University of Zurich[UZH]

Bayesian Optimization with Uncertainty Bounds for Neural Networks

Master's Thesis
February 3, 2021

Marius Högger
of Zurich, ZH, Switzerland

Student-ID: 13-920-053
marius.hoegger@uzh.ch

Advisor: **Jakob Weissteiner**

Prof. Dr. Sven Seuken
Department of Informatics
University of Zurich
http://www.ifi.uzh.ch/ce

Acknowledgements

First I would like to thank my advisor, Jakob Weissteiner from the Computation and Economics research Group at the University of Zurich for guiding me through my Master's Thesis. Whenever I had questions about my research or needed some guidance for the writing I received valuable answers and feedback from you.

I would also like to thank Hanna Sophia Wutte and Jakob Heiss from the Department of Mathematics at the ETH Zurich for the countless virtual meeting together with Jakob Weissteiner and me where we had very constructive discussions.

My gratitude goes to Prof. Dr. Sven Seuken for giving me the opportunity to research such an interesting and thriving topic at his Computation and Economics Research Group.

I would also like thank the University of Zurich for providing such a encouraging learn and research environment as well as the required computational infrastructure.

Last, but not least I like to acknowledge Joel Barmettler and Douglas Da Silva as second readers of this thesis and thank them for their valuable comments on this work.

Abstract

For most state-of-the-art estimator methods such as Gaussian processes, high dimensional Bayesian optimization is still a very challenging problem. This thesis studies the application of neural networks (NNs) as estimators in Bayesian optimization and assesses if and how they can overcome this curse of dimensionality. Since quantifying the uncertainty of predictions is key in Bayesian optimization, several methods of representing predictive uncertainty for NNs are considered.
Specifically, this thesis compares a very recent approach introduced by Heiss et al. (2021) called *neural optimization-based model uncertainty (NOMU)*, against *Deep Ensembles* and *MC Dropout*, two more established NN-based methods, and Gaussian processes. Moreover, to increase the performance and robustness of the NN-based estimators, several adjustments in the estimation and optimization algorithms are presented.
Experimental evaluations on synthetic data confirm that Gaussian processes outperform the NN-based methods in low dimensional settings (1D-2D) and show that NOMU performs as good or better than the other NN-based methods. However, the experiments in higher dimensions suggest that the NN-based methods improve and finally manage to outperform Gaussian processes with standard configurations. Furthermore, the results indicate that especially in higher dimensions, NOMU performs robustly as good or better than the other NN-based methods.

Zusammenfassung

Für die meisten bereits etablierten Moddelierungs-Methoden wie der Gaussischen Prozess ist die hoch dimensionale Bayessche Optimierung noch immer eine Herausforderung. Diese Arbeit studiert die Anwendung von neuronalen Netzwerken (NN) als Modellierungs-Methode für Bayessche Optimierung und untersucht ob und wie diese den Fluch der Dimensionalität überwinden können. Da die Quantifizierung der Unsicherheit einer Schätzung eine Kernrolle in der Bayessche Optimierung spielt, liegt der Fokus auf NNs welche in der Lage sind ihre eigene Modell-Unsicherheit zu schätzen.
Genauer vergleicht diese Arbeit den sehr neuen Ansatz names *neural optimization-based model uncertainty (NOMU)*, eingeführt von Heiss et al. (2021), mit den etablierteren NN-basierten Methoden *Deep Ensembles* und *MC Dropout* sowie dem Gaussischen Prozess. Des Weiteren werden mehrer Anpassungen an den Moddelierungs und Optimierungs Algorithmen vorgestellt um die Performanz und Robustheit der NN-basierten Modellierungs-Methoden zu erhöhen.
Experimentelle Auswertungen auf syntetischen Daten bestätigen, dass der Gaussische Prozess die NN-basierten Methoden bei Problemen in tiefen Dimensionen (1D-2D) übertrifft und zeigen, dass NOMU gleich gute oder bessere Resultate liefert als die anderen NN-basierten Methoden. Die Experimente in höheren Dimensionen jedoch suggerieren, dass die NN-basierten Methoden sich gegenüber dem mit Standartwerten konfigurierten Gaussichen Prozess verbessern und es schaffen diesen sogar zu überholen. Des Weiteren zeigen die Resultate dass speziell in höheren Dimensionen NOMU robustere und bessere Resultate erzielt als die anderen NN-basierten Methoden.

Contents

1

Introduction

Machine Learning (ML) and Neural Networks (NNs) are gaining wide popularity in the recent days due to their increased ease of use and the availability of the required resources. Both are used as tools for regression tasks, where a trained model approximates an unknown function to generate output values for new input values, and also in classification tasks where a trained model describes the separation between data points with different labels, such that the label for an new input value can be determined. Generally speaking, NNs can be applied in all sorts of research sectors and industries, as long as there are enough data points from which the model can learn. However, NNs are not a new tool and there has been much research put into the architecture of fast and accurate network models increasing the predictive power.
NNs are quite complex to understand and therefore often considered as black box systems. Often the networks are just applied as oracles that tell the truth without questioning the results or knowing the data it was trained on. The accuracy of the prediction is only considered during the training phase but no longer when making the predictions on new data. This is however problematic especially with the increasing use of transfer learning (Pan and Yang, 2009) where pre-trained models are retrained for different purposes or already trained models are just applied without knowing the exact training setup. Doing so it is very likely that a neural network model is fed with data that are far from the input space on which the model was trained for. The model will always give a prediction however in that case the quality of the prediction is questionable since the model wont be robust for input areas where no training data originate from (Novak et al., 2018). Following this explanation it is apparent why it is important to be able to make the model not only predict the target value but also its own internal model uncertainty. Especially, when the neural network model is used in a highly critical field of application, like self driving cars, medical diagnosis or fraud detection.

Being able to estimate the function representing the relation of different variables often one wants execute subsequent task with that function. One example for such an task is finding the optima of that unknown function, this problem is common in science and practice. For example which ration of two chemicals results in the strongest reaction or where on a goldmine is the most gold located. There are multiple optimization schemes on how to find the optima but all have in common that they need some samples to start

with. Each sample gives more information on how the unknown function might look like and for easy problems one can just take many samples however there are also problems where taking a sample requires a lot of resources such as time and money like the noted examples of the chemicals of the goldmine. Sampling the entire input space would be a costly operation, therefor other optimization strategies should be utilized.One of those is the Bayesian optimization (BO) which is an algorithm that takes the prediction of the function itself and the estimated uncertainty for that prediction into account and as a result proposes the point where the next sample should be taken (Jones et al., 1998). This new sample, together with all previous samples, is then used as new information base for the estimator to improve its model. The BO has the advantages that these samples are proposed in a manner that reveals the most possible information. As a result, the unknown function can be optimized without the need of a large amount of samples, thus BO is used for tasks where there is a fixed budget of evaluation, due to cost or time constraints. The propose the next sample the BO constructs optimizes the so called acquisition function which is a combination of the mean and uncertainty estimates. Consequently to be able to use NNs for BO it is required that the NN-based method also returns estimates for mean and uncertainty. The classical BO uses a Gaussian process as estimator to predict the uncertainty and the target function, however this method is difficult to apply in higher dimensions (Srinivas et al., 2012).
Using NNs that are able to estimate their model uncertainty allows the BO to construct acquisition functions even in higher dimensions as NNs can work well in with high dimensional inputs such as images (LeCun et al., 1998). Since Bayesian optimization, in its classical form, fails in more than 10 dimension having an alternative solution is really needed for systems that have an input space with more then 10 dimensions (Li et al., 2016). Since a large field of application of BO is the hyperparameter tuning of machine learning tasks, using NNs could allow the BO to optimize the hyperparameters for models with more than 10 parameters (Snoek et al., 2012). Like this larger Machine learning models can be optimized rather cost efficient which is relevant since most machine learning tasks require substantial resources for training and thus it is favourable to not repeat this process countless times to just find the best hyperparameter configuration.
New fields of applications are possible through the support of high dimensional inputs. First, it is possible to directly use images as inputs, because each image pixel can be considered as a new dimension. Second, the neural network formulation of uncertainty estimation function can help formulate the acquisition functions, which are used for the Bayesian optimization, in a way such that it can be optimized using a Mixed Integer Programming solver.
This thesis compares the novel method for uncertainty estimation with neural networks by (Heiss et al., 2021) called *Neural Optimization-based Model Uncertainty* (NOMU) with the *Deep Ensemble* method by (Lakshminarayanan et al., 2016) and the *MC Dropout* method by (Gal and Ghahramani, 2016). The focus of the comparison lies on their applicability for Bayesian optimization. The experiments in this thesis aim to highlight characteristics of the different methods in low and high dimensional settings. As a benchmark for the experiment the classical Gaussian Process is used. The methods are compared with respect to their accuracy of the found optima when used in com-

bination with BO. Computational resources and computation time are factors that are not included in the final analysis. The comparison is based on experiments on a set of synthetic functions with input dimensions between 1 and 20.
Based on the findings of the first experiments in lower dimensions adjustments are made to improve the methods and to encourage the escape from local optima. Subsequently the experiment in higher dimensions (5-20) are aimed to find out to what extend neural network models with model uncertainty can enable BO in higher dimensions where the Gaussian process is no longer applicable.

A highly modularized python library is developed to enable a fast and convenient experimental and analytic setup. This user-friendly library contains implementations of all three uncertainty-estimating NN-based methods, as well as the Bayesian optimization. The different algorithms are set up in a modular way so that for the existing sub-processes it can be easily switched between different implementations.
Experiments can be setup through a configuration file that defines which method and sub-processes should be used, and which hyperparameters can be applied. Every sub process is able to gather information which can be helpful for the analysis. Also it is possible to store all different network models that are trained in each step of the BO algorithm. For convenience there is basic data preprocessing included in the library as well as a specific plotting framework to enable consistent data inspection and interpretation.

The literature for the different fields such as Bayesian optimization and neural network uncertainties are reviewed in Chapter 2.
Chapter 3 introduces the important concepts of Bayesian probability and explains the Bayesian optimization algorithm. Additionally the term uncertainty is looked at in more detail. The estimator method used in this thesis are presented in Chapter 4, beginning with the NN-based methods before the basics of the Gaussian Process are explained. The acquisition function as a important part of the BO algorithm are discussed in Chapter 5 together with the different optimizers for the acquisition functions. The experimental work which has been executed for this thesis is listed in Chapter 6 and final Chapters 7 & 8 summarize the results of this thesis in the conclusions and highlights possible future work which can be build on the foundations of this thesis.

2

Literature Overview

2.1 Uncertainty Quantification for Neural Networks

The quantification of uncertainty of NN-estimates has been an active field of research for a while already. However, there have been several different focus points over various past works. Early publications such as (Chryssolouris et al., 1996) and (Khardon and Wachman, 2007) focused on the data noise perspective where the assumption is that the sampled data may return different target values for the same input if sampled multiple times. Frénay and Verleysen (2013) give a nice overview of the case when the label, target values for classification tasks, is noisy and how to deal with that uncertainty. Conversely, most of the work in that respect aims to remove or reduce the effect of noisy samples, for example, by detecting outliers in the training data (Sigurdsson et al., 2002). The perspective of model uncertainty, where the goal is to quantify the uncertainty of the model itself, irrespective of data noise, for example due to missing training data in certain regions has only recently been addressed by research groups. Works by Khosravi et al. (2010) and Pearce et al. (2018) constructed networks with multiple outputs where some of them are aimed to train upper and lower bound for the uncertainty.
One approach that is often used for quantification of the uncertainty of a Bayesian neural networks is the variational inference (Graves, 2011; Blundell et al., 2015; Hernández-Lobato and Adams, 2015). With the help of the inference the posterior distribution over the model weights is approximated which then represents the model uncertainty. Gal and Ghahramani (2016) showed that Bayesian inference can be approximated by using Dropout layers in the NN-architecture. In this case the model uncertainty is a result of an stochastic forward pass through the NN. More recently the authors have extended the application of this Dropout based uncertainty estimation to convolutional neural network to apply it on images (Gal et al., 2017).
Very recently Curi et al. (2020) presented an algorithm which uses *hallucinated* control action to control the model uncertainty to achieve an *optimistic* exploration in the context of reinforcement learning. In reinforcement learning such an *optimistic* approach explores actions that are expected to yield high values, so called Q-values. In this context *optimistic* can be seen as focusing on the upper bound and thus the algorithm favours to learn uncertain but high valued action over certain lower valued actions.

2.2 Bayesian Optimization

When an algorithm is able to find the optima of a function with the least amount of evaluations is called to be *efficient*. By employing some prior assumptions over the objective function allows BO to be efficient. This prior is based on the Bayes' theorem which gives the method its name.
The idea of applying prior beliefs to enhance optimization methods is not a new concept. The basic theory in global optimization originates from the work of Kushner (1964), and was further developed by Zhilinskas (1975) and Močkus (1975a). However the theory did not get much recognition until Jones et al. (1998) combined theory from several literatures into one paper and showed that in terms of minimal function evaluation the BO is one of the most efficient approaches. Not only the field of global optimization was using that approach of using the prior, similar approaches were also found in the field of mathematical geology where it was called *kriging* and in the field of statistics (Jones et al., 1998).

Bayesian optimization is particularly relevant on the domain of global optimization where the objective function has no know expression and also its derivatives are unknown (Brochu et al., 2010). Bayesian optimization is based on stochastic processes, Gaussian processes in particular, which assign a probability distribution over functions. These distribution can incorporate priors in the form of Kernels. Williams and Rasmussen (2006) describe these kernels thoroughly and they also make the connection between Gaussian processes and NNs.

The Bayesian optimization is a wide term and is root to many inherit method such the efficient Global Optimization also known as Sequential Kriging Optimization (Huang et al., 2006). As a result of the wide range interpretation of the BO there is a lot of research in this field where different aspects are being improved. For example the further reduction of the number of evaluations used which can be achieved by drawing data with a lower fidelity from a surrogate function to the objective (Huang et al., 2006). Or the application of BO to constrained or multi-objective optimization (Močkus and Močkus, 1991).

With the recent development in machine learning and the contribution of Snoek et al. (2012) exploring the usability of the BO for the hyperparameter tuning, the research around BO grew in relevance. With the black-box characteristics of machine learning algorithms and their vast complexity and thus non-trivial or no-existing closed form formulation they are a perfect fit for the application in BO. Additionally for larger network sizes the characteristic of long training times and thus great evaluation cost is also given. Further research on the hyperparameter setting has shown that similar tasks can be optimized together meaning that the function evaluation on one task, so for one objective function, can also be used for a similar but different objective function (Swersky et al., 2013). Wang et al. (2020) investigated on the parallel optimization using Bayesian optimization which got more relevant due to the use of distributed computing

using cloud computing.
Additional to the over all BO research a whole research field on its own is based around the optimal strategies of finding the next point to evaluate. The next evaluation point is determined by the optima of the acquisition function which is usually a combination of the posterior and the uncertainty. The intuitive upper-uncertainty-bound *GP-UCB*, a linear combination of mean and uncertainty, is being analyzed by Srinivas et al. (2009). The most popular acquisition function, the Expected Improvement is investigated by Jones (2001b). But there are a vast amount of acquisition function variations being researched on.
It is important to mention that there are plenty more past and current research being done on the subject, including summary papers and tutorials such as (Frazier, 2018), (Brochu et al., 2010) and (Sasena, 2002).

2.3 High Dimensional Bayesian Optimization & Performance Benchmarks

Bayesian optimization is a research field that is very well explored. However classical BO lacks in ability to solve problems in higher dimensions. It may be well disputed where exactly the critical value lies for the maximal supported dimensionality however it can be agreed on that is might be around 10 (Wang et al., 2016), or lower. With the more recent applications of BO for hyperparameter tuning in machine learning (Snoek et al., 2012) this excludes models with many hyperparameter from being optimized by Bayesian optimization.
In the same context of BO for hyperparameter tuning, Malkomes and Garnett (2018) present an algorithm that uses Bayesian optimization to tune Bayesian optimization itself. Their algorithms also rely on a uncertainty notation when selecting the next point of evaluations. They use an ensemble of models to create such a uncertainty notation and to refine the overall process of the BO.
Another intuitive algorithm for solving the optimization issue in higher dimension by Moriconi et al. (2020) is to apply the optimization on the lower dimensional sub spaces and the reconstructing the initial input space.
The issue of high dimensional problem is tackled in the works of Wang et al. (2016). The proposed approach called *REMBO* uses random embedding to project the problem into a different frame. Using this adaption they are able to show that the problems with large extrinsic but small intrinsic dimensionality can still be solved with BO. Thus with this approach a system does not need to be manually reduced to its core dimensions, however this method does still not solve the issue with problems that have high intrinsic dimensionalities.
Another approach that addresses the high dimensional problems but also the computational expensiveness of function evaluations is called BOHAMIANN and is presented in the works of Springenberg et al. (2016). Through the application of Bayesian networks and subsequent Bayesian inference and scale adaption they achieve an robust optimizer

that is also scalable. Their approach is based on Hamiltonian Monte Carlo Methods as presented in (Chen et al., 2014). For their experiment they focused on the benchmark setting presented in (Eggensperger et al., 2013) which includes synthetic functions as well as hyperparameter optimization tasks.

3

Problem Overview

This chapter gives an overview of the problem of model uncertainty and optimization, together with the basic formal models of the underlying theory. It begins with the introduction of the concept of Bayesian probability to build the basis to the probabilistic deductions used for the Bayesian optimization. Uncertainty plays such a big role in the Bayesian optimization as well as the probabilistic methods that are used in this thesis, therefore the term of uncertainty is inspected in more detail and different aspects of uncertainty are highlighted.

3.1 Bayesian Probability

To understand the concept of Bayesian optimization first the basic concept of the Bayesian probability has to be introduced. In contrast to the classical probability which is a physical property of the world, the Bayesian probability is based on a belief about the frequency of an event (Heckerman, 2008). Instead of estimating the physical probability by means of repeated sampling, in Bayesian probability a belief is formulated about the uncertainty of the physical probability after some amount of samples are being taken. This belief, together with rules of probability, is then used to compute the probability of the outcome in the next sample (Heckerman, 2008).

For common understanding let's introduce the following notation. Let's denote the random variable for an outcome or event as θ. Together with the belief or prior information ξ the uncertainty about the random variable can be expressed by the probability density function $p(\theta|\xi)$. The distribution after drawing samples is called posterior distribution $p(\theta|\mathcal{D})$ with $\mathcal{D}$ being the set of samples. Having defined these variables now the Bayesian theorem can be used in conjunction with the Bayesian rule to derive a formula for the Bayesian probability of the random variable θ given the set of samples $\mathcal{D}$ and the prior belief ξ, $p(\theta|D, \xi)$.

