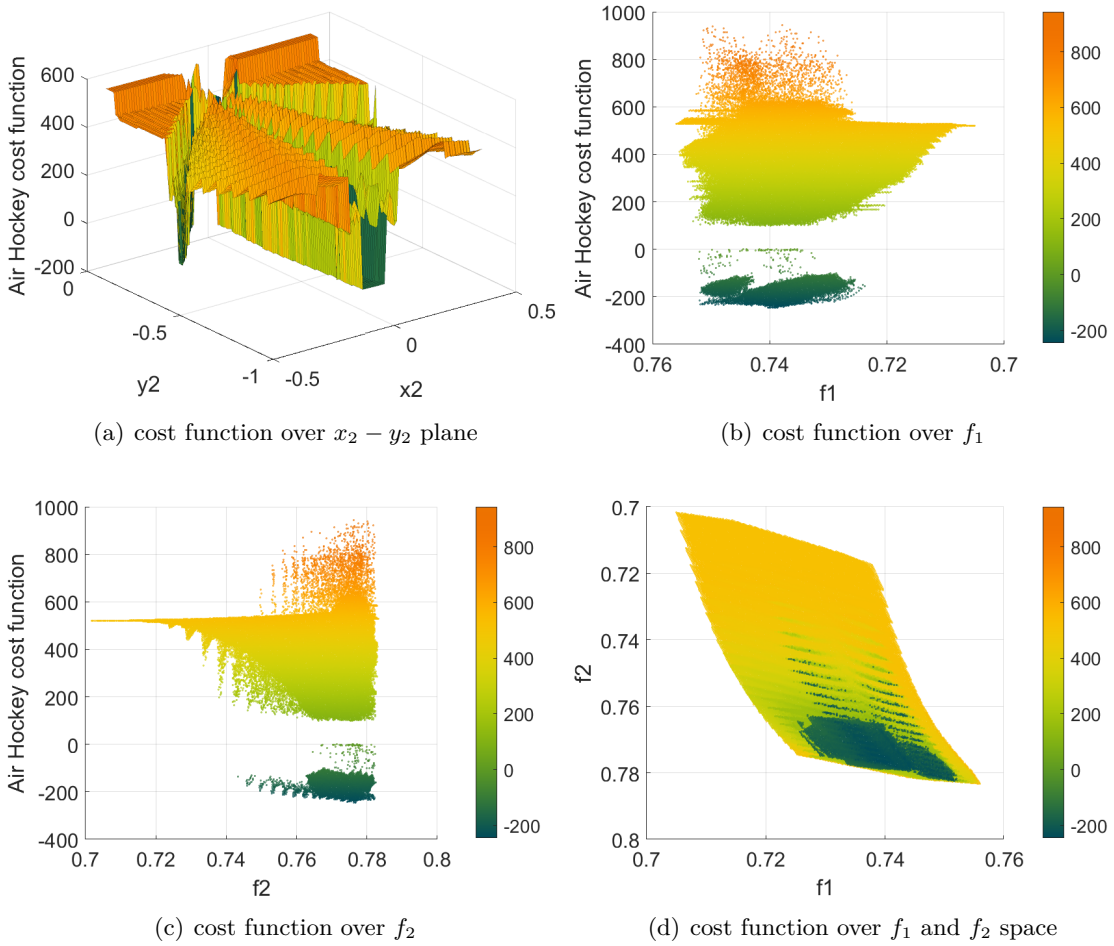
space and the feature space respectively, with $d < D$. This can increase the complexity, compared to the modeling in BO with only D dimensions. However, the introduction of features can provide better information about the objective function. In order to increase the conditioning in HIBO, the kernel matrix of the used GP can be manipulated. Therefor the kernel function is only used to compute the correlation among parameters and features. The remaining elements on the kernel matrix remain at zero. Thus, the parameters only depend on the features. An additional drawback of HIBO compared to the other algorithms is that the second acquisition function has D free parameters to optimize. Except for BO, the other algorithms only optimize on d dimension, whereby the complexity of the model can be reduced.