Theorem 1 (Bayes' theorem)**.** *The posterior probability of a model M, given some evidence E is equal to the likelihood of the evidence E appearing given a that model M is true times the prior belief about the probability of the model P(M) divided by the*

independent probability of the evidence E appearing.

$$P(M \mid E) = \frac{P(E \mid M)P(M)}{P(E)} \tag{3.1}$$

The probability distribution of the random variable given the previous samples and the prior knowledge is given by:

$$p(\theta|\mathcal{D}, \xi) = \frac{p(\theta|\xi)p(\mathcal{D}|\theta, \xi)}{p(\mathcal{D}|\xi)} \tag{3.2}$$

where the term $p(\mathcal{D}|\theta, \xi)$ is known as the likelihood function of the sampling.
Using the joint probability distribution it follows that $p(\mathcal{D}|\xi)$ can also be expressed as:

$$p(\mathcal{D}|\xi) = \int p(\mathcal{D}|\theta, \xi)p(\theta|\xi)d\theta \tag{3.3}$$

This probability $p(\mathcal{D}|\xi)$ can be interpreted as the chances of certain samples in $\mathcal{D}$ to appear when the prior belief ξ is applied. Thus one can predict which future points are more likely to be sampled.

3.2 Bayesian Optimization

For Bayesian optimization the goal is to optimize an unknown function f within a given limit of l function evaluations.

$$x^* = \underset{x \in X}{\operatorname{argmax}} f(x) \quad \text{and} \quad \hat{x}^* = \underset{x \in \mathcal{D}^l}{\operatorname{argmax}} f(x) \tag{3.4}$$

where:

- x^* is the true, analytical optima of the target function f
- $\hat{x}^*$ is the found optima as result of the Bayesian optimization
- $\mathcal{D}^l$ is the set of point-samples after l function evaluations

It is important that the optimization algorithm converges ($\hat{x^*} \to x^*$) within the given number of samples to have a successful optimization.
Bayesian optimization is based on the uncertainty definition introduced in the previous Section 3.1. However instead of assuming the random variable θ to be a numerical value it is now considered to be function itself, the random function. This random function is also following the rules of a certain distribution which is defined by the prior belief, previously noted as ξ. Each of the samples is a new function, which is a regression result through the point-samples ($\mathcal{D}$). All function samples act as one sample in the Bayesian formulation from equation 3.2. From the probability distribution of this random function the mean function $\mu(x)$ can be derived as well as uncertainty function $\sigma(x)$. Both of these functions have a closed form formulation. Thus, the Bayesian optimization can

optimize the mean function $\mu(x)$ with the use any optimization methods, such as Grid-search, Brent-Search or others. The input value of the resulting optima is then used as the input to the next function evaluation. However rather than directly optimizing the mean function $\mu(x)$ the Bayesian optimization combines the mean and uncertainty functions together into a so called acquisition function. In this acquisition function it is possible to favor points where there is a lot of uncertainty about the function value over the highest predicted mean value. There are plenty different formulations of such acquisition functions. Section 5.1 discusses the most common acquisition functions which are also used in this these in more detail.

Algorithm 1: Bayesian Optimization

Data: $[x_1, ..., x_l] = \mathcal{D}^l$
Result: $\hat{y^*} \coloneqq$ guess for $\max_{x \in \mathcal{X}} f(x)$
1 N := number of BO steps;
2 n = 0;
3 **while** $0 \leq n \leq N$ **do**
4 train the estimator with the given samples $\mathcal{D}^{l+n}$ and their function evaluations;
5 optimize the acquisition function, use mean and uncertainty estimates $\hat{\mu}(x)$, $\hat{\sigma}(x)$ of estimator $\implies x^*$;
6 add sample x^* to the sample set $\implies \mathcal{D}^{l+n+1}$;
7 **end**
8 $\hat{y^*} = \max_{x \in \mathcal{D}^{l+N+1}} f(x)$

In classical Bayesian optimization the method of regressing the point-samples (part of line 5 in Algorithm 1) is the Gaussian process. This regression method is introduced in more detail in Section 4.2. However there are many other methods of regression that can be used and most modern ones are based on NNs, as shown in Section 4.1.

As previously mentioned, optimizing the evaluation of the acquisition function is less expensive than optimizing the underlying target function. Consequently, the choice of optimization schemes is much wider since they do not have to be as efficient. There has been plenty of research about which optimization schemes work best for which situation (Wilson et al., 2018). Section 5.2 gives a brief overview over the most important acquisition function optimizers relevant for this work.

3.3 Uncertainty Excursion

Uncertainty plays a key role in this thesis. It allows to asses the confidence of neural network predictions and, in combination with BO, allows for confidence guided search and thus enables exploration. However, uncertainty is not that clear of a term. There are different understandings of uncertainty that have to be distinguished for this work.

When doing regression, the goal is to find a model M such that the following holds:

$$M(x_i) = y_i = f(x_i) + \sigma \tag{3.5}$$

where f is the true, but unknown function, x_i and y_i are the input and the respective output values of the samples and σ is the noise. Noise can be the result additional explanatory factors that are not respected in M or inaccuracies in the measurement of y_i (Pearce et al., 2018). If the evaluation of a certain sample always returns the same target value, usually the data is considered to be noiseless. However, in practice, due to measurement errors or other inconsistencies, one should always consider data noise or data uncertainty. Similarly, the model estimates can also depend on the model parameters (i.e., the samples a models was trained with) which results in different estimates for the same inputs, meaning the model is not certain about the estimation. In such cases, one can speak about model noise or model uncertainty. If the regression estimate $\hat{f}(x^*)$ of a new sample x^* is the result of a stochastic process it is the result of a posterior distribution $p(f(x^*)|\mathcal{D})$. In this case there is always some uncertainty attached to the distribution which results in model uncertainty. Thus the overall uncertainty σ can be split into the data noise σ_n (aleatoric uncertainty) and the model uncertainty σ_f (epistemic uncertainty).

$$\sigma_y^2 = \sigma_n^2 + \sigma_f^2 \tag{3.6}$$

In the setting of the ongoing sampling during the BO these two sources of uncertainty have different characteristics. First, data noise does not increase or decrease depending on the number of evaluations, but the model uncertainty is influenced by the number of samples. The expected behaviour is for the uncertainty to decrease when more data points are fed to the regression, as shown by Lakshminarayanan et al. (2016)[1]. The difference between model and data uncertainty can made clearer when looking at uncertainty intervals (Heiss et al., 2021).

Definition 1 (Credible Intervals (CI))**.** *An α-CI at $x_i \in X$ with $\alpha \in (0, 1)$, is defined as an Interval I such that $p(f(x_i) \in I|\mathcal{D}) = \alpha$*

Definition 2 (Prediction Interval (PI))**.** *An α-PI at $x_i \in X$ with $\alpha \in (0, 1)$, is defined as an Interval I such that $p(y_i \in I|\mathcal{D}) = \alpha$, where $y_i = f(x_i) + \sigma_n$*

The model uncertainty originated from the probabilistic process behind the estimation method and is always present is such models. The data noise however depends on the data generating process which is problem specific and not related to the estimator. As a result, it is possible to have problems that do not have any data noise, in which case the

[1] In (Lakshminarayanan et al., 2016) different terms are used such as *calibration* and *out-of-distribution*. In simple words *calibration* describes the difference between predicted uncertainty by the model and the true uncertainty which is encoded in the data. *Out-of-distribution* can be explained as the effect of increasing uncertainty when the input data is shifted away from the data that the model was trained on.

Prediction Interval equals the *Credible Interval*. Throughout this thesis only noiseless regressions are taken into account, where it assumes no data noise and the only effect is the model uncertainty itself.

4

Probabilistic Methods in Machine Learning

In this chapter all main NN-based estimators are formally introduced. Namely these are the *neural-optimization based Model-uncertainty* (NOMU), the Deep Ensemble (DE) and the Monte Carlo Dropout (DO) approach. The Gaussian process (GP) is also introduced as a widely used probabilistic method, which however is not based on NNs. For all estimators the formal definition is given as well as the basic theoretical reasonings. In addition important characteristics for this thesis are highlighted and also adjustment to the original formulations are made to fit the setting of this thesis.

4.1 Uncertainty Estimation for Neural Networks

Neural networks and deep neural network got a lot of attention in the past years ergo there has been a lot of research effort on them. As a result the prediction performance increased considerably. However despite all the research efforts, it is still an unsolved problem how one can quantify the predictive uncertainty on NNs (Lakshminarayanan et al., 2016). This section introduces multiple approaches which allow NN-models to quantify their own model uncertainty. First the very recent *NOMU* approach of Heiss et al. (2021) is introduced followed by the *Monte Carlo Dropout* method and the *Deep Ensemble* method.

4.1.1 Neural optimization-based Model Uncertainty (NOMU)

The *Neural Optimization-based Model Uncertainty* (NOMU) approach by Heiss et al. (2021), in contrast to the later introduced Deep Ensembles and MC Dropout methods, actively learns the uncertainty bounds during the training and does not derive the uncertainty from the trained network itself. To achieve this, the NOMU defines a network architecture which has two outputs, one for the model prediction $\hat{f}$ and the other for the model uncertainty $\hat{\sigma}$. When there is data noise, then the model prediction $\hat{f}$ can be interpreted as mean prediction $\hat{\mu}$. For reasons of consistency from now on always $\hat{\mu}$ is used to denote the model prediction, even for noiseless data. The following theoretical foundations are based on the work of Heiss et al. (2021).

Network architecture

The architecture of the NOMU model, as described in (Heiss et al., 2021), consists of two parallel networks. One of these networks will learn the target function $\hat{\mu}$ and will further be referred as the *main-net*. The other network is referred as the *side-net* and is responsible for training the model uncertainty $\hat{\sigma}$. The input to the whole model is forwarded to the main and the side-net equally. Both, side and main network have one output each. However the output of the main-net is purely based on the last hidden layer L^{K-1} of the the main-net itself. The side-net's output however depends on the output of the last hidden layer of both the side-net and main-net. The connection from the main-net to the side-net's output allows for implicit transfer learning and is indicated in Figure 4.1 by the green dashed lines. Consequently the uncertainty (output of the side-net) can also depend on the model prediction. Speaking in word of functions, this means that the uncertainty can vary depending on the function value itself. Like this, regions of interest for the learned function, will also be regions of interest for the uncertainty. The exact architecture of main and side-net can vary. The number of layers and nodes per layer can be adjusted depending of the problem that it faces. There is no constraint that forces main and side-net to have the same architecture, so in principal they can differ is number of layers and number of nodes per layer. For this thesis however main and side-net will always have exact the same structure.

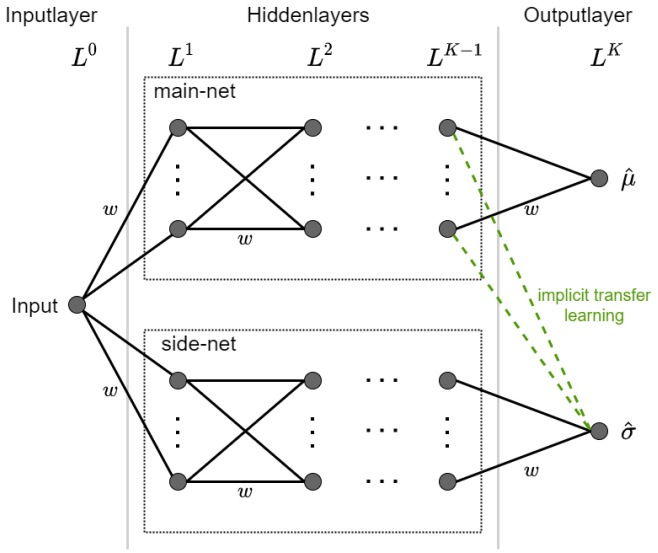


Figure 4.1: **NOMU network architecture.** Illustration of the network architecture of the NOMU method with the main and side-net structure and the implicit transfer learning connection.

The whole architecture with both, side and main-net, uses the *Rectified Linear Unit* (ReLU) activation function for all nodes. The activation function in NNs is responsible to transform all the summed up inputs of an node into its output (Figure 4.2). In the

case of the ReLU activation if the sum of the inputs is positive then the output is equal to that sum. If it is negative, then the output is zero ($\mathrm{ReLU}(x) \equiv \max\{0, x\}$). The architecture is trained with T epochs of gradient decent and with a ridge regularization parameter $\lambda > 0$. Consequently, the resulting loss, including the regularisation is defined as follows:

$$\mathcal{L}_{\text{final}} \coloneqq \mathcal{L}_{\text{NOMU}} + \lambda \|\mathbf{w}\|_2^2 \tag{4.1}$$

By adding the regularization factor together with the L2-norm of the weight vector encourages the model to assign smaller weights if possible and makes the regression a *ridge regression*. As a result by assigning zero weight instead of a small weight to some of the nodes inputs the model becomes less complex since less variables have to be considered during the calculations of the node output (Figure 4.2). Also using regularization the model usually tend to generalize better and over fit less.

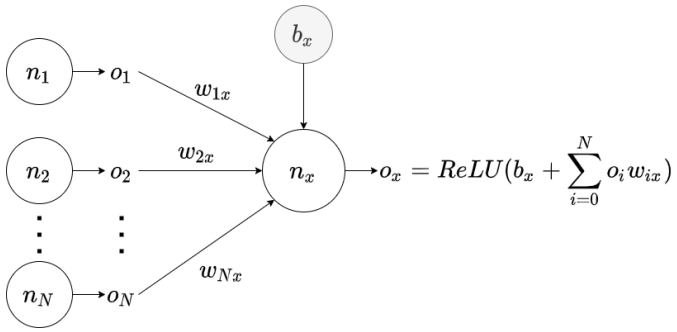


Figure 4.2: **Neural network node.** Illustration of network node with output of previous nodes as input as well as the weights. As activation the ReLU is used as seen in the formula for the output.

The intuitive idea behind the NOMU method can be explained using the analogy to an elastic wire which represent in this case the uncertainty. A key aspect of NOMU is the use of augmented data points which act as additional training points but only for the prediction of the uncertainty and not for the model prediction. These augmented data points have a constant force that pulls up the uncertainty. Additionally the loss is constructed in a way that everywhere where there are (real) training points the uncertainty "wire" is fixed at the model prediction[1] but in regions without training data the uncertainty wire is elastic and the applied "force" results in an increase of the uncertainty.

[1] In the noiseless case the uncertainty should be zero at the sample point itself since the sample itself provides the evidence.

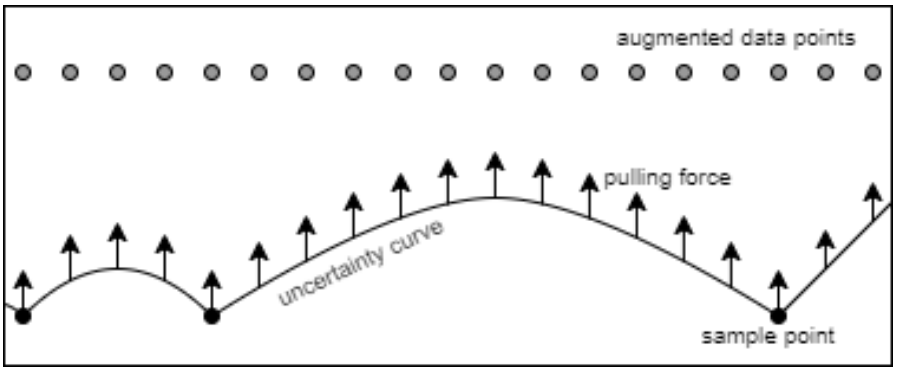


Figure 4.3: **NOMU elastic wire intuition.** Illustration of the intuitive concept of an elastic wire as the upper uncertainty bound

The idea of the augmented data points is motivated by a desired characteristic that the model uncertainty in regions with less training points should be larger than where there are many training points. This desiderata is listed in the work by Heiss et al. (2021) as Desiderata D1 . Another desiderata that is related to the augmented data points is D3 which states that for the noiseless case it should be possible to have zero model uncertainty at training points. To really enforce these characteristics however a fitting loss function has to be constructed.

Loss

Given a training process where D^{train} is the set of training points and $\hat{\mu}$ is the model prediction and $\hat{\sigma}$ is the predicted model uncertainty. Then the loss function of the NOMU method is defined as follows:

$$\mathcal{L}_{\text{NOMU}} \coloneqq \underbrace{\sum_{(x,y)\in D^{train}} \ell(\hat{\mu}(x), y)}_{\text{(i)}} + \underbrace{\pi_{sqr} \cdot \sum_{(x,y)\in D^{train}} (\hat{\sigma}(x))^2}_{\text{(ii)}} + \underbrace{\pi_{exp} \cdot \frac{1}{\lambda_s(\mathcal{X})} \int_{\mathcal{X}} e^{-c_{exp}\cdot\hat{\sigma}(x)} dx}_{\text{(iii)}} \tag{4.2}$$

The first sum (i) ensures that the model prediction $\hat{\mu}$ is as close as possible to the true value given by the sample target y. ℓ is any pointwise error function that ensures that the regression task is learned. The state of the art error function that is used for this is the squared loss.
The second sum of the loss (ii) controls the prediction of the model uncertainty at the sample points. In general the model uncertainty at a sample point should be as small as possible. For a problem setting without data noise it even should be zero. Including the $\hat{\sigma}(x)^2$ output in the loss guides the model to keep the uncertainty small at the sample points.
However with only (i) and (ii) only the sample points are considered in the loss. To make use of the augmented data points a third term to the loss is added. This third part (iii) ensures that the uncertainty $\hat{\sigma}$ is not zero for the augmented data points as

the goal is to have non-zero uncertainty in regions where there are no samples.

In practice the integral in (*iii*) is approximated with a Monte-Carlo integration using the augmented data points mentioned above as the Monte-Carlo samples. With the selection of the squared loss error function in (*i*) and the Monte-Carlo integral using L uniformly distributed augmented data points ($D^{aug} \coloneqq \{x_1, ..., x_L\} \overset{i.i.d}{\sim} Unif(\mathcal{X})$) this results in the following practical loss formulation.

$$\mathcal{L}_{\text{NOMU}_{\text{practical}}} \coloneqq \underbrace{\sum_{(x,y)\in D^{train}} (\hat{f}(x) - y)^2}_{\text{(i)}} + \underbrace{\pi_{sqr} \cdot \sum_{(x,y)\in D^{train}} (\hat{\sigma}(x))^2}_{\text{(ii)}} + \underbrace{\pi_{exp} \cdot \frac{1}{L} \sum_{x\in D^{aug}} e^{-c_{exp}\cdot\hat{\sigma}(x)} dx}_{\text{(iii)}} \tag{4.3}$$

Each of the different sums is weighted relative to the others. As a result the parameter for the first sum is fixed to one and for the second and third sum the hyper parameter π_{sqr} and π_{exp} are introduced. With π_{sqr} it can be controlled how strictly the uncertainty at the sample points should be forced to be zero. π_{exp} controls how much the augmented data points are weighed against the sample points.