Feature Generation

Fig. 5.1 shows an exemplary feature space, created by HIBO during the experiment of learning the P controller policy parameters in the first setting. Additionally, the cost function in dependency of the last way point is shown in Fig. 5.1(a). The influence of the last way point is clearly visible, since the puck has to move straight towards the goal, or bounce the wall in certain angle to reach the goal. The plots of the feature space shows, that features, corresponding to task solving parameters, are located in a certain area. However, the cost function in this area is very rugged, as this area contains solutions with very different cost values. Even some of the worst solutions are located within this area. This can corrupt the prediction of a GP Regression and the performance of the optimization process. Therefore, the feature generation can be improved. Instead of learning the feature generation jointly with the GP parameters on the log marginal likelihood from Eq. (2.7), other functions can be considered. An example is the minimization of the third derivative, called jerk, resulting in smoother surfaces. Another potential reason is the number of iterations performed in the optimization process. In this experiment HIBO did compute 52 evaluations of the air hockey simulation. Increasing the number of evaluations could improve the generation of the feature space. However, the number of evaluations should not be to high, since the objective function is connected with high evaluations costs. In the experiments, the feature generation of HIBO is implemented by an ANN with the log-sigmoid function from Eq. (2.9) as the activation function. Thus, the feature values are located in the range of $[0, 1]$. However, the projection of the parameter space into the feature space, does not always covers the full range, as seen in Fig. 5.1. In the current implementation of HIBO, the first acquisition function, applied on the feature space, searches on the full range of $[0, 1]^d$. Thus, the optimization of this acquisition function, can lead to a feature, that does not correspond to the features generated by projecting previous evaluated parameters. As a consequence, the search space of the second acquisition function is restricted to an area, where no training data is available, since only the projection of the parameter space can be used to train the GP. By restricting the search space of the first acquisition function to the projected parameter space, or rescaling the projection to a fixed range, like $[0, 1]^d$, the performance of HIBO could be improved, potentially. The suggested modifications of HIBO, can be examined in further work.

MBO

In most experiments MBO performed poor. Only in the experiment optimizing the Ackley function MBO gained improvement and outperforms BO, HIBO and REMBO. This does not match the results from [34], where MBO performed superior to REMBO and other



(a) cost function over $x_2 - y_2$ plane

(b) cost function over f_1

(c) cost function over f_2

(d) cost function over f_1 and f_2 space

Figure 5.1: Comparison of the cost function in setting 1 over the parameter space and the feature space with $d = 2$, learned by HIBO. (a) The cost function plotted over the $x_2 - y_2$ plane, for fixed $K_P = 9$, $x_1 = 0.25$ and $y_1 = -0.4$. (b) - (d) The cost function over the feature space, where f_1 and f_2 are the learned features. To generate the features, a grid in the parameter space, with 20 points per dimension is mapped into a feature space. The values of the cost function are received by evaluating the corresponding parameters.

algorithms like [17]. However, [34] used the Michalewicz function and Rosenbrock function as a benchmark. Additionally, in [34] the optimization process lasts 500 iterations. In this thesis, a maximum of 50 iterations were performed. Hence, the result from this thesis and from [34] are not directly comparable. However, the performance differences may derive from different implementations and different approaches to learn the hyper parameters of MBO. Optimizing with MBO leads to high running times. The most expensive operation is the inversion of the kernel matrix, while mapping the feature space to the parameter space with a Multi Output GP. As the kernel matrix is build with the Kronecker product, the matrix has a size of $ND \times ND$, with N as the number of samples and D as the dimensionality of the output. Thus, with higher dimensions and a advancing optimization process, the running time of an iteration significantly increases. This is the reason why MBO is not considered in the high dimensional benchmark experiments, where the running time of the total optimization process over 30 iterations would last about 6 hours.

REMBO

For REMBO it is assumed that the objective function has a lower dimensional effective dimensionality. Hence, the experimental setup on the benchmark functions is in favor of REMBO, since low dimensional versions of the benchmark functions are embedded in higher dimensions. However, REMBO yielded divers results. At the Schwefel function, REMBO was able to handle higher dimensions better than the other algorithms. Otherwise, REMBO was not able to gain improvement at the high dimensional experiment on the Ackley function. At the air hockey experiments, REMBO performed ordinary, but was not able to score goals in most cases. REMBO creates a random embedding by using a randomly created projection matrix. Since the embedding and the projection matrix are constant over the optimization process, no adaption to the correlation of the parameters is possible. Therefor REMBO only can learn the correlation between the embedding and the objective function. Additionally, REMBO is based on a linear transformation. Thus, the correlation of the embedding and the parameter space is assumed to be linear.