Mix Integer Programming Support:

One major advantage of the NOMU method is that it predicts the model uncertainty with its own neural network (side-net). This network expression allows solving for its optima using a Mixed Integer Programming Solver such as CPLEX (IBM, 2021). More specific explanations how to optimize the uncertainty using the MIP follow in Section 5.2.3. This applicability of the MIP can be useful for subsequent tasks operated on the uncertainty. The straight forward example for this is the acquisition function optimization in the BO. Instead of having to sample points repeatedly the MIP can be used which calculates the optima based on the NN-structure and its learned weights. This it especially promising in higher dimensions where the number of samples needed for a reliable optimization grows fast.

Uncertainty Bounding activation:

The NOMU method applies a specific activation function to the output of the side-net, thus the predicted model uncertainty. This activation function T incorporates a notion of maximal and minimal uncertainty (ℓ_{min}, ℓ_{max})

$$T(\sigma) \coloneqq \ell_{max} \left(1 - exp\left(\frac{-(ReLU(\sigma) + \ell_{min})}{\ell_{max}}\right)\right) \tag{4.4}$$

Using this activation it is ensured that the uncertainty is bounded. The third part (*iii*) of the loss function (Equation 4.3) tries to assign as high of value as possible to the uncertainty output for the augmented data points. Therefore, an upper bound to the uncertainty is useful because else it can occur that, especially at the boundaries of the

input space, the uncertainty grows extremely large if there is a lack of sample points. Since the mean width of the uncertainty bounds are used as a metric in this thesis, letting the uncertainty grow to big at the boundaries would lead to a miss interpretation of the mean width as most of its value would come from the boundary. Also when progressing with BO very big uncertainty values at the boundaries result in acquisition functions which always suggest boundary points to be the next points to evaluate. Evaluating boundary point generally reveals less information than a point inside the input space. For an intuitive understanding of this one can consider again the illustration of the NOMU as an elastic wire. Not only can the augmented data points be considered as constant force pulling up but also the sample point can be considered as hooks where the wire is fixed. When adding a sample to a situation where the elastic wire previously was above the sample position, then the hook pulls down the wire compared to the previous situation. The information gained can be interpreted as the number of wire segments that change position by adding the new sample. When the new sample is at the boundary half of the segments affected lie outside the input space and are thus irrelevant.
Besides adding an upper bound to the uncertainty, the activation also imposes a lower bound. Such a lower bound can increase numerical stability since values close or equal to zero can result in problematic behaviour.
However using the above activation introduces some non-linearity into the formulation and thus it is no longer possible to apply Mixed Integer Programming. Thus there is a secondary formulation of the activation which only use linearities in form of ReLU.

$$T(\sigma) \coloneqq \ell_{min} + ReLU(\sigma - \ell_{min}) - ReLU(\sigma - \ell_{max}) \tag{4.5}$$

4.1.2 Monte Carlo Dropout

In their work Gal and Ghahramani (2016) show that applying a dropout layer to each weight layer is equivalent to a probabilistic deep Gaussian Process (Damianou and Lawrence, 2013).
By adding a dropout layer to a neural network layer the nodes in that layer are dropped randomly with a certain probability p. In other words in every pass for each node a independent coin is tossed. This coin has a probability of p to show heads. If heads is shown then the node is ignored by setting its output to zero. As a result the number of active nodes which contribute to the sum inside the activation function vary from training pass to training pass and also a certain node only sometimes plays a role in the network. Usually dropout is used in NNs to avoid overfitting (Srivastava et al., 2014).
The findings of Gal and Ghahramani (2016) are completely independent of the underlying network structure, meaning this method can be applied to any network architecture to enhance it with a uncertainty estimating behaviour. In the following the a brief introduction into the MC Dropout methods and its derivation is given as in (Gal and

Ghahramani, 2016).

The goal is to find a formulation for the loss function of a network applying dropout to each of its $j = 1, ..., L$ layers using $\mathbf{W}_j$ to denote the weights and $\mathbf{b}_j$ the bias of that layers. We denote $\hat{y}$ as the output of the NN-model. Given that the underlying network (without dropout) has a loss function $\mathcal{L}(\cdot, \cdot)$ the combined loss for the network with dropout and x_i, y_i the input output pairs for $1 \leq i \leq N$ is given by.

$$\mathcal{L}_{Dropout} \coloneqq \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{y}_i) + \lambda \sum_{j=1}^{L} (\|\mathbf{W}_j\|_2^2 + \|\mathbf{b}_j\|_2^2) \tag{4.6}$$

Once again λ denoted the weight decay of the L_2 regularisation.
MC Dropout can be seen as some sort of variational inference for a Bayesian NN. Thus let's denote $\boldsymbol{\omega} = \{\mathbf{W_i}\}_{i=1}^{L}$ the set of weights in the neural network with L layers which also can be viewed as set of random variables in a probabilistic model. Now the predictive distribution for a new input-output pair (x^*, y^*) is given by:

$$p(y^*|x^*, \mathbf{X}, \mathbf{Y}) = \int p(y^*|x^*, \omega) p(\omega|\mathbf{X}, \mathbf{Y}) d\omega \tag{4.7}$$

In variational Inference a variational distribution $q(\omega)$ is defined which should approximate the prior $p(\omega|X, Y)$ as closely as possible. This can substituted into Equation 4.7 to result in:

$$p(y^*|x^*, \mathbf{X}, \mathbf{Y}) = \int p(y^*|x^*, \omega) q(\omega) d\omega \tag{4.8}$$

To ensure that the approximation $q(\omega) \approx p(\omega|\mathbf{X}, \mathbf{Y})$ is as accurate as possible one can use the Kullback-Leiber (KL) divergence which measures the dissimilarity between two distributions. Hence the goal is to minimize KL divergence.

$$\underset{q}{argmin}\ \mathrm{KL}(q(\omega) \| p(\omega|\mathbf{X}, \mathbf{Y})) \tag{4.9}$$

Instead of minimizing the KL divergence however, it is also possible to maximize the log evidence lower bound:

$$\int q(\omega) log(p(Y|X, \omega)) d\omega - KL(q(\omega) \| p(\omega)) \leq L = log(p(Y|X)) \tag{4.10}$$

Where L is the log-likelihood. This can now be viewed as the new objective. To have a consistent notion of loss this can be negated to get the loss objective for the variational Inference. Conversely to the objective for the loss the goal is to minimize it.

$$\mathcal{L}_{VI} \coloneqq -\left(\int q(\omega) log(p(Y|X, \omega)) d\omega - KL(q(\omega) \| p(\omega)) \right) \tag{4.11}$$

Gal and Ghahramani (2016) show that this variational inference objective ($\mathcal{L}_{VI}$) is equivalent to the Gaussian process approximation of a NN objective when using Dropout as

the regularisation ($\mathcal{L}_{Dropout}$). Training a network with dropout is equivalent to performing variational inference thus training a network is also equal to drawing a sample from the variational distribution $q(\omega)$. The predictive uncertainty $p(y^*|X,Y)$ thus approximates the expectation over neural network solution derived by training with Dropout. This expectation can then again by approximated using Monte Carlo.

$$p(y^*|X,Y) = \frac{1}{M}\sum_{m=1}^{M} p(y^*|x^*,\omega^m) \tag{4.12}$$

where m is one neural network result of a training with an individual dropout pattern. Given a neural network which is trained using dropout methods, denoted as $\mathcal{NN}_m(x)$, the mean prediction $\hat{\mu}(x)$ and the uncertainty prediction $\hat{\sigma}^2(x)$ can be noted as follows.

$$\hat{\mu}(x) \coloneqq \frac{1}{M}\sum_{m=1}^{M} \mathcal{NN}_m(x) \tag{4.13}$$

$$\hat{\sigma}(x)^2 \coloneqq \frac{1}{M}\sum_{m=1}^{M} (\mathcal{NN}_m(x) - \hat{\mu}(x))^2 + \sigma_n^2(x) \tag{4.14}$$

To be able to derive the uncertainty and the mean prediction using MC dropout the same trained model has to be evaluated M times for each guess for $\hat{\mu}(x)$ and $\hat{\sigma}^2(x)$. The term $\sigma_n^2(x)$ represents the part of the data noise in the case for a noisy setup in the noiseless case it resolves to zero.

4.1.3 Deep Ensembles

The theory behind the Deep Ensembles as described by Lakshminarayanan et al. (2016) is multi fold. First the concept of ensembles is used and then several additions are made by defining proper scoring rules (loss) and training processes.
The concepts of using ensembles is not all that new. Using ensembles means that rather than training only one single network, multiple network are trained and then combined. Due to some random initialisation which differ between the different training processes the resulting networks do not always produce exactly the same predictions and do not have the same weights. Perform model combination on that ensemble one gets a more robust and thus more powerful model. It is worth mentioning that model combination performed by ensembles contrast the Bayesian model averaging (BMA) which performs only soft model selection and thus tries to find the single best model Lakshminarayanan et al. (2016) within the set of models. According to Lakshminarayanan et al. (2016) one can treat the ensemble as a mixture model which is uniformly weighted. For an ensemble with M network models the predictions can be combined into a mixture of Gaussian distributions.:

$$p(y|x) = \frac{1}{M}\sum_{m=1}^{M} p_{\theta_m}(y|x, \theta_m) \tag{4.15}$$

where θ_m are the parameters of the m-th model in the ensemble. From this mixture we can draw again an mean μ^* and uncertainty σ^*.

$$\mu^*(x) = \frac{1}{M}\sum_{m=1}^{M} \mu_{\theta_m}(x) \tag{4.16}$$

$$\sigma^{*2}(x) = \frac{1}{M}\sum_{m=1}^{M} (\sigma^2_{\theta_m}(x) + \mu^2_{\theta_m}(x)) - \mu^{*2}(x) \tag{4.17}$$

The resulting model thus predicts then according to a normal distribution $\mathcal{N}(\mu^*(x), \sigma^{*2}(x))$. Using this ensemble formulation it is possible to receive a uncertainty estimation even if every single network model has only one output and hence does not incorporate any uncertainty by itself like the NOMU model does. The method of model combination is thus also possible for a setting without data noise where the resulting uncertainty is a pure epistemic uncertainty. However Lakshminarayanan et al. (2016) base their work on a setting with data noise. As a result normally distributed data is assumed. To be able to incorporate the data variance a model architecture with two outputs in the last layer is used. One output for the mean μ_{θ_m} and one for the uncertainty $\sigma\theta_m$. Each observed value thus is just a sample form a Gaussian distribution. The prediction which the model makes is the mean and the variance of that said Gaussian distribution. However, having two network output, standard loss functions like MSE are no longer applicable, thus Lakshminarayanan et al. (2016) suggest the negative log-likelihood (NLL).

$$\mathcal{L}_{\text{NLL}} \coloneqq -log(p(y|x)) = \underbrace{\frac{log(\sigma^2(x))}{2}}_{\text{(i)}} + \underbrace{\frac{(y-\mu(x))^2}{2\sigma^2}}_{\text{(ii)}} + C \tag{4.18}$$

This loss is used individually for each network model in the ensemble during training. The intuition behind the negative log likelihood is that the numerator of the second term (ii) makes sure that the mean prediction μ is as close to the observed y as possible. The denominator of that term allows that the deviation of the mean from the observed y can be large. However in this case the variance σ^2 has to be large as well. This allows the model to predict large variance in regions where noisy observations are made. The first term ensures that the model does not just scale up the variance to infinity to always negate the effect of the deviation of mean to the y. However, the first term (i) uses the logarithm of the variance predicted by the model and the model in theory is allowed to predict negative or zero values. Thus, for numerical stability it is required that the σ used in the log likelihood is always positive. This can be achieved by applying the softplus function to the variance before using it in the NLL.

$$f_{\text{softplus}}(x) = log(1 + exp(x)) \tag{4.19}$$

Noiseless Setting

As already indicated the Deep Ensembles as described by Lakshminarayanan et al. (2016) focus on the case where the observation can be noisy. For this thesis however the focus lies on noiseless data were exact function evaluations are assumed. In this case the formulation (Equation 4.18) is no longer applicable because using negative log-likelihood can not ensure zero variance at the observation and therefore some modifications to the loss have to be made. First of all the network architecture is changed to only return one prediction that is the mean. Predicting the variance is no longer meaningful since all training data will always have zero variance. However there is another mechanism to "produce" uncertainty without having a secondary network output. The state of the art Mean Squared error (MSE) loss function can be used.

$$\mathcal{L}_{\text{MSE}} \coloneqq \frac{1}{M}\sum_{i=1}^{M}(\hat{\mu_i}(x) - y_i)^2 \tag{4.20}$$

where $\hat{\mu}_i(x)$ is the prediction of the network, y_i is the true function value for each of the training points x_i and M is the number or models in the ensemble.

For the noiseless case the Deep Ensemble method that is used in this thesis is mainly based on the model combination where all the models are combined to produce a joint distribution similar to the MC Dropout model.

4.2 Gaussian Process

Section 3.1 and 3.2 presented that the Bayesian optimization employs some kind of prior belief about the characteristics of the target function. The most commonly used modeling methods for this prior is the Gaussian process, an extension to the multivariate Gaussian distribution to an infinite-dimensional stochastic process (Brochu et al., 2010). While the Gaussian distribution is a distribution over point samples, the Gaussian process is a distribution over the function space. Thus, also the mean and the covariance are not numerical values but rather functions. This probability of distribution over the random functions $\mathbf{f}(\mathbf{x}) = (f(x_1), ..., f(x_N))$ is fully defined by the mean function $\boldsymbol{\mu} = (\mu(x_1), ..., \mu(x_N))$ and covariance matrix $\boldsymbol{K}_{ij} = k(x_i, x_j)$ and $\mathcal{N}$ being the multivariate normal distribution.

$$P(\mathbf{f} \mid \boldsymbol{X}) = \mathcal{N}(\mathbf{f} \mid \boldsymbol{\mu}, \boldsymbol{K}) \tag{4.21}$$

The goal is to get mean and uncertainty for a new sample given some prior belief and the previous samples. Let's use t as counter for our Bayesian optimization steps. For a given step t we have $\boldsymbol{x_t} = (x_1, ..., x_{N_t})$ samples and the corresponding random function $\mathbf{f_t}$ and the observations $\mathcal{D}_t$. Given a new sample $\boldsymbol{x}^*$ we can combine it with its random function $\mathbf{f}^*$ and the the previous observations $\mathcal{D}_1, ..., \mathcal{D}_t$ into a joint distribution.

$$\begin{bmatrix} \mathbf{f_1} \\ \vdots \\ \mathbf{f_t} \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu(x_1)} \\ \vdots \\ \boldsymbol{\mu(x_t)} \\ \boldsymbol{\mu(x^*)} \end{bmatrix}, \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \dots & k(\boldsymbol{x_1}, \boldsymbol{x_t}) & k(\boldsymbol{x_1}, \boldsymbol{x^*}) \\ \vdots & \ddots & \vdots & \vdots \\ k(\boldsymbol{x_t}, \boldsymbol{x_1}) & \dots & k(\boldsymbol{x_t}, \boldsymbol{x_t}) & k(\boldsymbol{x_t}, \boldsymbol{x^*}) \\ k(\boldsymbol{x^*}, \boldsymbol{x_1}) & \dots & k(\boldsymbol{x^*}, \boldsymbol{x_t}) & k(\boldsymbol{x^*}, \boldsymbol{x^*}) \end{bmatrix} \right) \tag{4.22}$$

We can compact the Equation 4.22 using the following identities:

$$\boldsymbol{K} = \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \dots & k(\boldsymbol{x_1}, \boldsymbol{x_t}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x_t}, \boldsymbol{x_1}) & \dots & k(\boldsymbol{x_t}, \boldsymbol{x_t}) \end{bmatrix}, \boldsymbol{k} = \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x^*}) \\ \vdots \\ k(\boldsymbol{x_t}, \boldsymbol{x^*}) \end{bmatrix} \tag{4.23}$$

Additionally we can simplify Equation 4.22 with the assumption that the prior mean $\boldsymbol{\mu(x)} = \mathbf{0}$ and $\boldsymbol{\mu(x^*)} = 0$

$$\begin{bmatrix} \mathbf{f} \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^T & k(\boldsymbol{x^*}, \boldsymbol{x^*}) \end{bmatrix} \right) \tag{4.24}$$

By applying the Sherman-Morris-Woodburry formula, as explained by Williams and Rasmussen (2006), we get:

$$P(f^* \mid \mathcal{D}_t, x^*) \sim \mathcal{N}\left(\mu_t(x^* \mid \mathcal{D}_t), {\sigma_t}^2(x^* \mid \mathcal{D}_t)\right) \tag{4.25}$$

with

$$\boxed{\begin{aligned} \mu_t(x \mid \mathcal{D}_t) &= \mathbf{k}^T \mathbf{K}^{-1} \mathbf{f} \\ {\sigma_t}^2(x^* \mid \mathcal{D}_t) &= k(\boldsymbol{x^*}, \boldsymbol{x^*}) - \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \end{aligned}} \tag{4.26}$$

These are now the predictions for the mean and the uncertainty for a new sample at x^*.

Noise

The above formulation only holds when we do not encounter any data noise like in this thesis. However, in practice often the evaluation of the function contains some noise which can be caused by some external influence to the measurement which can not be prevented. In this case the function evaluation $f(x)$ is perturbed by noise. As a result we write $f_{noisy}(x) = f_{noiseless}(x) + \epsilon$. Usually the assumption is posed that the noise is normal distributed with $\mathcal{N}\left(0, {\sigma_n}^2\right)$. So we can reformulate Equation 4.27 into:

$$\begin{bmatrix} \mathbf{f} \\ f^* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + {\sigma_n}^2 I & \mathbf{k} \\ \mathbf{k}^T & k(\boldsymbol{x^*}, \boldsymbol{x^*}) \end{bmatrix}\right) \tag{4.27}$$

with

$$\begin{aligned} \mu_t(x^* \mid \mathcal{D}_t) &= \mathbf{k}^T[\mathbf{K} + {\sigma_n}^2 I]^{-1}\mathbf{f} \\ {\sigma_t}^2(x^* \mid \mathcal{D}_t) &= k(\boldsymbol{x^*}, \boldsymbol{x^*}) - \mathbf{k}^T[\mathbf{K} + {\sigma_n}^2 I]^{-1}\mathbf{k} \end{aligned} \tag{4.28}$$

4.2.1 Kernels

As it becomes quite obvious, the covariance function is a very crucial part of the Gaussian process. The choice of the covariance function, often referred as kernel, has influence on many aspects of the drawn function samples for example smoothness (Brochu et al., 2010). So the kernel basically encodes the prior that the model has about the target function.

Squared Exponential Kernel

The probably most common kernel is the squared exponential kernel (SE), also called Gaussian Kernel or Radial Basis Function (RBF). However, this kernel is actually quite naive since all divergences of all features of the input x affect the covariance equally (Brochu et al., 2010).

$$k(x_i, x_j) = exp(-\frac{1}{2\theta^2}\|x_i - x_j\|^2) \tag{4.29}$$

The hyperparameter θ, known as length-scale parameter, controls the width of the range from which evaluated points are considered to make the prediction (Williams and Rasmussen, 2006). A small length-scale allows the random function to oscillate rather fast but also bears the risk of overfitting.

Matérn Kernel

A more advanced but also widely used kernel, especially for BO is the Matérn kernel. (Matérn, 2013) It adds a smoothness parameter ζ.

$$k(x_i, x_j) = \frac{1}{2^{\zeta-1}\Gamma(\zeta)}(2\sqrt{\zeta}\|x_i - x_j\|)^{\zeta} H_{\zeta}(2\sqrt{\zeta}\|x_i - x_j\|) \tag{4.30}$$

where $\Gamma(\cdot)$ is the Gamma function and $H_{\zeta}(\cdot)$ is the Bessel function (of order ζ). If we choose a very large ζ then the Matérn kernel acts like the squared exponential kernel since for $\zeta \to \infty$ the kernel is equal to the squared exponential. Where as for $\zeta = 0.5$ it es equal to the unsquared exponential kernel (Brochu et al., 2010).

White Kernel

The White Kernel or White Noise kernel is not to be considered as a real kernel. It is more like an modification to existing kernels. It primarily is used when multiple kernels are combined like in the python implementation of the Gaussian process by Sklearn[2] which is used for this work. In such a case the Withe Kernel is used to explain the noise of the signal as independently and identically normally-distributed. Thus this kernel (adaption) is only used for a noisy setting and thus is not used in this thesis[3].

$$k(x_i, x_j) = \begin{cases} \delta, & \text{if } x_i = x_j \\ 0, & \text{otherwise} \end{cases} \tag{4.31}$$

where δ is the noise level.

[2]Sklearn is a Python library with various machine learning algorithms for regression, classification, clustering and much more.

[3]Despite not being used in this thesis the implementation developed for this thesis does support the White Kernel.

5

Acquisition Functions in Bayesian Optimization

Acquisition functions play a key role in the Bayesian optimization. Their formulation guides the algorithm in its searching phase.
This chapter discusses the most common formulations for acquisition functions and presents possible algorithms for optimizing them.

5.1 Acquisition Functions

This section introduces the Upper Bound acquisition function, together with the state of the art acquisition functions Expected Improvement (EI) and Probability of Improvement (POI). Since the acquisition function responsible for guiding the search and proposing the new sample points it is a crucial part of the BO algorithm and depending on formulation of acquisition function the trade off between exploration and exploitation can be made. Exploration means that regions of the input space from which previously no samples where taken are being sampled now, so that unknown regions are explored. Exploitation on the other hand means that for a region that is already explored the optimal point is search for by taking relatively small steps.

5.1.1 Upper Bound

The most basic acquisition function is the *Upper Bound* function which is a linear combination of the mean $\mu(x)$ and the uncertainty $\sigma(x)$.

$$UB(x) = \mu(x) + \beta * \sigma(x) \tag{5.1}$$

In its most basic form the factor β is set to one. In this case the *Upper Bound* acquisition function is just the sum of mean and uncertainty. For minimization problems the equivalent would be the lower bound where the uncertainty is subtracted from the mean. This formulation is very intuitive as the area between the upper and the lower uncertainty bound is the area where target values are expected. As a result the upper bound maps each input to the optimal target value to be expected, even with low probability. So if such a point is evaluated it has for sure the highest chance to be better than the

highest previous sample. Since the mean prediction as well as the uncertainty, are taken into account it is possible that a point has higher acquisition value while having a lower mean prediction just because the uncertainty tells it that the estimator is really unsure about the mean prediction.

5.1.2 Probability Of Improvement

Probability of Improvement (POI) is another basic acquisition function suggested in the work of Kushner (1964). The POI is basic in its goal of finding the point which is most probable to have a function value that is higher than the highest past function evaluation, called *incumbent* x^+.

$$PI(x) = P(f(x) \geq f(x^+)) = \Phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right) \tag{5.2}$$

with $\Phi(x)$ being the cumulative distribution function (CDF)
As a result it does not really matter how big the mean prediction is of that point as long as it is most probably higher than the incumbent. Thus this acquisition function is clearly focused on exploration since even points that are infinitesimal greater than the incumbent are valid points to explore.
To balance the exploration an additional trade-off parameter $\xi \geq 0$ can be added to balance between exploration and exploitation. Kushner (1964) recommends to not use a constant value for that trade-off parameter but rather start with a fairly high value for ξ and then schedule a decay. The BO will then start with much exploration in the first steps whereas in the last steps it will focus more on the exploitation. The formulation with trade-off parameter looks as follows.

$$PI(x) = P(f(x) \geq f(x^+) + \xi) = \Phi\left(\frac{\mu(x) - f(x^+) - \xi}{\sigma(x)}\right) \tag{5.3}$$

5.1.3 Expected Improvement

The most established acquisition function is the *Expected Improvement* (EI). EI also considers the probability of improvement however it additionally accounts for the magnitude of improvement. The EI formulated by Močkus (1975b) looks the following:

$$EI(x) = (\mu(x) - f(x^+))\Phi(Z) + \sigma(x)\phi(Z) \tag{5.4}$$

where ϕ is the probability density function (PDF) and Φ is the CDF and Z is a variable defined as

$$Z = \frac{\mu(x) - f(x^+)}{\sigma(x)} \tag{5.5}$$

5.2 Optimization of Acquisition Function

After explaining the acquisition function formulation, this section discusses the schemes for optimizing them.
In contrast to Bayesian optimization, for the acquisition function optimization it is assumed that the function evaluation is cheap and fast. Thus taking more samples in favour of getting a more accurate solution is accepted, however accuracy is not crucial. Even if the optimizer does not find the real, global optima of the acquisition function, local optima will still yield much information which will give an improvement in the Bayesian optimization step. To begin with the very intuitive *Grid Search* algorithm is explained and its characteristics are highlighted. Since the Grid Search algorithm suffers greatly under the curse of dimensionality this thesis uses the *DIRECT* algorithm for the higher dimensions which is also discussed in thus section. In the end the Mixed Integer Programming is explained which can be used to optimize the acquisition function when it can formulated as system of objectives and constraints. As shown in (Fischetti and Jo, 2018) and (Weissteiner and Seuken, 2020) neural networks can also be represented by such a system and thus can be optimized using a Mixed Integer Programming solver.

5.2.1 Grid Search

Gird Search is the very basic approach where the input spaces is sectioned into a grid. This means for every dimension a certain amount of equidistant points are fixed on the input range. The result is a grid or a mesh defined by the points where the grid lines or mesh lines intersect. These are the samples that are being used. Like this the whole input space in sample with equidistant points. The resulting function values are stored in a list and the maximum is easily found by value comparison. The sample point which resulted in the maximum target value is then the input to the optima that has been searched for and which can be used as next sample point in the Bayesian optimization. Grid search ensures that the input space is equally explored, which means the hypercubes missed by the search are equal in size everywhere in the input space. However depending on the resolution, the number of points that are selected per dimension defines how granular the grid is and therefore how big the chances are to miss the actual optima. Since every additional dimension added to the system includes a similar amount of points, the number of samples are combined and grow exponentially with the amount of added dimensions. Hence this approach is only applicable in lower dimensions.

5.2.2 DIRECT

A popular global optimization algorithm proposed by Jones (2001a) is called DIRECT. This optimizer handles internally the trade-off between exploration and exploitation and allocates the effort accordingly to be able to find the optima globally. DIRECT evaluates in every iteration several search points based on a weighing between global and local search criteria (Jones, 2001a). One main advantage besides being global is that

DIRECT does not need any hyperparameters which need to be tuned. Where are advantages there are also disadvantages. In this case it is the curse of dimensionality which DIRECT suffers. Jones (2001a) explains this by the several problems that the algorithm tries to solve collectively, such as the support of continuous and integer variables or multi modal functions (Jones, 2001a).

The algorithm works similarly to a Pattern Search algorithm. First the n-dimensional search space is divides into multiple hyper-rectangles. This is also where the name originates from, DIRECT stands for **DI**viding **RECT**angles. Referring to the two dimensional case for explanation, the search space is split in, for example, 3 different rectangles. For each rectangle the center point is being sampled. Then, according to a special rule which is explained later, some of these rectangles are selected and split into smaller rectangles. From the new set of rectangles the new center points are sampled. This process repeats itself until a maximum number of evaluations or a maximum number if iterations is reached. Worth mentioning is that in every step all the rectangles are respected when deciding which one to select for further splitting, this means a rectangle that was not selected in the first step can still be selected in any of the subsequent steps.

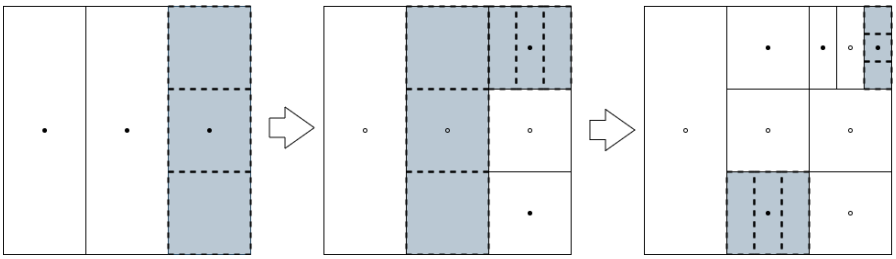


Figure 5.1: **Illustration of the DIRECT algorithm.** Illustration how the rectangles are selected and divided. In each step fist for all rectangles the center point is evaluated (if not already), these new evaluations are marked with a full circle. Next some rectangles, marked by the gray area, are selected and then divided into the new rectangles outlined by the dashed line.

The rule applied to select the rectangles first sorts all the rectangle by size. The size is defined by the distance from the center point to any of the vertices. For each size of rectangles the one with the largest function evaluation at it's center is a possible candidate. Actually selected rectangles lie on the convex hull when graphing the size against the function evaluation. As this is a simplified explanation, please refer to the original paper by Jones (2001a) for more details.

5.2.3 Mixed Integer Program (for NN)

Mixed integer programming is a super class of integer programming. These methods try to optimize a certain objective, a function for example, by applying some constraints. In

pure integer programming the variables which should be optimized are only allowed to have integer values. In mixed integer programming also non integer values are possible. The general concept of (mixed integer) programming is best explained visually. In the example in Figure 5.2 the goal is to maximize the number of products of type A. Without any constraints this would result in an infinite amount of items of type A, therefore it needs meaningful constraints. In real life these constraints are often given by the setting itself like by some relation between certain variables. An example from practice for such a constraint are storage capabilities. For example a company can have only 8 storage spaces and a product B uses two spaces whereas product A only one. Another constraint could be that the company always can not have more than 2 items more of type A than of type B. This setting coincides with the setting depicted in Figure 5.2.

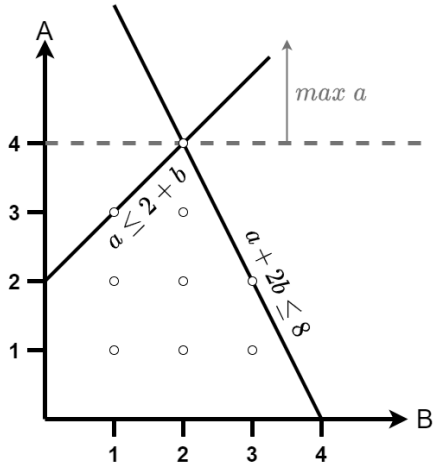


Figure 5.2: **Visual example of a MIP problem.** Example for a integer programming problem with solution $a = 4$, $b = 2$

For mixed integer programming there are advanced solvers available such as CPLEX (IBM, 2021).

It might be still not clear how MIP can help finding the optima of the acquisition function. As a reminder, the goal is to optimize a function which is represented as a neural network. This means each input is mapped to an output by inputting it to the network and running an forward pass. The result in the end is then the function value which should be optimized.

Let's first introduce the structure of a learned neural network, thus the network formulation of a certain function, so that later the MIP can be applied to it. A neural network consists of several layers of neurons (also known as nodes). The layers $l_0, ...l_L$ are ordered sequentially where l_0 is the *input layer* and l_L is the *output layer*. All layers in between are *hidden layers*. Each layer consists of $d^{(l)}$ nodes, which is the dimensionality of the layer. These nodes are denoted as $n_i^{(l)}$ where the superscript (l) indicates which layer the node is in and the subscript (i) is the index of the node inside this layer. The nodes of different layers are connected, in feed forward networks nodes of one layer are

always connected only to nodes of the next layer. Nodes which are in the same layer are never connected. In a fully connected network each node in one layer is connected to each node in the next layer. These connections between nodes are called edges and are weighted. For a connection of a node i in layer l, $(n_i^{(l)})$ to a node j in layer $l+1$, $(n_j^{(l+1)})$ the weight is denoted as $w_{ij}^{(l)}$. For a trained network these weights are fixed, since the goal of the training process is exactly to determine the weights. These weights can be interpreted as a indicator to how much influence does the output $o_i^{(l)}$ of node $n_i^{(l)}$ has in the calculation that node $n_j^{(l+1)}$ makes. Every node can be assigned a certain node function that it applies to its inputs and which results in its output. Usually NNs also have one additional node in each layer which has a constant output of 1, usually it remains at index 0. Thus we can say $o_0^{(l)} = 1$. This node is used to model the bias. In a fully connected network this means that each node has a constant input of 1 weighted with $w_{0j}^{(l)}$. For convenience the bias nodes are often omitted and the weight of its edges is introduced into each node as a bias term $b_i^{(l)}$ for node $n_i^{(l)}$. In a network with dropout it is possible that certain nodes are not active. For node $n_i^{(l)}$ we use $y_i^{(l)}$ as the binary variable to indicate if the node is active. Value of 1 means the node is active, value 0 means the node is inactive (dropped) (Cheng et al., 2017).

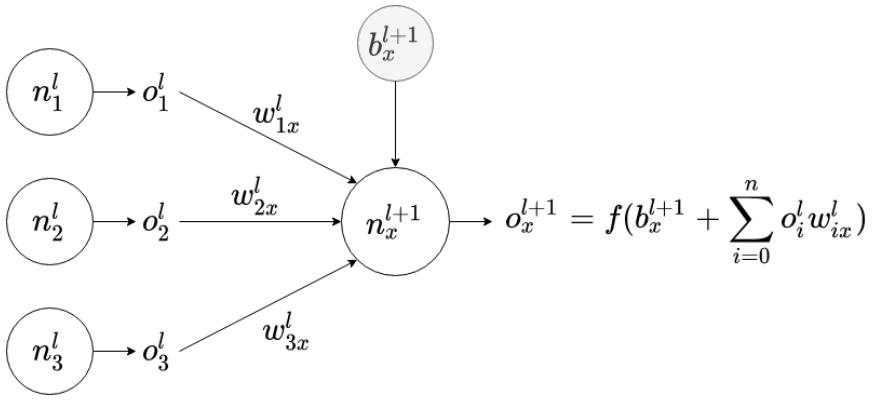


Figure 5.3: **Illustration of a neural network node.** Illustration of a NN-node of a fully connected network (without dropout) showing the different inputs and outputs with the proper notation used in the theoretical explanation of the MIP problem for NNs.

Here is a list of all different component that are included in the structure of the neural network and which are relevant for the mathematical notation of the MIP formulation.

$n_i^{(l)}$: is the node at index i in layer l

$o_i^{(l)}$: is the output of node $n_i^{(l)}$

$w_{ij}^{(l)}$: is the weight between node $n_i^{(l)}$ and $n_j^{(l+1)}$

$b_i^{(l)}$: is the bias in node $n_i^{(l)}$

$y_i^{(l)}$: indicates if node $n_i^{(l)}$ is active or not.

For a more convenient mathematical notation these variables can be combined into vectors and matrices which then can be used to formulate the relationships more concisely. Additionally the output vector is separated into a positive component $\boldsymbol{z^k}$ and negative component $\boldsymbol{s^k}$ which is used as a slack variable (Weissteiner and Seuken, 2020).

$\boldsymbol{o^k}$: are all the outputs of the nodes in the k-th layer listed in a vector.

$\boldsymbol{z^k}$: is the positive component of $\boldsymbol{o^k}$

$\boldsymbol{s^k}$: is the absolute value of the negative component of $\boldsymbol{o^k}$

$\boldsymbol{W^k}$: weight matrix of edges between the k-th and the (k-1)-th layer $\boldsymbol{W^k} = \begin{bmatrix} w_{11}^k & \dots & w_{1K}^k \\ \vdots & \ddots & \vdots \\ w_{K1}^k & \dots & w_{KK}^k \end{bmatrix}$

$\boldsymbol{b^k}$: are all biases for the nodes in the k-th layer listed in a vector

$\boldsymbol{y^k}$: are all binary variables $y_i^{(k)}$ for all nodes in the k-th layer listed as vector.

For the MIP formulation certain constraints are needed. Thus an assumption is made which results in the big-M constraint (Grossmann, 2002). It is assumed that:

Assumption 1. *Big-M Constraint For all $K \in K$ there exist a large enough constant $M \in \mathbb{R}_+$, such that $|((\boldsymbol{W^k})^\top \boldsymbol{o^k} + 1\boldsymbol{b^k})_j| \leq M$ for each entry j.*

From this Big-M constraint we can construct the first constraints for the MIP:

$$\boldsymbol{z^k} - \boldsymbol{s^k} = (\boldsymbol{W^{(k-1)}})^\top \boldsymbol{o^{(k-1)}} + \boldsymbol{b^k} \tag{5.6}$$

$$0 \leq \boldsymbol{z^k} \leq \boldsymbol{y^k} \cdot M \tag{5.7}$$

$$0 \leq \boldsymbol{s^k} \leq (1 - \boldsymbol{y^k}) \cdot M \tag{5.8}$$

A detailed proof can be found in (Weissteiner and Seuken, 2020).