DBO

DBO optimizes directly on the parameter space. However, only a randomly chosen subset of parameters is considered in an iteration. Thus, DBO can consider the correlation of the parameters, if those are chosen, but has to operate on a low number of dimensions. In the experiments DBO performed well. Except for the Schwefel function and some settings in the air hockey simulation, DBO was able to yield good results, compared to the other algorithms. DBO seems to perform well, if the parameters are not highly correlated or have steady influence, regarding the objective function. An example is the Ackley function, where optimizing a single dimension leads to a general improvement. In the case of the air hockey simulation, the first setting allows a simple solution, by directly moving towards the puck, since mallet and puck are directly aligned towards the goal. As long as the last way point is located between the goals, the y value can be changed with only small impact to the performance. This simplifies the optimization. In the other settings the x and y values of the last way point are more correlated. Moving the last checkpoint more towards the left or right, requires to move also in the y direction, since only a certain angle leads to goals, which is visible in Fig. 5.1(a). As in each iteration a number of dimensions are randomly chosen, such correlations are not always considered. Additionally, DBO is not able to adapt to the objective function, since a uniform distribution is used

to choose the dimensions. In the setup for the benchmark experiments, there are a lot dimension with zero impact. If DBO is able to detect dimensions with less impact, DBO could decrease the probability for choosing this dimension. In [48], PCA and eigenvalues are used to compute a probability for each dimension to be chosen. However, PCA requires an appropriate amount of training data to perform well. Another possible approach to compute the probability, is to measure the amount of improvement made by changing the value of the dimension. If an improvement is made within an iteration, the probability of being chosen is increased for all chosen dimensions. Over time favorable dimension can be detected. For evolutionary algorithms, there exits different approaches to compute probabilities based on the performance of a populations individuals [19]. Those can be adapted to DBO, by using the amount of improvement as a performance measurement, when choosing the dimensions.

Experimental Setup

The experimental setup for the benchmark functions, described in Section 4.1, embeds a lower d-dimensional benchmark function into a higher D-dimensional space, with $d < D$. However, only d dimensions are used to evaluate the benchmark function. The remaining dimensions have no impact onto the results and can not be optimized. Thus, the experimental setup does not provide realistic high dimensional optimization, since it is rare that a parameter does not have any impact on the performance. Hence, additional experiments with BO, DBO, HIBO and REMBO were executed on the Ackley and Schwefel function, where a D-dimensional version of the function is optimized, with $D \in \{10, 30, 50\}$. Since the Branin function is restricted to two dimensions, this experiment is not performed for the Branin function. The results are shown in Fig. A.1 and Tab. A.1 for the Schwefel function and in Fig. A.2 and Tab. A.2 for the Ackley function. Generally the value of the best found solution is inferior, compared to the results from Section 4.1. This is related to the higher number of dimensions that have to be optimized. However, the relation among the algorithms performance remains the same. The Schwefel function is optimized best by REMBO, and DBO achieves the best results on the Ackley function. Furthermore the performance of BO and HIBO is similar. Hence, the conclusions based on the results in Section 4.1 also hold for high dimensional optimization, with a high number of parameters being influential.