Using the Big-M constraints the whole MIP problem formulation can be written down as:

$$
\begin{aligned}
&\operatorname{argmax}\{z^K\} \\
\text{s.t.}\quad & \\
& z^{k=0} = x \\
& \left.\begin{aligned}
\boldsymbol{z^k} - \boldsymbol{s^k} &= (\boldsymbol{W^{(k-1)}})^\top \boldsymbol{o^{(k-1)}} + \boldsymbol{b^k} \\
0 \leq \boldsymbol{z^k} &\leq \boldsymbol{y^k} \cdot M \\
0 \leq \boldsymbol{s^k} &\leq (1 - \boldsymbol{y^k}) \cdot M \\
y^k &\in \{0,1\}^{d_k}
\end{aligned}\right\} \quad k \in 1, ..., K-1 \\
& z^K = (\boldsymbol{W^{(k-1)}})^\top \boldsymbol{z^{(K-1)}} + \boldsymbol{b^K}
\end{aligned}
\tag{5.9}
$$

The last constraint which is specific for the last layer is different since for the output the slack variable $\boldsymbol{y^K}$ is always 1 as the output is never dropped. Thus following from Equation 5.8 $\boldsymbol{s^K}$ is strictly equal to 0. As a result the last equation can be written in a simplified form.

Mixed Integer Program for NOMU architecture

The formulation of in Equation 5.9 however only supports the standard feed forward structure with a sequence of fully connected layers. The NOMU model however does not have such a simple structure, therefore the MIP formula has to be adjusted before it can be applied to NOMU.
Luckily NOMU's architecture has a lot of similarities to a normal feed forward network and thus the formulation has to be adjusted only slightly. The NOMU model structure consists of two different fully connected networks which share the same input and have different outputs. So both network parts (main and side-net) can be considered separately for most of the constrains used for the MIP. Consequently instead of variables z, o , y and s there are now always two variables, one for the main-net and one for the side-net. The network part which a variable belongs to is indicated with a subscript m for the main-net and a subscript s for the side-net. As already stated, both network parts share the same input, thus:

$$z_m^0 = z_s^0 \tag{5.10}$$

The shared input however is not the only part where main and side-net are connected. There are also connections from the last hidden layer of the main-net to the output layer of the side-net. The weight for these connections are included in the weight matrix of the last layer for the side-net only. This weight transformation has therefore no effect on the main-net's constraints, only for the last layer constraint of the side-net. To be able to formulate the relation neatly the weight matrix W for the side-net can be spilt into the part which relate to the weight of the output of the last hidden layer of the main-net (W_{1s}) and the side-net (W_{2s}) respectively.

$$(\boldsymbol{W_s^{K_s}})^\top = [(\boldsymbol{W_{1s}^{K_s}})^\top, (\boldsymbol{W_{2s}^{K_s}})^\top] \tag{5.11}$$

The NOMU network architecture has two outputs, $z_m^{K_m}$ and $z_s^{K_s}$, which should ideally be used in the objective. For the application of BO the objective is to maximize the acquisition function. To keep the notation simple and general $\text{acq}(\mu, \sigma)$ is used as a general acquisition function which uses mean and uncertainty as inputs. The new objection thus looks as follows:

$$\text{argmax}\{\text{acq}(z_m^{K_m}, z_s^{K_s})\} \tag{5.12}$$

With this the adaption of the constraint formulation for the MIP to the NOMU architecture is completed and looks like the following.

$$\begin{aligned}
&\text{argmax}\{\text{acq}(z_m^{K_m}, z_s^{K_s})\} \\
&\text{s.t.} \\
&z_m^{k=0} = z_s^{k=0} = x \\
&\left.\begin{aligned}
\boldsymbol{z_m^{k_m}} - \boldsymbol{s_m^{k_m}} &= (\boldsymbol{W_m^{(k_m-1)}})^\top \boldsymbol{o_m^{(k_m-1)}} + \boldsymbol{b_m^{k_m}} \\
\boldsymbol{z_s^{k_s}} - \boldsymbol{s_s^{k_s}} &= (\boldsymbol{W_s^{(k_s-1)}})^\top \boldsymbol{o_s^{(k_s-1)}} + \boldsymbol{b_s^{k_s}} \\
0 \leq \boldsymbol{z_m^{k_m}} &\leq \boldsymbol{y_m^{k_m}} \cdot M_m \\
0 \leq \boldsymbol{z_s^{k_s}} &\leq \boldsymbol{y_s^{k_s}} \cdot M_s \\
0 \leq \boldsymbol{s_m^{k_m}} &\leq (1 - \boldsymbol{y_m^{k_m}}) \cdot M_m \\
0 \leq \boldsymbol{s_s^{k_s}} &\leq (1 - \boldsymbol{y_s^{k_s}}) \cdot M_s \\
y_m^{k_m} &\in \{0,1\}^{d_{k_m}} \\
y_s^{k_s} &\in \{0,1\}^{d_{k_s}}
\end{aligned}\right\} \begin{aligned} &\text{k}_m \in 1, ..., K_m - 1 \\ &\text{k}_s \in 1, ..., K_s - 1 \end{aligned} \\
&z_m^{K_m} = (\boldsymbol{W_m^{(K_m)}})^\top \boldsymbol{z_m^{(K_m-1)}} + \boldsymbol{b_m^{K_m}} \\
&(\boldsymbol{W_s^{K_s}})^\top = [(\boldsymbol{W_{1s}^{K_s}})^\top, (\boldsymbol{W_{2s}^{K_s}})^\top] \\
&z_s^{K_s} = (\boldsymbol{W_{1s}^{(K_s)}})^\top \boldsymbol{z_m^{(K_m-1)}} + (\boldsymbol{W_{2s}^{(K_s)}})^\top \boldsymbol{z_s^{(K_s-1)}} + \boldsymbol{b_s^{K_s}}
\end{aligned} \tag{5.13}$$

6

Experiments and Simulations

This chapter describes all of the experiments made for this thesis in a chronological order. However, before listing the experiments the main metrics that are used to compare the different estimators are explained followed by the discussion of some preparatory work that has been done to get some pre-experimental insights. For each subsequent experiments first the overall experimental setup is explained before highlighting the exact configuration in more detail. Next the results of the experiment are presented and discussed followed by the main findings that are summarized in the conclusions.

6.1 Metrics

6.1.1 Regret

The main metric that is used in this thesis to compare the different estimators is the so called *regret*. The regret describes the error between the currently optimal value sampled by the BO and the true optimal value. Therefore the regret as metric can only be calculated when the true optima is known. Intuitively one can think of the regret as the numerical value which can be improved to reach the true optima. As long as the true optima is not reached one can not be fully satisfied. Formally the regret is defined as follows:

Definition 3. ***Regret:*** *If $\mathcal{D}$ is the set of samples and $y^* = f(x^*)$ is the true optima to the target function f at input x^*, then*

$$\hat{y} = \max_{x \in \mathcal{D}} f(x)$$

is the optimal sample and the regret is

$$\|\hat{y} - y^*\|$$

When comparing methods a smaller regret is better than a large one since indicates that the found optima is closer to the true optima.
In the context of the BO the regret can be calculated after every step since with every

step a new sample is taken which might improve the value of the optimal sample $\hat{y}$. Note that in the Definition 3 the regret is based on the optimal sample within the whole sample set $\mathcal{D}$, so including the starting samples. As a result it is not possible for the regret value to increase after a new step since adding a new sample can only decrease the value of the optimal sample[1], $\hat{y}$. However it is possible that the regret value stays the same over multiple steps, when the new samples do not result in a larger function evaluation value. This often happens during the exploration phase of the BO algorithm.

Whenever only the regret after the last step is considered it will be referred to as *final regret*.

6.1.2 Ranking

To quantify the difference between the estimator methods they are ranked according to a scheme based on the final regret which incorporates not only the mean over all runs but also the confidence interval. Primarily the upper confidence bound of the 95% confidence interval is considered and the method which has the lowest is ranked at rank one. However it is possible that another method is assigned the same rank when its lower confidence bound is lower than the upper confidence bound of best one. All the other ranks are derived by counting the number of upper confidence bounds that are below the lower confidence bound of the method that should be ranked. As a consequence it is possible to have for example two first ranks and a second one but also it is possible to have two first ranks, no second one and a third one as depicted in Figure 6.1.

(a) Two first and one Second place

(b) Two first and one Third place

Figure 6.1: **Possible ranking orders.** Illustration of the ranking order showing visually how the methods are ranked. The dashed lines indicate upper and lower confidence bounds. In (b) the most right method is ranked third since both upper confidence bounds of the other methods are below the lower confidence bounds of itself. In (a) the same method is ranked second since only one upper bound of another method is below its own lower bound.

[1]This holds only when there is no data noise present. With data noise it is not guaranteed that evaluating the function f at the same point multiple times always yields the same function value.

6.2 Preparations and Exploration

Before starting with the experimental work, the four methods are tested in the context of Bayesian optimization. This explorative work has its main goal to test the implementations of the different method and their integration into the BO algorithm. However the data that results from these preparatory works can already be used to get some initial understanding on how the different methods behave and special characteristics can be identified.
To see the behaviour of the method for each step of the BO the predicted mean function estimates and the uncertainty bounds are plotted together with the acquisition function, its maximum and therefore the suggested next sample point. As a reference the true synthetic function is also plotted as well as the sample points used for the training of the estimator in this step. To be able to compare the methods the same starting configurations (same sets of initial samples) are used for all methods.

6.2.1 Insights

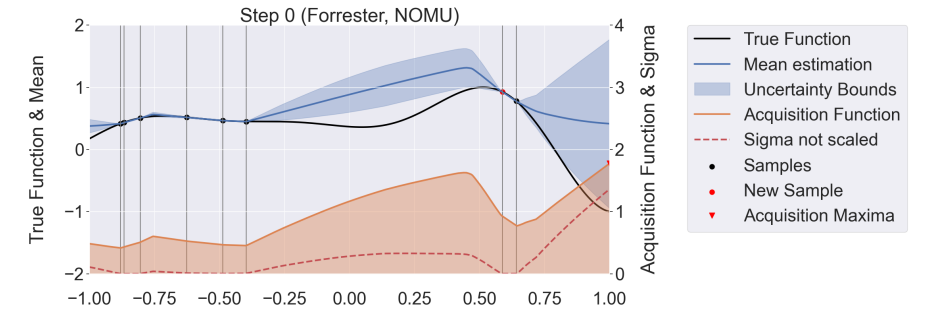
From this initial explorative work it becomes apparent that the different width methods produce uncertainty bounds with different magnitude (see Figure 6.2). *DeepEnsembles* for example, produces very slim uncertainty bounds meaning that the predicted σ itself has a very small absolute value. It should be clear that depending on the acquisition function the scale of the uncertainty has a large influence on the behaviour of the BO. When taking the *UpperBound* acquisition function, the small absolute uncertainty values will lead the BO algorithm to overvalue the mean prediction and always evaluate where the mean prediction is highest. However, if the uncertainty is too large the BO will only explore and will have no incentive to exploit promising regions. Thus finding a reasonable choice of the scaling of the uncertainty is key to have a reasonable trade-off between exploration and exploitation. Having such large differences in the uncertainty scale can make comparing the methods challenging since it is difficult to identify if existing performance differences are caused by the method itself or rather just by the uncertainty scale.

Mean Width Scaling

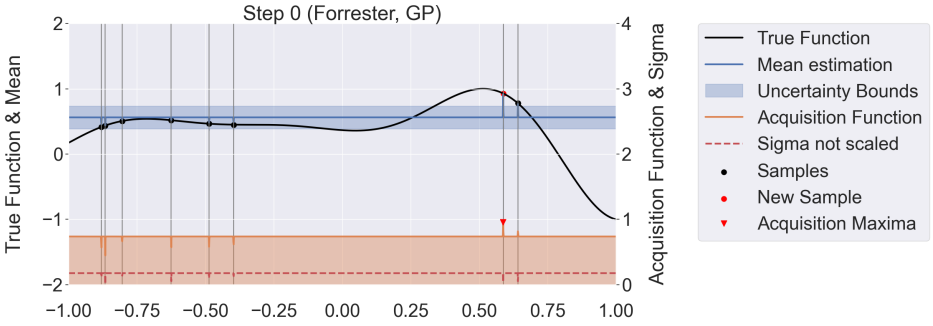
As a result for the experimental work of this thesis the different methods are adjusted so that they produce uncertainty predictions within the same scale. To achieve this an additional factor c for the predicted uncertainty σ_{pure} is introduced which in this thesis is referred to as *scaling factor*.

$$\tilde{\sigma} = c \cdot \sigma_{\text{pure}} \tag{6.1}$$

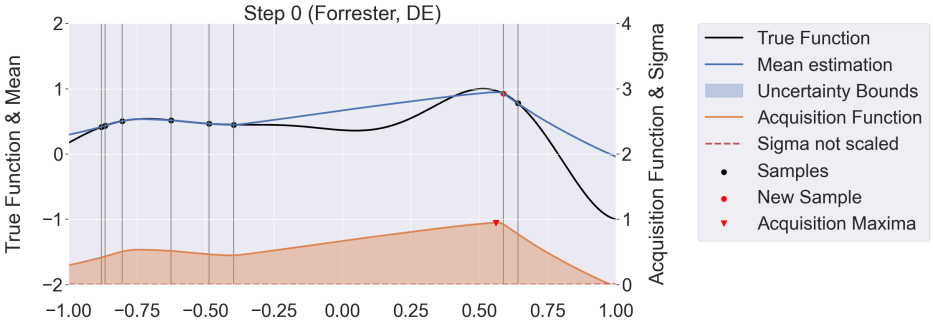
Having this additional factor makes it possible to re-scale the uncertainty prediction of the different methods, however, it is not clear which exact factors to use. As a result the



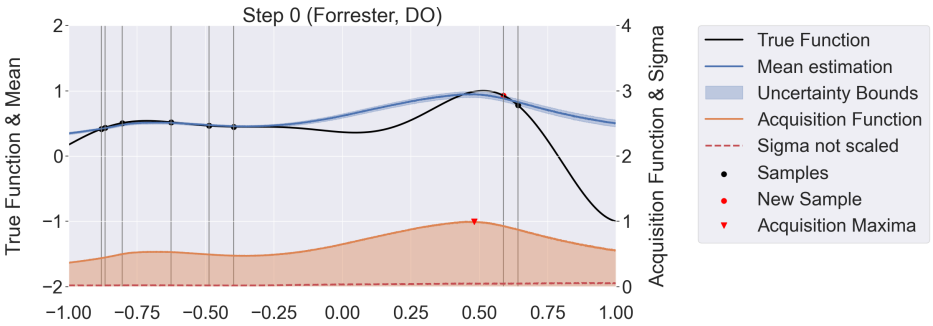
(a) NOMU



(b) Gaussian process (GP)



(c) Deep Ensembles (DE)



(d) MC Dropout (DO)

Figure 6.2: **Estimates in step 0 across the different methods on Forrester.** Mean and uncertainty estimates in the very first step of the BO for the instance Nr. 8 and the FORRESTER function.each method has different estimates and therefore proposes a different next sample (red triangle).

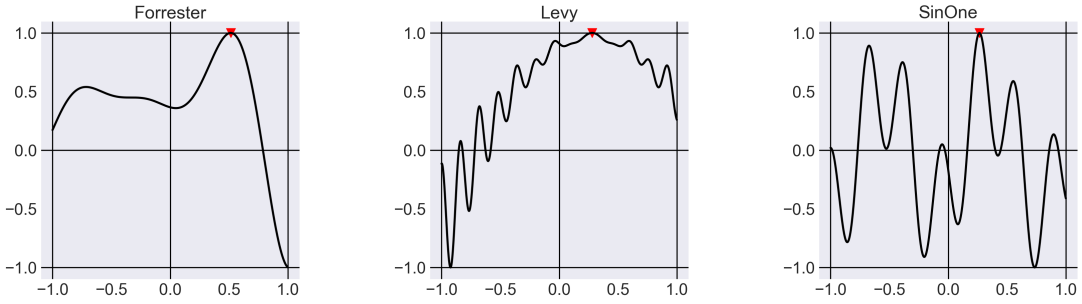
c is internally calculated based on the mean width of the uncertainty bounds. This mean width is simple to understand and it is calculated by taking samples of the uncertainty predictions over the whole input space. Then for each sample the width between the uncertainty bounds ($2 \cdot \hat{\sigma}$) is derived and the mean over all samples is taken. This approach of using the mean width is based on the same approach that is used in (Heiss et al., 2021). Also this mean width acts as good metric to intuitively see how the scales of the raw uncertainty estimates compare.
For the experimental work of this thesis different mean width budgets are used which are always applied to each of the methods. This ensures that intrinsic behaviour which favour small uncertainties are still respected, thus when comparing the methods the mean width that produced the best results is selected individually for every method. The scaling factor c is calculated such that the following condition is fulfilled:

$$\frac{1}{N} \sum_{i=1}^{N} 2 \cdot c \cdot \hat{\sigma}(x_i) \stackrel{!}{=} \text{budget} \tag{6.2}$$

Looking at the uncertainty predictions of the NOMU method in Figure 6.2a it stands out that the uncertainty grows very quickly towards the boundaries of the input space. This effect is not surprising when looking at the Equation 4.3. This effect, however, is not favourable in the context of BO since it generates a high tendency to evaluate the target function at the boundaries first, before exploring other regions. Why this is problematic is described in Section 4.1.1. In the same section also the solution to this problem, the additional activation is described which resulted from discussions with the authors of (Heiss et al., 2021) about insights gotten from this explorative work.

6.3 One Dimensional Synthetic Functions

With this first experiment the goal is to compare the different estimators, NOMU, Deep Ensemble, MC Dropout and the Gaussian Process on a very simple setting. To be able to calculate the regret as the metric (see Section 6.1.1) a setting with known true optima is required. The setting chosen, is a set of three different one dimensional synthetic functions which are regularly used as benchmarks (Eggensperger et al., 2013). The functions that are used are the Forrester, Levy and the SinOne (Sinus-One).



(a) Forrester (b) Levy (c) SinOne

Figure 6.3: **1D synthetic benchmark functions.** Set of one dimensional synthetic benchmark function scaled to have input and output range of $[-1, 1)$. All function has been modified to result in a maximization problem and therefore the true optima are marked as red triangles.

6.3.1 Experiment Setup

To be able to compare the results across the different synthetic functions, they are scaled to have equal input and output range of $[-1, 1]$. Consequently the minimum and maximum of function values are at $x_{\min} = -1.0$ and $x_{\max} = 1.0$. This allows to compare the regret value directly. By having functions with the same input space it is also reasonable to apply the same mean width budgets to all different synthetic functions. For this experiment mean width budgets of 0.1, 0.25, 0.5, 1.0 and 2.0 are tested.

Each BO instance is started with 8 samples which are drawn from an uniform distribution over the whole input space, so in the one dimensional case from $[-1, 1)$. Then the Bayesian optimization is run for 15 steps. After each step a new sample is added with a function evaluation at the location proposed by the BO algorithm. Since the configuration of the starting samples can matter for the performance of the Bayesian optimization, 30 instances with different starting samples are run so that conclusions can be drawn from mean and median performance of this set of 30 instances. All methods are run with the same collection of starting sample sets to ensure that for the presence of particular challenging starting configuration all methods have the same challenge.