6

Summary

This thesis introduced and examined the HIBO algorithm with automatic feature learning. The motivation is to propose an extension for BO, which is able to perform with a higher number of parameters to be optimized. Therefor HIBO uses a mGP to learn a feature generation, which is implemented with an ANN. An acquisition function computes a feature, optimizing the current model of the objective function. By fusing the parameter and feature space, and constraining the search space to the previous computed feature, a new set of parameters is received for evaluation. The HIBO algorithm is compared to three existing extensions of BO for higher dimensions. This includes DBO, which randomly selects a subset of parameters to be optimized for each iteration. REMBO performs the optimization of the objective function in a random embedding, and maps the embedding onto the parameter space, with a linear transformation. The third algorithm is MBO, which uses a mGP to compute a feature space and an optimal feature, regarding the objective function. This feature is then mapped back into the parameter space with a Multi Output GP. Three benchmark functions and an air hockey simulation are used, to compare the performance of the algorithms. Overall, no algorithm was able to perform best in every experiment. Hence, finding the best algorithm in the context of BO depends on the problem that has to be solved. HIBO performed similar to BO in all experiments. Thus, HIBO in its proposed version, does not improve the BO algorithm in low dimensional settings as well as in high dimensional settings. The chosen approach to condition the parameter space with the features, is not able to guide the optimization process towards better solutions. Additional modifications for HIBO are suggested, which can be evaluated in further work. DBO and REMBO performed well in most experiments. However, depending on the given problem, both algorithms did not always perform superior to BO. DBO is not able to adapt to the given problem, since DBO always randomly samples dimensions with a uniform distribution. For DBO a modification is suggested, which can be considered in further research. All in all, the proposed HIBO algorithm is not able to adapt BO to high dimensions. However DBO and REMBO yield better solutions than BO, depending on the given problem.

Bibliography

[1] Abramowitz, M. *Handbook of Mathematical Functions, With Formulas, Graphs, and Mathematical Tables,* New York, NY, USA: Dover Publications, Inc., 1974. ISBN: 0486612724.

[2] Agarwalla, A., Jain, A. K., Manohar, K., Saxena, A., and Mukhopadhyay, J. Bayesian Optimisation with Prior Reuse for Motion Planning in Robot Soccer. In: *arXiv e-prints*.arXiv:1611.01851:arXiv:1611.01851, 2016. arXiv: 1611.01851 [cs.RO].

[3] Bonilla, E. V., Chai, K. M., and Williams, C. Multi-task Gaussian Process Prediction. In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis. Curran Associates, Inc., 2008, pp. 153–160. URL: http://papers.nips.cc/paper/3189-multi-task-gaussian-process-prediction.pdf.

[4] Brochu, E., Cora, V. M., and de Freitas, N. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. In: *arXiv e-prints*.arXiv:1012.2599:arXiv:1012.2599, 2010. arXiv: 1012.2599 [cs.LG].

[5] Brooks, R. A. Artificial Life and Real Robots. In: *Proceedings of the First European Conference on Artificial Life*. MIT Press, 1992, pp. 3–10.

[6] Broyden, C. G. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. In: *IMA Journal of Applied Mathematics* 6(1):76–90, Mar. 1970. ISSN: 0272-4960. DOI: 10.1093/imamat/6.1.76. eprint: http://oup.prod.sis.lan/imamat/article-pdf/6/1/76/2233756/6-1-76.pdf. URL: https://doi.org/10.1093/imamat/6.1.76.

[7] Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold Gaussian Processes for Regression. In: *arXiv e-prints*.arXiv:1402.5876:arXiv:1402.5876, 2014. arXiv: 1402.5876 [stat.ML].

[8] Calandra, R., Seyfarth, A., Peters, J., and Deisenroth, M. P. Bayesian optimization for learning gaits under uncertainty. In: *Annals of Mathematics and Artificial Intelligence* 76(1):5–23, 2016. ISSN: 1573-7470. DOI: 10.1007/s10472-015-9463-9. URL: https://doi.org/10.1007/s10472-015-9463-9.

[9] Cox, D. D. and John, S. A statistical method for global optimization. In: *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*. 1992, 1241–1246 vol.2. DOI: 10.1109/ICSMC.1992.271617.

[10] *File:Derived Neuron schema with no labels.svg*. Accessed: Sep 23 2019. URL: https://commons.wikimedia.org/wiki/File:Derived_Neuron_schema_with_no_labels.svg.

[11] Fletcher, R. A new approach to variable metric algorithms. In: *The Computer Journal* 13(3):317–322, Jan. 1970. ISSN: 0010-4620. DOI: 10.1093/comjnl/13.3.317. eprint: http://oup.prod.sis.lan/comjnl/article-pdf/13/3/317/988678/130317.pdf. URL: https://doi.org/10.1093/comjnl/13.3.317.

[12] Frazier, P. I. A Tutorial on Bayesian Optimization. In: *CoRR* abs/1807.02811, 2018.

[13] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. In: *Biological Cybernetics* 36(4):193–202, 1980. ISSN: 1432-0770. DOI: 10.1007/BF00344251. URL: https://doi.org/10.1007/BF00344251.

[14] Garnett, R., Osborne, M. A., and Roberts, S. J. Bayesian optimization for sensor set selection. In: *IPSN*. 2010.

[15] Goldfarv, D. A family of variable-metric methods derived by variational means. In: *Math. Comp.* 24(109):23–26, 1970.

[16] Goovaerts, P. *Geostatistics for natural resources evaluation*. Oxford: Oxford University Press, 1997.

[17] Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. In: *ACS Central Science* 4(2):268–276, 2018. PMID: 29532027. DOI: 10.1021/acscentsci.7b00572. eprint: https://doi.org/10.1021/acscentsci.7b00572. URL: https://doi.org/10.1021/acscentsci.7b00572.

[18] Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., et al. Deep Speech: Scaling up end-to-end speech recognition. In: *arXiv e-prints*.arXiv:1412.5567:arXiv:1412.5567, 2014. arXiv: 1412.5567 [cs.CL].

[19] Hassani, A. and Treijs, J. An overview of standard and parallel genetic algorithms. In: *IDT Workshop on Interesting Results in Computer Science and Engineering, Mälardalen University*. 2009, pp. 1–7.

[20] Hickish, B., Fletcher, D. I., and Harrison, R. F. Investigating Bayesian Optimization for rail network optimization. In: *International Journal of Rail Transportation* 0(0):1–17, 2019. DOI: 10.1080/23248378.2019.1669500. eprint: https://doi.org/10.1080/23248378.2019.1669500. URL: https://doi.org/10.1080/23248378.2019.1669500.

[21] Jamil, M. and Yang, X. S. A literature survey of benchmark functions for global optimisation problems. In: *International Journal of Mathematical Modelling and Numerical Optimisation* 4(2):150, 2013. ISSN: 2040-3615. DOI: 10.1504/ijmmno.2013.055204. URL: http://dx.doi.org/10.1504/IJMMNO.2013.055204.

[22] Kandasamy, K., Schneider, J., and Poczos, B. High Dimensional Bayesian Optimisation and Bandits via Additive Models. In: *arXiv e-prints*.arXiv:1503.01673:arXiv:1503.01673, 2015. arXiv: 1503.01673 [stat.ML].

[23] Kaneko, K., Kaminaga, H., Sakaguchi, T., Kajita, S., Morisawa, M., Kumagai, I., and Kanehiro, F. Humanoid Robot HRP-5P: An Electrically Actuated Humanoid Robot With High-Power and Wide-Range Joints. In: *IEEE Robotics and Automation Letters* 4:1431–1438, 2019.

[24] Koos, S., Mouret, J.-B., and Doncieux, S. The Transferability Approach: Crossing the Reality Gap in Evolutionary Robotics. In: *IEEE Transactions on Evolutionary Computation* 17:122–145, 2013.

[25] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Commun. ACM* 60(6):84–90, May 2017. ISSN: 0001-0782. DOI: 10.1145/3065386. URL: http://doi.acm.org/10.1145/3065386.

[26] Kushner, H. J. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. In: *Journal of Basic Engineering* 86(1):97, 1964. DOI: 10.1115/1.3653121. URL: https://doi.org/10.1115%2F1.3653121.

[27] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE* 86(11):2278–2324, 1998. DOI: 10.1109/5.726791.

[28] Lehman, J., Clune, J., Misevic, D., Adami, C., Altenberg, L., Beaulieu, J., Bentley, P. J., Bernard, S., Beslon, G., Bryson, D. M., et al. The Surprising Creativity of Digital Evolution: A Collection of Anecdotes from the Evolutionary Computation and Artificial Life Research Communities. In: *arXiv e-prints*.arXiv:1803.03453:arXiv:1803.03453, 2018. arXiv: 1803.03453 [cs.NE].

[29] Li, C., Gupta, S., Rana, S., Nguyen, V., Venkatesh, S., and Shilton, A. High Dimensional Bayesian Optimization Using Dropout. In: *arXiv e-prints*.arXiv:1802.05400:arXiv:1802.05400, 2018. arXiv: 1802.05400 [stat.ML].