Comparable Neural Network Complexities: For the NN-based estimators, namely NOMU, Deep Ensemble and MC Dropout it is ensured that their architectures have comparable network complexities which is represented by the number of weights. This is done because the predictive power strongly relates to the complexity of the model. The baseline is set by the NOMU method which uses three hidden layers with 1024 nodes each. The same structure of layers is used for both nets, the main and the side net. As a result the MC Dropout method for which also three hidden layers are used but which does not make use of a side net, more nodes are available per layer. Thus MC Dropout uses 1024 nodes in the first and third and 2048 nodes in the second hidden layer. For the Deep Ensemble method which uses a ensemble of 5 models, the number of nodes available per model has to be reduced. Hence 256, 1024 and 512 nodes are being used in the first, second an third layer respectively.

Additionally it is ensured that the regularization of the different methods is adjusted so that they are equal. This is necessary due to different implementation of the loss. Also for model architectures with dropout layers the dropped nodes do not contribute in the sum of outputs, thus the remaining nodes have more weight to them. This has to be considered in the regularization.

Noiseless Setting: Since this experiment is based on synthetic functions where the exact function value can be evaluated there is no data noise present and the experiment setup itself does not add artificial noise. By default most of the methods however are configured for noisy data. Therefore the NN-based methods are adjusted to not consider data noise.
For the Deep Ensemble method the adjustment is made according to Section 4.1.3 by using the Mean-Squared-Error (MSE) loss instead of the Negative Log-Likelyhood (NLL).
For the MC Dropout method no adjustments are made since due to the model sampling method of the MC Dropout model no real separated notion of data and model uncertainty can be identified.
The NOMU method the first sum (i) in equation 4.3 ensures that the model tries to predict as close as possible to the sample points. By selecting the parameters for the other two summands π_{sqr} and π_{exp} rather small, in particular smaller that 1 it can be ensured that the first summand stays prominent in the loss and therefore only small data uncertainty is allowed at the sample point itself.

Mean width calculation: To calculate the mean width, the uncertainty estimation is sampled on a grid. All uncertainty samples are multiplied by a factor of 2 since the goal is to get the mean width, they range between upper and lower uncertainty bound. Finally the mean of all these samples is taken.
The mean width is only calculated in the very first step of the BO algorithm. From it

the appropriate scaling factor c is derived and used in all subsequent steps. However, for each of the different instances a different mean width is possible.

6.3.2 Detailed Configuration

NOMU: The NOMU network model has the same structure for the main and the side net. Both of them have three hidden layers with 1024 nodes each. With this configuration the model has 4 205 570 weights to learn when there is one input. For all layers the ReLU activation is used in every node. For the additional activation a maximum uncertainty value of 2 and a minimum of 10^{-6} are used.

MC Dropout: The MC Dropout model does only use a simple feed forward network structure consisting of three hidden layers where the first and the last have 1024 nodes and the middle layer has 2048. For the case with one input this results in 4 200 449 weights. The Dropout probability is set to 0.5.

Deep Ensembles: For the Deep Ensemble the three hidden layers are used with 256, 1024 and 512 nodes respectively. The ensemble contains 5 networks. With one input, this results in 3 944 965 weight available for this DeepEnsemble estimator. ReLU is used as the activation functions and the regularization that is used is the L2-regularization with an value of 10^{-8}. Since the experiment is based on synthetic function the adaption for the noiseless case is applied (see section 4.1.3).

Gaussian Process: the Gaussian process is run with a RBF kernel with the default parametrization[2]. The kernel parameters can be retrained and adjusted in every step of the bayesian optimization.

Mean width calculation: The mean width is calculated on a grid with 2 000 grid point in each of the dimensions.

6.3.3 Results

There are multiple different aspects that can be analyzed and be discussed therefore the Results are sectioned into different parts, each focusing on one aspect.

Best Mean Width Budgets per Method Comparing the different mean width budgets it is very apparent that for all neural network based methods the regrets for different instances can vary a lot, hence the exact location of the starting samples can influence the performance. This is indicated by the large confidence interval around the regret mean as depicted in Figure 6.4 for the NOMU method and in Figure A.3 for all the other methods. This constitutes a problem for the quantification of the comparison since when the ranking rule described in Section 6.1.2 is applied, then all methods have

[2]For the implementation the RBF kernel from SKlearn was used with its default parameters

rank 1 since all confidence intervals overlap. Table 6.1 lists all ranks based on the mean and median and thus makes this effect very apparent.

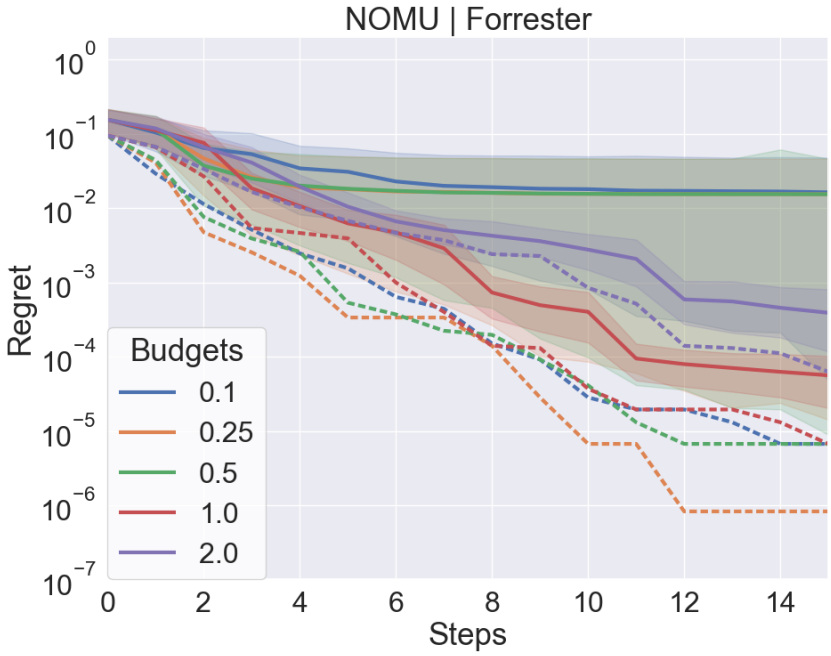


Figure 6.4: **Mean and median regret curve for NOMU on Forrester for different budgets.** For each of the mean width budgets in every step the mean regret over all 30 instances is calculated which results in the solid lines. The dashed lines show the median regret. For the mean of the regrets additionally the 95% confidence interval is depicted as the colored areas.

Looking at the individual mean width budgets for a specific functions and methods it is noticeable that the smaller budgets (0.1, 0.25, 0.5) tend to produce large confidence intervals more likely than the larger mean width budgets (1.0, 2.0). This can be observed nicely in Figure 6.4 and Figure A.3d.

Estimator Performance Comparison: Since there are multiple mean width budgets per method that are ranked first it is ambiguous which one to select for comparing the different methods. Based on the ranking scheme for the following comparisons the mean width budget which has the lowest upper confidence bound is considered as the "best" and used in Figures 6.5, A.6 and A.7. It is very apparent that the Gaussian process performs significantly better than all the other methods, for all synthetic functions at test. Overall the FORRESTER function has the largest differences in performance between the different estimators.
For all methods it applies that the median (indicated with dashed lines) always lies below

		Mean Width Budget									
		0.1		0.25		0.5		1.0		2.0	
f	estimator	μ	M	μ	M	μ	M	μ	M	μ	M
Forrester	NOMU	**1**	2	**1**	**1**	**1**	2	**1**	2	**1**	5
	GP	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	4	**1**
	DE	**1**	**1**	**1**	2	2	3	**1**	4	3	5
	DO	**1**	**1**	**1**	4	**1**	**1**	**1**	**1**	**1**	5
Levy	NOMU	**1**	3	**1**	**1**	**1**	**1**	**1**	5	**1**	4
	GP	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5
	DE	**1**	3	**1**	**1**	**1**	2	**1**	5	2	4
	DO	**1**	**1**	**1**	2	**1**	2	**1**	4	**1**	5
SinOne	NOMU	**1**	4	**1**	**1**	**1**	**1**	**1**	3	**1**	5
	GP	2	**1**	2	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	DE	**1**	4	**1**	2	**1**	**1**	**1**	5	**1**	3
	DO	**1**	5	**1**	**1**	**1**	2	**1**	4	**1**	3

Table 6.1: **Rank table for the different budgets per estimator in 1D.** For each estimator the final regrets for the different mean width budgets are ranked according to the ranking scheme which results in the ranks listed in the column μ. Using a normal ranking where lowest equals best also the medians are ranked in the column with header M. The same table is repeated for all three synthetic benchmark functions.

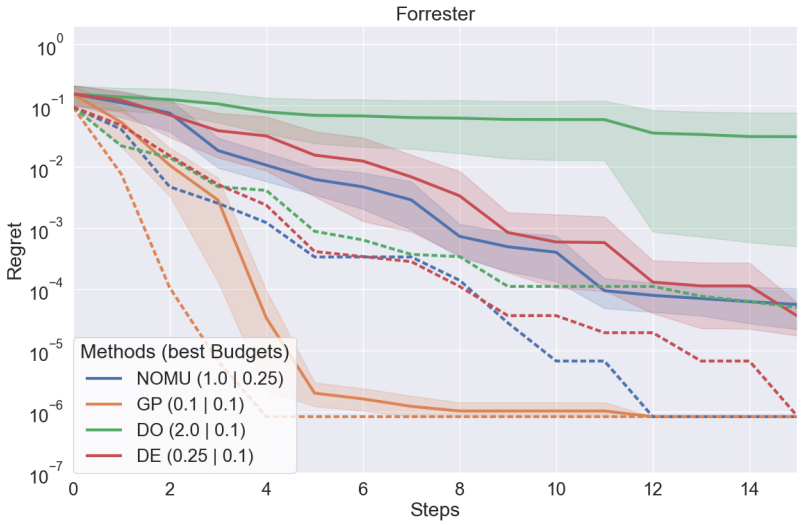


Figure 6.5: **Mean and Median regret curve for all estimators with their best budget on Forrester.** For each estimator the best mean width budget is selected for the mean and the median the respective regret curves are displayed. For the mean the budgets are selected by lowest upper confidence bound of the final regrets. The selected budgets are indicated in the legend after the method identifiers by the numbers in the bracket where the first one displays the best budget for the mean and the second one the best for the median.

the mean suggesting that way more than half of the instances perform better than the mean. The fact that the median lies even outside of the 95% confidence interval indicates that there are only very few outlier instances which drag the mean upwards to a bigger, thus worse regret. One outlier run with a large regret value has a large influence on the mean because all the other regrets are really small. Additionally, since the graphs are logarithmic scaled on the y-axis this effect is visually magnified.
Another observation that can be made is that the performance of the methods depends on the function that it tries to optimize. All of the different functions do have local optima however their structure still differs a lot. The Forrester for example has only one local optima which is much smaller than the global one. Also there are a lot of points around the global optima that are bigger than the local optima thus it is expected to be relatively easy for the methods to reach the global optima. The SinOne function however has many local optima which are peak-like. Also the SinOne function varies a lot in y when moving on x. Due to this large variability and the multiple local optima it is expected to be harder to find the global optima. The Levy function has also an interesting property that despite some variation in y and local optima it has overall parabola shape for which the optima coincides with the global optima. As a result it is possible to find the global optima even when the local oscillations are ignored and only the overall tendency of the curve is considered.

Bayesian Optimization Step Inspection: The regret curves already indicated that there are some instances which performed significantly worse than the rest. To further pin down which of the 30 instances performed bad and whether they are true outliers or whether the distribution just is skewed, the regrets after the last step are combined into a box plot. For each budget the mean and the 95% confidence interval is shown including the outliers. Additionally the outliers are numbered with the ID of the instance to be able to identify them. Figure 6.6 shows this box plot for NOMU on the Forrester function. It stands out that there is one clear outlier (Nr. 24) for the three small mean with budgets. Also the are some other outlier instances which however differ depending on the budget. Knowing which instances result in extremely bad regrets these instances can be inspected in more detail. Comparing the box plot with the regret curves it becomes clear that the outliers indicted in the box plot are the main cause for the large confidence interval for the regret mean as the interquartile range is much smaller.
When inspecting this instance with Nr. 24 for the NOMU method, the effect of the additional activation function becomes apparent now. In Figure 6.7 the red dashed line shows that the uncertainty estimate grows really quickly since there are no samples with x value higher than 0.25. However, because of the activation, the uncertainty estimation asymptotically approaches a maximum value of 2 as set in the configuration. As a result the acquisition function does not explode and is still in a reasonable range. Nevertheless the next sample that is proposed is the border point on the right. When looking at all the steps of the instances (see Figure A.21) it stands out that this specific configuration of starting samples almost lie on a straight line which makes the estimator believe that the function is almost constant. As a result the BO never explores the region around the

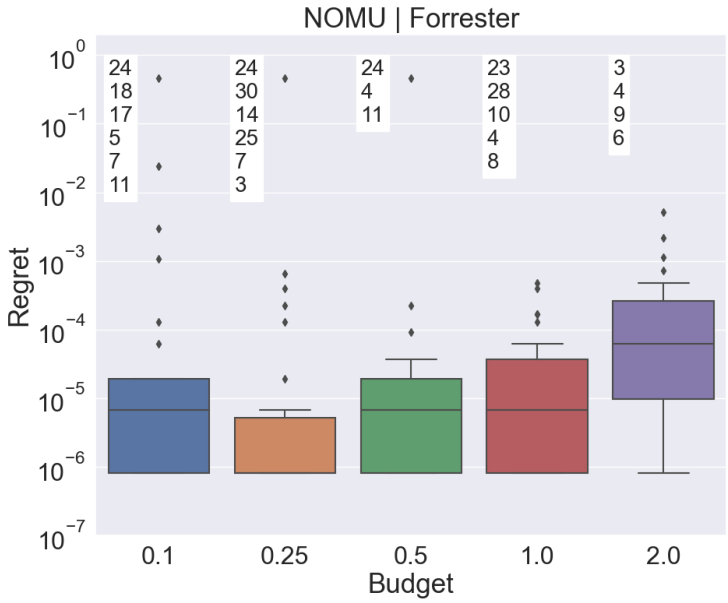


Figure 6.6: **Final regret distribution over the different instances, NOMU, Forrester.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret.

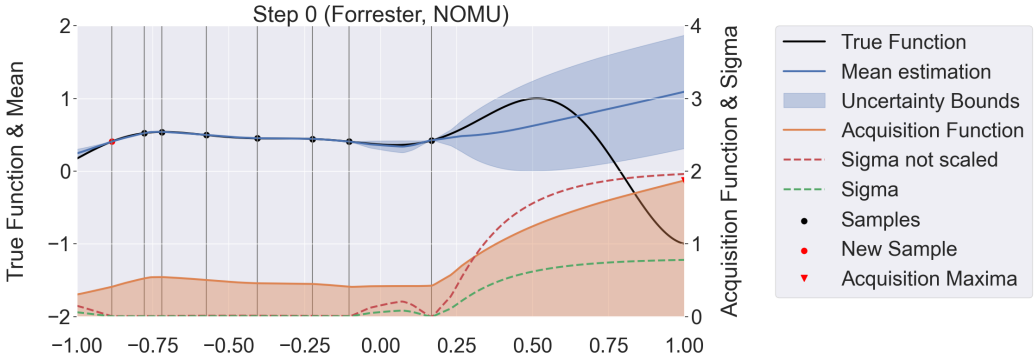


Figure 6.7: **First step NOMU on Forrester showing the NOMU-activation effect.** Estimations during the first step of NOMU on the Forrester function for instance Nr 24. with mean width budget 0.5. The red dashed line displays the uncertainty estimation after the activation was applied but before the uncertainty is scaled by the scaling factor c. The green dashed line shows the uncertainty estimate after the scaling with c.

true optima. This is also because the uncertainty is scaled down due to the mean width budget because in the first step the uncertainty at the right was quite big. However one could imagine that scaling up the uncertainty by weighing it more in the acquisition function or by scaling up the scaling factor c, eventually the optima of the acquisition function would lie in the desired region and the BO would evaluate the function there and then would be able to make more progress.

Another interesting observation can be made for the MC Dropout method. In addition to not exactly fitting through the samples, the uncertainty bounds lie like a tube around the mean prediction. Despite this rather naive behaviour the MC Dropout manages to produce quite good results. Especially for the Levy function the tube-like character is not hurting too much because the overall structure of the Levy is like a parabola with some smaller oscillations. Since the optima of the underling parabola and the Levy function coincide the optima can still be found and also the uncertainty tube captures most of the oscillation. Even for the SinOne function which has much larger oscillations the tube character of the uncertainty bound does not hurt the method too much.

Scaling Factor: As already mentioned in section 6.2 the Deep Ensemble estimator produces much smaller pure uncertainty estimates than the others. By pure uncertainty the prediction is meant before it is scaled by the scaling factor c which is derived from the mean width budget. To investigate the difference in scale of the pure uncertainty estimates in more detail Table A.1 lists all values for the scaling factor c for all methods-budget-function combinations. Comparing the scaling factor c for the different methods with a fixed budget directly reflects the magnitude of the pure uncertainty. The larger the scaling factor c the smaller was the original mean width. It stands out that the Deep Ensemble estimates uncertainties of a factor 100 to 1000 times smaller than the other estimators. It is also noticeable that the NOMU method irrespective of the function form, always produces uncertainty bounds with similar mean widths. For the Gaussian process however the scaling factor c varies a lot depending on the underlying function. For the relatively smooth Forrester function the uncertainty is rather small, smaller than for NOMU. For the SinOne function which changes with high frequency the pure uncertainty is much larger (indicated by a much smaller scaling factor c).

6.3.4 Conclusions

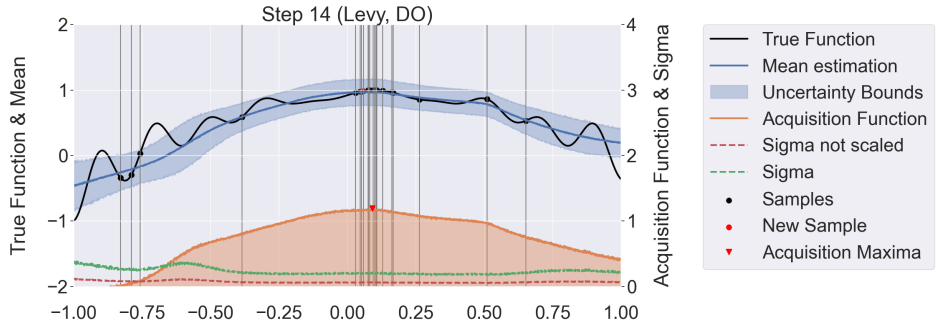
The first experiment has shown that in the low dimensional case the Gaussian process achieves best results after the last step as well as in early stages of the BO and is unbeaten by the NN-based methods.
For all NN-based methods it holds that the performance in terms of final regret, differs a lot depending on the mean width budget and the exact configuration of the starting samples. The later can also cause the algorithm to get stuck in local minima if the uncertainty is not weighted enough.
For the MC Dropout the observation was made that it does not fit the data points exactly despite the noiseless setting, nevertheless the BO still manages find the optima.