[30] Lizotte, D., Wang, T., Bowling, M., and Schuurmans, D. Automatic Gait Optimization with Gaussian Process Regression. In: *Proceedings of the 20th International Joint Conference on Artifical Intelligence*. IJCAI'07. Hyderabad, India: Morgan Kaufmann Publishers Inc., 2007, pp. 944–949. URL: http://dl.acm.org/citation.cfm?id=1625275.1625428.

[31] Marchant, R. and Ramos, F. Bayesian optimisation for Intelligent Environmental Monitoring. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 2242–2249. DOI: 10.1109/IROS.2012.6385653.

[32] Marco, A., Hennig, P., Bohg, J., Schaal, S., and Trimpe, S. Automatic LQR tuning based on Gaussian process global optimization. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 270–277. DOI: 10.1109/ICRA.2016.7487144.

[33] McCulloch, W. S. and Pitts, W. A logical calculus of the ideas immanent in nervous activity. In: *The bulletin of mathematical biophysics* 5(4):115–133, 1943. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259.

[34] Moriconi, R., Deisenroth, M. P., and Sesh Kumar, K. S. High-dimensional Bayesian optimization using low-dimensional feature spaces. In: *arXiv e-prints*.arXiv:1902.10675:arXiv:1902.10675, 2019. arXiv: 1902.10675 [stat.ML].

[35] Močkus, J., Tiesis, V., and Žilinskas, A. The application of Bayesian methods for seeking the extremum. In: vol. 2. Elsevier, Sept. 1978, pp. 117–129. ISBN: 0-444-85171-2.

[36] Nolfi, S. and Floreano, D. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. Scituate, MA, USA: Bradford Company, 2004. ISBN: 0262640562.

[37] Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. Cambridge, MA, USA: MIT Press, Jan. 2006, p. 248.

[38] Ruehl, S. W., Parlitz, C., Heppner, G., Hermann, A., Roennau, A., and Dillmann, R. Experimental evaluation of the schunk 5-Finger gripping hand for grasping tasks. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*. 2014, pp. 2465–2470. DOI: 10.1109/ROBIO.2014.7090710.

[39] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. In: *Nature* 323:533–536, 1986.

[40] Sano, S., Kadowaki, T., Tsuda, K., and Kimura, S. Application of Bayesian Optimization for Pharmaceutical Product Development. In: *Journal of Pharmaceutical Innovation*, 2019. ISSN: 1939-8042. DOI: 10.1007/s12247-019-09382-8. URL: https://doi.org/10.1007/s12247-019-09382-8.

[41] *Schrunk SVH*. Accessed at Nov 1 2019. URL: https://schunk.com/de_en/gripping-systems/series/svh/.

[42] Sewak, M. *Deep Reinforcement Learning: Frontiers of Artificial Intelligence*. Jan. 2019. ISBN: 978-981-13-8284-0. DOI: 10.1007/978-981-13-8285-7.

[43] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. Taking the Human Out of the Loop: A Review of Bayesian Optimization. In: *Proceedings of the IEEE* 104(1):148–175, 2016. ISSN: 0018-9219. DOI: 10.1109/JPROC.2015.2494218.

[44] Shanno, D. F. Conditioning of quasi-Newton methods for function minimization. In: *Mathematics of computation* 24(111):647–656, 1970.

[45] Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. Gaussian Process Bandits without Regret: An Experimental Design Approach. In: *CoRR* abs/0912.3995, 2009. arXiv: 0912.3995. URL: http://arxiv.org/abs/0912.3995.

[46] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *Journal of Machine Learning Research* 15:1929–1958, 2014. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[47] Tremblay, J., To, T., Sundaralingam, B., Xiang, Y., Fox, D., and Birchfield, S. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. In: *CoRR* abs/1809.10790, 2018. arXiv: 1809.10790. URL: http://arxiv.org/abs/1809.10790.

[48] Ulmasov, D., Baroukh, C., Chachuat, B., Deisenroth, M. P., and Misener, R. Bayesian Optimization with Dimension Scheduling: Application to Biological Systems. In: *arXiv e-prints*.arXiv:1511.05385:arXiv:1511.05385, 2015. arXiv: 1511.05385 [stat.ML].

[49] Wackernagel, H. *Multivariate geostatistics: an introduction with applications*. English. Vol. 33. 8. 1996, 363A.

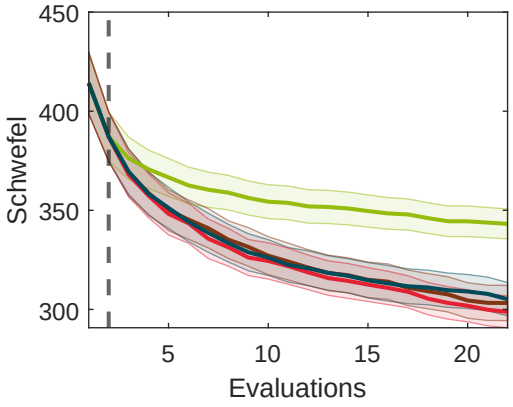
[50] Wang, Z., Hutter, F., Zoghi, M., Matheson, D., and de Freitas, N. Bayesian Optimization in a Billion Dimensions via Random Embeddings. In: *arXiv e-prints*.arXiv:1301.1942:arXiv:1301.1942, 2013. arXiv: 1301.1942 [stat.ML].

[51] Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and De Freitas, N. Bayesian Optimization in High Dimensions via Random Embeddings. In: *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*. IJCAI '13. Beijing, China: AAAI Press, 2013, pp. 1778–1784. ISBN: 978-1-57735-633-2. URL: http://dl.acm.org/citation.cfm?id=2540128.2540383.

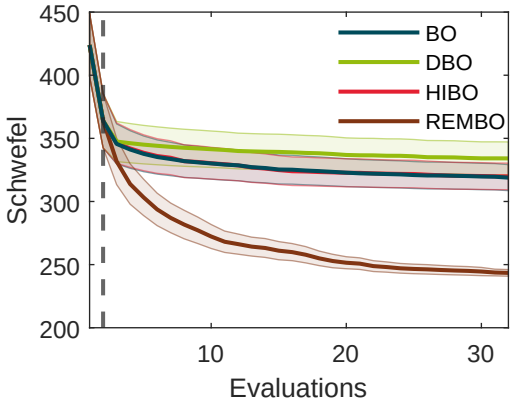
[52] Zhang, K., Zhang, Z., Li, Z., and Qiao, Y. Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks. In: *IEEE Signal Processing Letters* 23(10):1499–1503, 2016. DOI: 10.1109/LSP.2016.2603342. arXiv: 1604.02878 [cs.CV].