		Mean Width Budget									
f	estimator	0.1		0.25		0.5		1.0		2.0	
Forrester	NOMU	1.63×10^{-2}	3	1.54×10^{-2}	3	1.54×10^{-2}	3	$\mathbf{5.61 \times 10^{-5}}$	**1**	3.90×10^{-4}	2
	GP	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**	1.02×10^{-6}	2	2.85×10^{-6}	3
	DE	1.54×10^{-2}	4	$\mathbf{3.62 \times 10^{-5}}$	**1**	3.79×10^{-4}	2	1.56×10^{-3}	3	1.72×10^{-3}	3
	DO	$\mathbf{5.06 \times 10^{-2}}$	**1**	$\mathbf{4.76 \times 10^{-2}}$	**1**	$\mathbf{3.17 \times 10^{-2}}$	**1**	$\mathbf{4.71 \times 10^{-2}}$	**1**	$\mathbf{3.11 \times 10^{-2}}$	**1**
Levy	NOMU	6.51×10^{-3}	2	6.79×10^{-3}	2	4.76×10^{-3}	2	$\mathbf{9.25 \times 10^{-4}}$	**1**	$\mathbf{7.99 \times 10^{-4}}$	**1**
	GP	8.36×10^{-3}	4	$\mathbf{5.30 \times 10^{-6}}$	**1**	7.01×10^{-5}	2	8.28×10^{-4}	3	1.33×10^{-3}	3
	DE	9.62×10^{-3}	2	8.00×10^{-3}	2	$\mathbf{3.56 \times 10^{-3}}$	**1**	1.07×10^{-2}	2	2.63×10^{-2}	3
	DO	$\mathbf{4.31 \times 10^{-3}}$	**1**	$\mathbf{4.36 \times 10^{-3}}$	**1**	$\mathbf{4.37 \times 10^{-3}}$	**1**	$\mathbf{4.84 \times 10^{-3}}$	**1**	$\mathbf{8.55 \times 10^{-3}}$	**1**
SinOne	NOMU	$\mathbf{7.32 \times 10^{-2}}$	**1**	$\mathbf{6.95 \times 10^{-2}}$	**1**	$\mathbf{6.43 \times 10^{-2}}$	**1**	$\mathbf{6.72 \times 10^{-2}}$	**1**	$\mathbf{5.72 \times 10^{-2}}$	**1**
	GP	5.40×10^{-2}	3	5.40×10^{-2}	3	4.21×10^{-2}	2	2.28×10^{-2}	2	$\mathbf{7.27 \times 10^{-3}}$	**1**
	DE	$\mathbf{6.88 \times 10^{-2}}$	**1**	$\mathbf{6.15 \times 10^{-2}}$	**1**	$\mathbf{7.48 \times 10^{-2}}$	**1**	$\mathbf{5.71 \times 10^{-2}}$	**1**	$\mathbf{4.99 \times 10^{-2}}$	**1**
	DO	$\mathbf{8.08 \times 10^{-2}}$	**1**	$\mathbf{7.56 \times 10^{-2}}$	**1**	$\mathbf{8.05 \times 10^{-2}}$	**1**	$\mathbf{6.73 \times 10^{-2}}$	**1**	$\mathbf{8.10 \times 10^{-2}}$	**1**

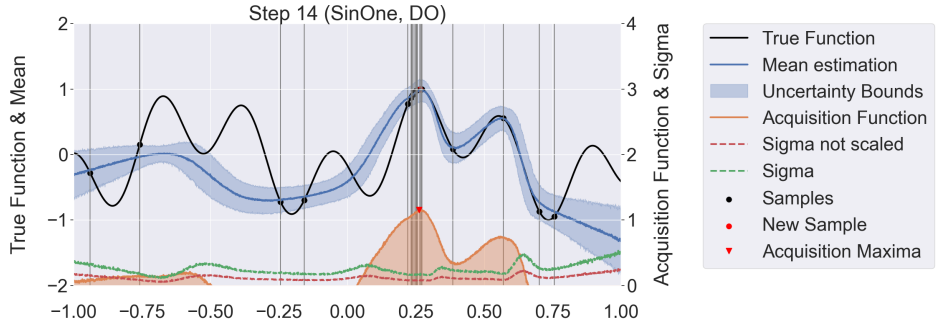
Table 6.2: **Full table of regret values.** This table lists all final mean regret value for all mean width budgets and all estimators. Next to the mean regret also the rank (according to the ranking scheme) is displayed and all first ranked budgets per method-function combination are highlighted in bold font.

		Mean Width Budget									
f	estimator	0.1		0.25		0.5		1.0		2.0	
Forrester	NOMU	0.62	[±0.16]	1.32	[±0.3]	2.74	[±0.69]	6.85	[±1.76]	10.98	[±2.6]
	GP	0.7	[±0.13]	1.6	[±0.35]	3.2	[±0.67]	6.61	[±1.4]	12.59	[±2.75]
	DE	731	[±350]	1255	[±489]	3773	[±2231]	8421	[±5738]	15501	[±8454]
	DO	0.95	[±0.09]	2.34	[±0.19]	4.67	[±0.39]	9.45	[±0.84]	18.91	[±1.5]
Levy	NOMU	0.6	[±0.18]	1.3	[±0.35]	2.5	[±0.65]	4.87	[±1.27]	9.46	[±2.26]
	GP	0.3	[±0.22]	0.84	[±0.57]	1.38	[±1.11]	3.36	[±2.28]	4.13	[±1.58]
	DE	215	[±95.34]	1003	[±1128]	1173	[±823]	2409	[±1932]	3371	[±1902]
	DO	0.8	[±0.07]	1.94	[±0.16]	3.96	[±0.33]	7.9	[±0.65]	15.81	[±1.36]
SinOne	NOMU	0.5	[±0.13]	1.71	[±0.57]	3.41	[±1.0]	7.14	[±2.6]	13.11	[±4.35]
	GP	0.09	[±0.01]	0.23	[±0.02]	0.46	[±0.05]	0.88	[±0.09]	1.71	[±0.19]
	DE	40.79	[±15.92]	91.46	[±29.55]	230	[±112]	353	[±120]	792	[±329]
	DO	0.52	[±0.06]	1.31	[±0.17]	2.66	[±0.31]	5.28	[±0.63]	10.68	[±1.42]

Table 6.3: **C values per method and mean width budget.** This table shows the mean scaling factor c derived from the estimated uncertainty and the give mean width budget per estimator method. These *c*-values have then been used throughout the whole process to scale the uncertainty. The first value describes the mean c-value over the 30 instances and in the brackets the 95% CI is indicated.



(a) MC Dropout performing on Levy



(b) MC Dropout performing on SinOne

Figure 6.8: **MC dropout estimates during last step on Levy and SinOne.** Mean and uncertainty estimations during the last step of the BO algorithm using the MC Dropout method for instance 18 (Levy) and 6 (SinOne), both with mean width budget of 0.5. Black dots and black vertical lines indicate the samples that were taken during the process showing that for both synthetic function the BO algorithm evaluates the function close to the true optima.

Another observation which is worth to keep in mind is that the uncertainty bounds for the MC Dropout lie tube-like around the mean prediction.
For the Deep Ensemble methods it was quantified that the uncertainty estimates are much smaller in magnitude compared to the other method.
Due to the large variance of the different instances it is difficult to quantify the difference between the methods. Consequently for subsequent experiments a measure has to be implemented which allows for more robust results.

6.4 One Dimensional Dynamic C Strategies

As seen from the results shown in Section 6.3 it is common that there are starting configurations which lead the BO to get stuck in certain regions, not being able to reach smaller regret values in later steps of the process. The consequences are quite large performance differences over different instances. This makes it challenging to make clear statements about the quantitative performance of the different methods.
The goal of the following experiment sequence is to asses different variations of methods for preventing instances to get stuck in a local optima so that the variance observed in the experiment in Section 6.3 becomes smaller allowing for a more robust quantification. In the detailed inspections of the instances in the previous experiment it was discovered that some starting configuration lead to very large uncertainty bounds at the very start of the BO run. Since the mean width calculation is only performed in the first step this leads to a scaling factor c that shrinks the uncertainty in every of the subsequent steps. Assuming that the mean prediction is rather robust throughout the process it makes sense to weight the uncertainty more to be able to escape a local optima and encourage more exploration. As a result an adjustment is made to the algorithm so that the scaling factor c can be doubled whenever it is noticed that the algorithm is stuck otherwise. The condition when the algorithm is considered to be stuck is when the algorithm proposes a point to sample which (in terms of the input) is close to an already samples point. Since in the later steps of the Bayesian optimization exploitation is favoured and thus should be allowed so that the process can converge in the promising region.
As a result, ε is defined as how close the proposed sample can be to already existing samples without triggering a doubling of the scaling factor c. To allow exploitation this ε has a decay strategy over time, similar to the trade-off parameter ξ for the Probability of Improvement acquisition function explained in Section 5.1.2.

In the following experiment two different strategies for the ε-decay are compared. The first approach is linearly decrease the ε with increasing amount of steps from an given initial ε_0 to a given final ε_n. For step i the ε_i is defined as follows.

Linear dynamic C:

$$\varepsilon_i = \varepsilon_0 + i \cdot \left(\frac{\varepsilon_n - \varepsilon_0}{n} \right) \tag{6.3}$$

where:

$\varepsilon_0 = \dfrac{l_{\text{interval}} \cdot h}{s_0}$

n is the number of steps for the Bayesian optimization

s_0 is the number of starting samples

l_{interval} is the input range per dimension

h is a parameter defining the initial ε in relation to the input range

Note that the final ε_n should be defined smaller than ε_0. When Grid Search is used as the acquisition function optimizer then it makes sense to choose ε_n to be equal to the grid-distance. This linear approach has as consequence a rather slow decrease in the ε, meaning that the BO algorithm is strongly forced to explore regions. Only at the very end it is allows to exploit already explored regions. Like this the chance of finding the correct region for the optima are high but it might happen that at the end there are no steps left for the algorithm to exploit this region. This means there might still be some improvement to be made. To counter this issue another strategy is experimented which allows a faster decrease in the ε. With an exponential decrease only the very first steps are forced to explore but rather quickly exploitation is allowed. Note that exploration is always possible, if the acquisition function suggests it.

Exponential dynamic C:

$$\varepsilon_i = \varepsilon_0 \cdot \left(\frac{\varepsilon_n}{\varepsilon_0}\right)^{i/n} \tag{6.4}$$

Dynamic C Padding: Another aspect that is taken into account with the Dynamic C adjustment is whether only in the last step it should be allowed to exploit with a minimal ε_n or whether the linear and exponential interpolation should finish before the last step so that for the last p steps it is possible to exploit with minimal ε. This number of last steps that are all using the same minimal ε_n is called *padding* in this work.
With padding the linear and exponential Dynamic C formulation are adjusted the following way:

Linear dynamic C with padding:

$$\varepsilon_i = \varepsilon_0 + i \cdot \left(\frac{\varepsilon_n - \varepsilon_0}{n - p}\right) \tag{6.5}$$

where:

$\varepsilon_0 = \dfrac{l_{\text{interval}} \cdot h}{s_0}$

p is the padding

Exponential dynamic C with padding:

$$\varepsilon_i = \varepsilon_0 \cdot \left(\frac{\varepsilon_n}{\varepsilon_0}\right)^{i/(n-p)} \tag{6.6}$$

Whenever the scaling factor c is doubled the acquisition function changes and thus has to be optimized again. However in terms of computational effort the optimization on the acquisition is usually considered rather irrelevant in comparison to target function evaluations. The newly found optima to the acquisition function is then the new proposed sample. If this is still too close to existing samples, then the c is doubled again.

The scaling factor c can be doubled until an appropriate sample is found or until a limit of doubling steps is reached.

As mentioned the scaling factor c is only calculated once during the very first step and hence it would also be possible to adjust the algorithm so that in each set a fresh scaling factor is calculated. This approach however is not examined in this thesis because the scaling factor, how it is interpreted in this work, should be a constant factor specific to the estimator and its characteristic and not to the instance itself. Ideally the scaling factor would be a hyperparameter with a fixed value, before running the algorithm.

6.4.1 Experiment Setup

After inspecting the first results of the linear interpolation experiment it was noticed that towards the end of the process most of the methods were able to achieve quite some improvements compared to the previous steps. As a result the padding mechanism as previously described was established. Intermediate experiments with different values between one and five for the padding showed that a padding value of 4 produced the best results for the FORRESTER function as a result for the DYNAMIC C strategies in the following experiment sequence will all work with the smallest ε_n in the steps 11 to 15. Also included in the analysis are the results of the previous experiment without dynamic C to identify the effect of this adjustment.

Overall the experimental setup is kept from the first experiment and applied to all the three different DYNAMIC C approaches, the linear decay without padding and the linear and exponential decay with padding.
Since the DYNAMIC C strategies can be viewed as the implementation of a different acquisition function with weighted uncertainty the same experimental setup is repeated with the EXPECTED IMPROVEMENT (Section 5.1.3) instead of the UPPER BOUND acquisition function.

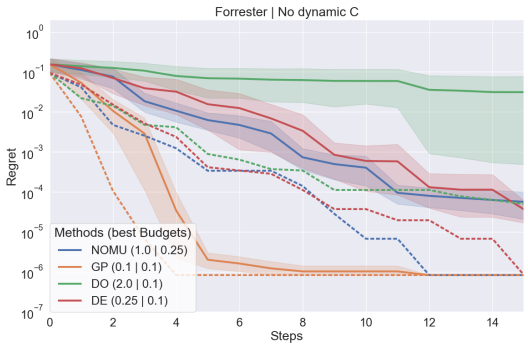
6.4.2 Detailed Configuration

As mentioned in the main configuration of the experiment, especially regarding the configuration of the estimator methods are the same as in the previous experiment (Section 6.3).

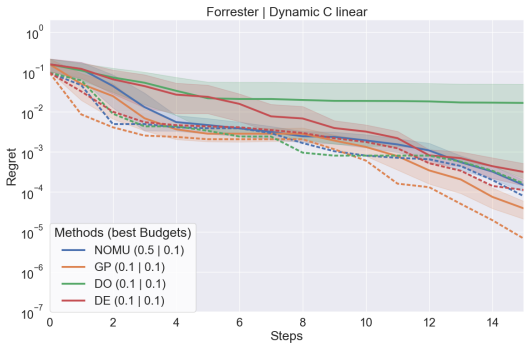
Dynamic C For the experiment variations which use the Dynamic C the parameter l_{interval} from Equation 6.3 and 6.4 is 2 since the input space for the function is scaled to $[-1, 1)$. The fraction parameter h is set to 0.25 so that even for equidistant distributed starting samples there is still room to find possible locations for a new sample that is not inside any of the ε-region to the other samples. As in the previous experiment 8 starting samples are used for each run, so $s_0 = 8$. For the optimization of the acquisition function still the GRID SEARCH is used with 2000 intervals per dimension which result in a interval width of 0.001 which is also set as the value for ε_n.

6.4.3 Results

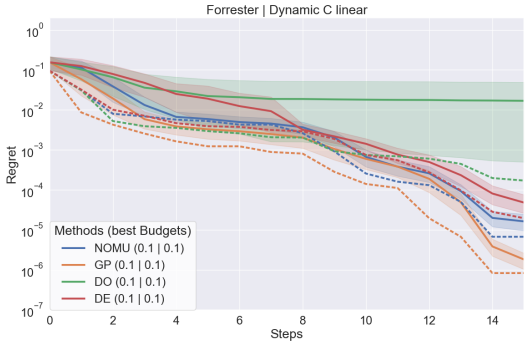
In Figure 6.9 it is noticeable that the progress of the Gaussian process is hindered when the dynamic C is applied. This can be explained by the fact that when dynamic C is utilized (especially with the linear variant without padding shown in Figure A.12b), the early steps of the BO is forced to sample points that are not close to previous samples. In the case that the algorithm finds the region of the true optima quickly, however it is not allowed exploit this region when using the dynamic C adjustment. Therefore the process if forced to explore and sample points which have a larger regret. Like this the regret cannot be improved until the process is allowed to exploit due to a smaller ε. It is noticeable that for the linear decay the padding improves the performance when considering the final regret in contrast to the case without padding. This effect makes sense since due to the padding the ε shrinks faster and reaches its minimum earlier and the BO has several steps available at the end to freely exploit the region around the assumed optima.



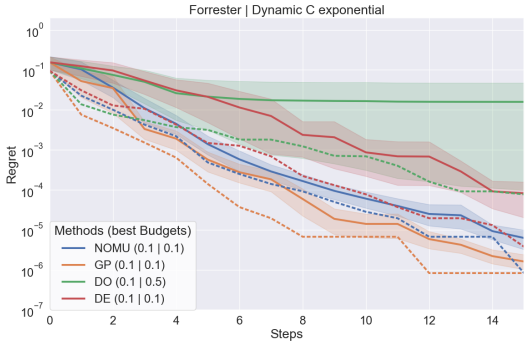
(a) without dynamic C



(b) exponential dynamic C (without padding)



(c) exponential dynamic C (with padding)



(d) exponential dynamic C (with padding)

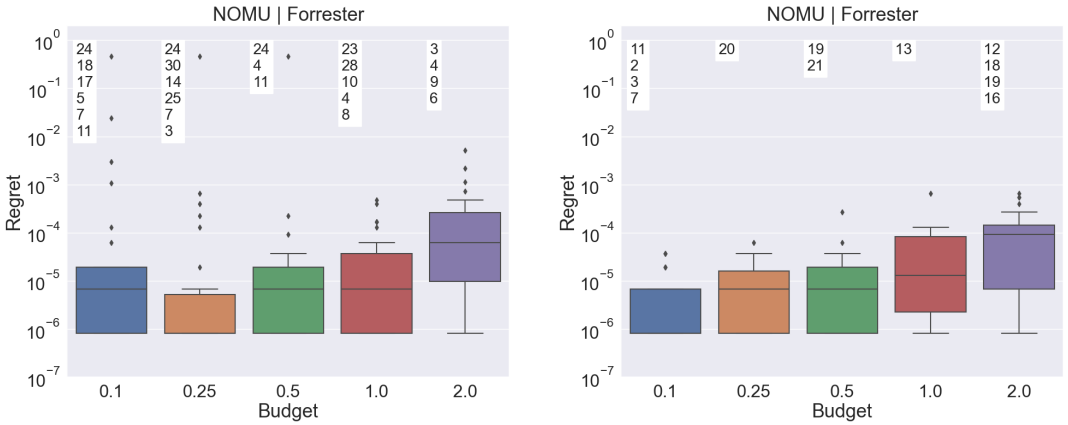
Figure 6.9: **Forrester - all methods with best budget, different Dynamic C approaches.** Each graph displays one regret curve per method. For each methods the budget which produces the smallest final regrets are selected, individually for mean and median.