A

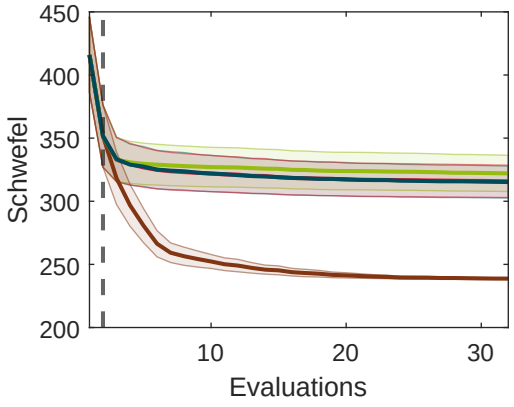
Appendix



(a) $d = 2$ und $D = 10$



(b) $d = 5$ und $D = 30$

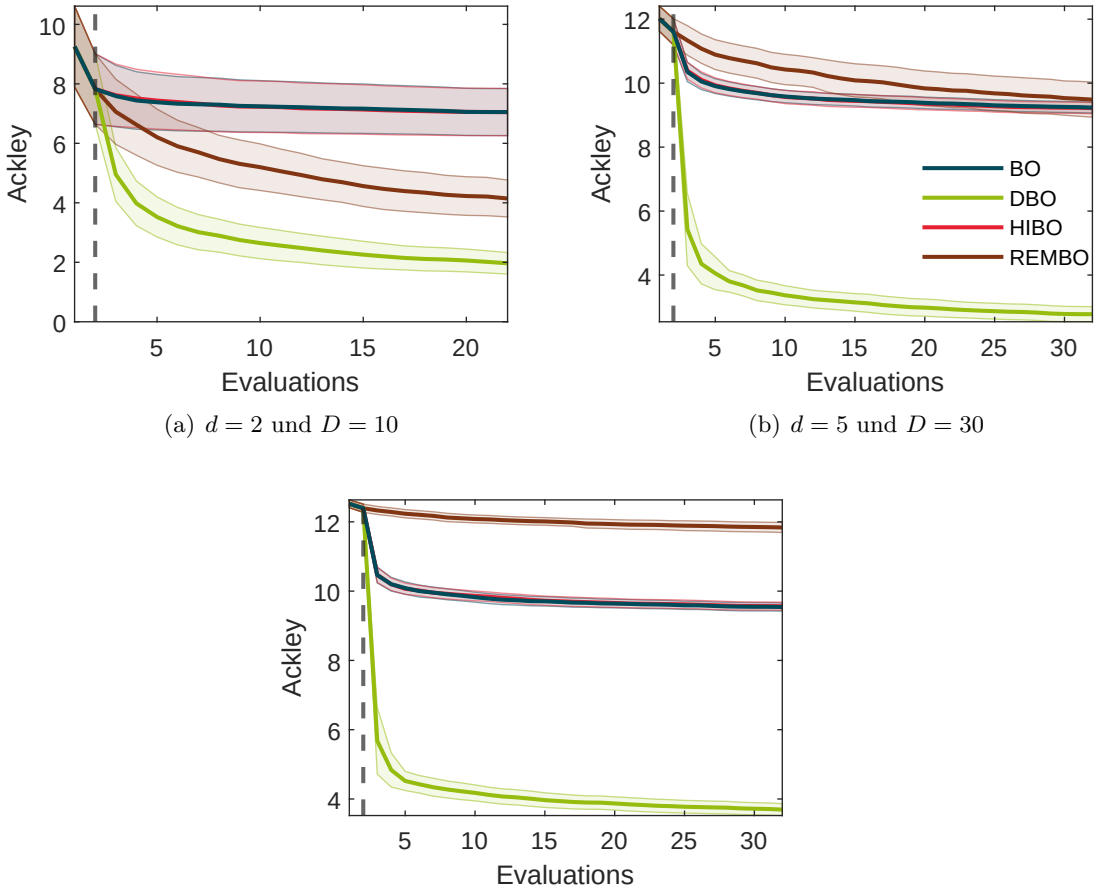


(c) $d = 10$ und $D = 50$

Figure A.1: (a)-(c) Mean and $\frac{1}{4}$ standard deviation over 200 trails, optimizing the D-dimensional Schwefel function. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process. DBO, HIBO and REMBO using a d-dimensional space.

Table A.1: Mean and standard deviation of the best parameter value on the D-dimensional Schwefel function function, found during the optimization process. DBO, HIBO and REMBO using a d-dimensional space. The values are computed based on 200 repetitions.

Algorithm	$d = 2$ $D = 10$	$d = 5$ $D = 30$	$d = 10$ $D = 50$
BO	303.63 ± 39.92	317.30 ± 46.85	313.66 ± 55.3
DBO	341.45 ± 38.65	332.46 ± 57.20	320.45 ± 61.68
HIBO	297.29 ± 38.34	318.27 ± 47.68	314.25 ± 55.78
REMBO	301.73 ± 41.45	242.23 ± 20.15	237.56 ± 17.07



(a) $d = 2$ und $D = 10$

(b) $d = 5$ und $D = 30$

(c) $d = 10$ und $D = 50$

Figure A.2: (a)-(c) Mean and $\frac{1}{2}$ standard deviation over 200 trails, optimizing the D-dimensional Ackley function. Thereby the best solution found so far is considered. The vertical line indicates the separation of the data set initialization and the optimization process. DBO, HIBO and REMBO using a d-dimensional space.

Table A.2: Mean and standard deviation of the best parameter value on the D-dimensional Ackley function function, found during the optimization process. DBO, HIBO and REMBO using a d-dimensional space. The values are computed based on 200 repetitions.

Algorithm	$d = 2$ $D = 10$	$d = 5$ $D = 30$	$d = 10$ $D = 50$
BO	7.02 ± 1.65	9.19 ± 0.74	9.49 ± 0.71
DBO	1.96 ± 0.74	2.76 ± 0.51	3.67 ± 0.43
HIBO	7.01 ± 1.66	9.17 ± 0.74	9.52 ± 0.71
REMBO	4.12 ± 1.27	9.44 ± 1.28	11.78 ± 0.88