Not only the linear but the exponential decay forces some more exploration than needed for the Gaussian process, however since the ε decays faster than in the linear case the scaling factor c is doubled less often (see Figure A.15) and the BO can start to exploit earlier. The final regret of the Gaussian process without dynamic C is never reached by any dynamic C adjustment. For the NOMU method however the exponential dynamic C was able to reduce the final regret mean from 5.61×10^{-5} to 6.24×10^{-6} in the case of the FORRESTER function (see Table 6.4). The median of the final regret stays the same however in the dynamic C case the median reaches its minimal regret only at the very last step whereas in the case without the dynamic C it reaches it faster. For small mean width budget the effect of the dynamic c is clearly visible. There are no longer instances that getting stuck at larger regrets like 10^{-2} for FORRESTER. With dynamic C after the last step the regret is smaller than 10^{-3} for all budgets. This reduction of outliers can also be observed when comparing the distribution of final regrets like in Figure 6.10. In this figure it can also be observed that overall the are much less outlier runs and the few remaining outliers are not as drastic anymore.

	Mean Width Budget									
variant	0.1		0.25		0.5		1.0		2.0	
no dc	$\mathbf{1.63 \times 10^{-2}}$	**1**	$\mathbf{1.54 \times 10^{-2}}$	**1**	$\mathbf{1.54 \times 10^{-2}}$	**1**	$\mathbf{5.61 \times 10^{-5}}$	**1**	$\mathbf{3.90 \times 10^{-4}}$	**1**
lin	$\mathbf{2.14 \times 10^{-4}}$	**1**	$\mathbf{1.51 \times 10^{-4}}$	**1**	$\mathbf{1.46 \times 10^{-4}}$	**1**	$\mathbf{2.55 \times 10^{-4}}$	**1**	$\mathbf{3.11 \times 10^{-4}}$	**1**
lin pad	$\mathbf{1.62 \times 10^{-5}}$	**1**	$\mathbf{4.58 \times 10^{-5}}$	**1**	$\mathbf{5.60 \times 10^{-5}}$	**1**	1.41×10^{-4}	2	4.14×10^{-4}	4
exp pad	$\mathbf{6.24 \times 10^{-6}}$	**1**	$\mathbf{1.07 \times 10^{-5}}$	**1**	$\mathbf{2.16 \times 10^{-5}}$	**1**	5.60×10^{-5}	2	1.35×10^{-4}	4

Table 6.4: **Final regrets for NOMU on Forrester for the different Dynamic C strategies.** This table lists the final regret means for the NOMU method for the different variants where "no dc" means that there in no Dynamic C however mean width scaling is still present. "lin." means that the ε decay linearly. "lin. pad." additionally has padding added to the strategy. The same goes for "exp. pad." where however the ε decays exponentially.

When looking at the distribution of the final regrets for each of the different variations, then one can see that compared to the version without dynamic C the ones which makes use of the dynamic C manage to prevent outliers which produce a really large regret. Figure 6.10a shows that without dynamic C the instance with "Nr. 24" produces really large regrets for the small mean width budgets. However in the cases where the dynamic C was applied there are no such large outliers. For the exponential dynamic C in Figure 6.10b, the instance "Nr. 24" is no longer an outlier at all. It is noticeable that for the FORRESTER the exponential dynamic C was able to shrink the interquartile range and also to lower the median for the mean width budget of 0.1 in addition to getting ride of the outliers which indicates that the instances got more robust.
In Figure A.21 in the Appendix the full run Nr. 24 without any Dynamic C is depicted. It is noticeable that the algorithm gets stuck in a local optima to the right. Figure 6.11

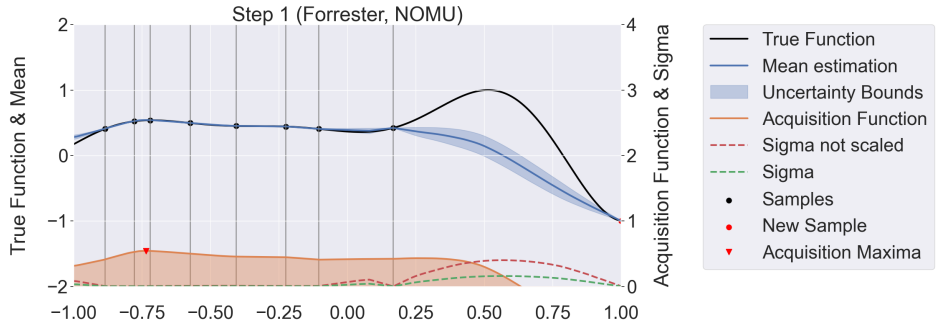


(a) without dynamic C (b) exponential dynamic C (with padding)

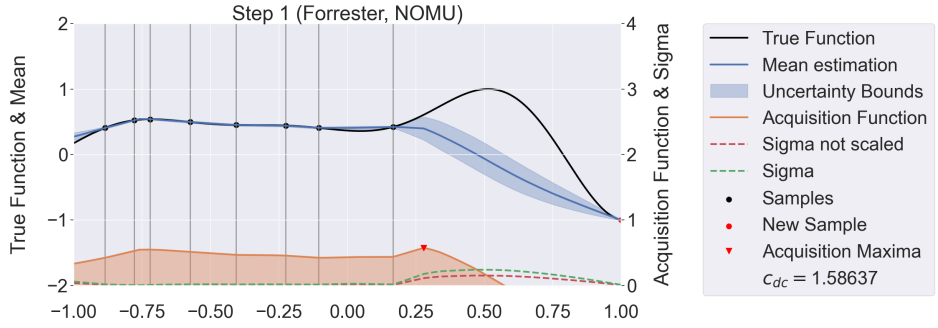
Figure 6.10: **Final regret distribution with and without Dynamic C for NOMU on Forrester.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret. The figure shows the result for when no Dynamic C is applied (same results as in Section 6.3). Figure (b) shows the result when Dynamic C is applied with exponential decaying ε and a padding of 4 steps.

represents the situation after the first step for each of the different experiment variations. Comparing them it becomes apparent that each variant proposes quite a different next sample indicated by the x-position of the red triangle. Without the dynamic C (Figure A.15a) the next sample is on the left side and all following samples will also be there. The other variants all propose a next sample further to the right closer to the true optima. The reason for this is that the dynamic C variations all scale up the uncertainty more than without the dynamic C . In that case the large uncertainty on the right gains enough strength to draw the attention towards this previously unexplored region. However, it is also important to mention that since neural network are learned during the process it is possible that different runs for the same set of starting points can lead to different mean and uncertainty predictions, which might also cause larger uncertainties due to a different mean width scaling, irrespective of the dynamic C approach.

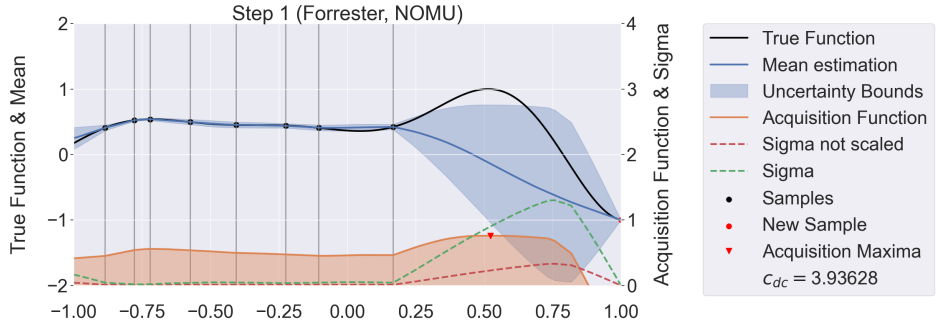
Whenever the dynamic c is used, either with linear or dynamic decay, then then smaller mean width budget tend to produce better results (Figure 6.9, A.13 and A.14). However when looking at the ranking many mean width budgets and dynamic C variants are jointly on rank one and thus no significant difference is quantified. Table 6.5 shows that for the Gaussian process without dynamic C always results in a better or equal rank for the Forrester function. However when applied to the SinOne function (Table A.3) then dynamic C performs better. Over all methods either the exponential decay version



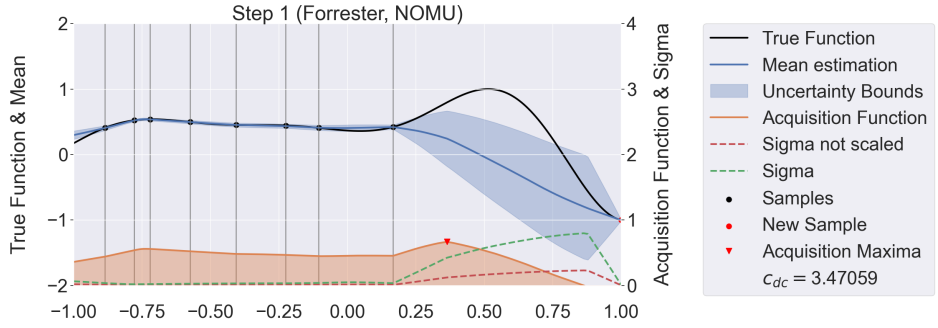
(a) without dynamic C



(b) linear dynamic C (without padding)



(c) linear dynamic C (with padding)



(d) exponential dynamic C (with padding)

Figure 6.11: **Differences of estimations for NOMU depending on Dynamic C variant.** Estimation results during the second step of the BO algorithm for the different variations of the Dynamic C for NOMU on Forrester. The scaling factor c is shown in the legend as c_{dc}.

of the dynamic C or no dynamic C rank first. However once again it is worth mentioning that one main reason why the version without dynamic C achieves good ranking is because the confidence intervall is rather large and thus chances of overlapping are also large. It is still less robust than without the dynamic C.
For completeness reasons it has to be mentioned that when comparing the Upper Bound acquisition function and the Expected Improvement on the same experimental setup for all dynamic C variations no real differences were identified, indicating that on the tested synthetic function the both acquisition function perform equally good (Table A.4).

6.4.4 Conclusions

Using the dynamic C method it is possible to use smaller mean width budgets without risking that the algorithm gets stuck in a local optima. Thus using the dynamic C adjustment produced more robust results. Another finding is also that using a ε in the dynamic C approach which only decays linearly slows down the progress drastically, therefore if using the dynamic C scaling using an exponential decay is highly recommended. Also it is recommended to use some padding for the decay rule so that there are multiple steps at the end of the process that can be exploited. Using small mean width budget yield as good or better results than large ones and since the Dynamic C mechanism is allowed to increase the value of the scaling factor c it is more robust to use them.
However it has to be said that in a simple setting like the one dimensional Forrester, the Gaussian process performs well without the dynamic C scaling and thus should not be tampered. The dynamic C adaption is mainly applicable for neural network approaches such as NOMU. However, even for them it does not decrease the regret by much as it mainly prevents outlier runs and thus produces more robust results.

Since no real difference between the Upper bound and the Expected Improvement acquisition function was determined, the UpperBound acquisition function will be further used for the remaining experiments.

			Mean Width Budget									
			0.1		0.25		0.5		1.0		2.0	
	f	variant	μ	M	μ	M	μ	M	μ	M	μ	M
Forrester	NOMU	no dc	**1**	3	**1**	**1**	**1**	3	2	3	4	13
		lin.	9	14	6	14	7	16	9	18	9	18
		lin. pad.	**1**	3	2	3	3	3	4	12	12	20
		exp. pad.	**1**	**1**	**1**	3	**1**	3	2	11	5	16
	GP	no dc	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	4	**1**
		lin. no pad.	13	13	13	17	13	17	16	19	16	20
		lin. pad.	4	**1**	5	**1**	4	**1**	10	13	13	13
		exp. pad.	4	**1**	**1**	**1**	5	**1**	5	**1**	10	13
	DE	no dc	**1**	**1**	**1**	3	3	7	**1**	12	11	20
		lin. no pad.	3	9	4	12	4	16	**1**	15	10	17
		lin. pad.	**1**	6	**1**	8	4	11	4	12	10	19
		exp. pad.	**1**	2	**1**	3	**1**	3	4	9	6	18
	DO	no dc	**1**	**1**	**1**	4	**1**	**1**	**1**	**1**	**1**	11
		lin. no pad.	**1**	8	**1**	11	**1**	15	**1**	17	**1**	20
		lin. pad.	**1**	9	**1**	16	**1**	11	**1**	19	**1**	18
		exp. pad.	**1**	6	**1**	7	**1**	5	**1**	10	**1**	11

Table 6.5: **Rank table for the different budgets per estimator and Dynamic C variant.** For each estimator and each Dynamic C variant the final regrets for the different mean width budgets are ranked according to the ranking scheme which results in the ranks indicated in the column μ. Using a normal ranking where lowest equals best also the medians are ranked in the column with header M. The whole table presents results for the FORRESTER function. "no dc" means that there in no Dynamic C however mean width scaling is still present. "lin." means that the ε decay linearly. "lin. pad." additionally has padding added to the strategy. The same goes for "exp. pad." where however the ε decays exponentially.

6.5 Two Dimensional Synthetic Functions

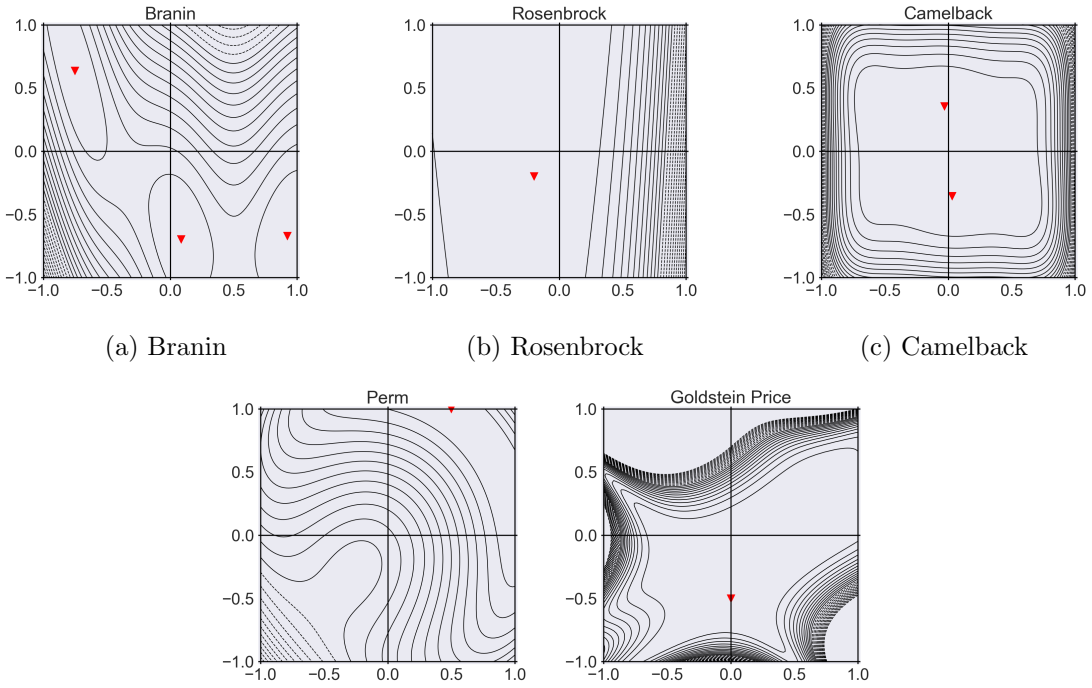
In the following experiment the behaviour of the different estimator methods in BO are tested when there is no longer just one input dimension but two. Due to the additional dimension there is much more space for exploration and the chances that the starting samples already reveal a lot of information about the target function is much smaller. In previous experiments it was discovered that some methods, for example the NOMU, sometimes have large uncertainty at the boundary which then are carried over into the acquisition function which makes the BO to mainly explore the boundaries of the input space. As a consequence the additional activation function was implemented into the NOMU method (see Section 4.1.1). In the one dimensional case there are only two boundary points but in the two dimensional case there already infinitely many[3]. Therefore the two dimensional setting is particularly interesting for inspecting the effect of the activation of the NOMU model but also to see how all the methods cope with two inputs.

6.5.1 Experiment Setup

In the experiment described in Section 6.4 the observation was made that the Gaussian Process performs well in low dimension such that the dynamic C scaling hinders its progress to a large extent. As a consequence for the following experiments there are always two versions of the Gaussian process used as benchmarks. First the default Gaussian Process scaled using the mean width scaling and affected by the dynamic c scaling. Second a pure version of the Gaussian process also based on the default configuration, however not scaled and therefore also not affected by the dynamic C scaling.

Like for the previous experiments synthetic functions are used so that there are analytical solution available for the function evaluations an therefore the exact regret can be calculated. Like for the one dimensional setting the functions that are used in the experiments are widely used in two dimensional benchmark functions. The set contains the Branin (Figure 6.12a) function which is the most used one for benchmarking optimization tasks, despite the fact that it has three optima. The other functions that are used are the Rosenbrock (Figure 6.12b), Camelback (Figure 6.12c), Perm (Figure 6.12d) and the Goldstein Price (Figure 6.12e) functions.
Most of the experiment setup is kept as it was in the one dimensional case. However due to the large increase of area that can be explored the number of steps that the BO can take is increased to 64. This amount of steps is large enough that the algorithm has more time to explore the true shape of the function before it starts exploiting the promising areas but the number is also small enough to not increase the run time too drastically. The algorithms need more time for exploration since the number of initial samples are kept at 8. It is interesting to see which method can cope with such sparse training info and can already in early stages produce good proposals for next samples.

[3]When the acquisition function optimizer is not constrained by a grid resolution.



(a) Branin (b) Rosenbrock (c) Camelback

(d) Perm (e) Goldstein Price

Figure 6.12: **2D synthetic benchmark functions.** Set of two dimensional synthetic benchmark functions scaled to have input and output range of $[-1, 1)^2$. All functions have been modified to result in a maximization problem and therefore the true optima are marked as red triangles.

With this methodology it is quite important for the methods to sample points which reveal much information so that the quality of the mean prediction can improve early on.

The conclusion that small mean width budget work best for almost every method and function combination from the previous experiment is used and thus there is no longer the need to test multiple budgets. Since the dynamic C mechanism is able to scale up the scaling factor c anyways an even smaller mean width budgets of 0.05 is used. Like this it can also be checked whether even smaller mean width budgets are worth it. If the results show that the dynamic C always scales up the scaling factor c at least once then the conclusion can be made that a mean width budget of 0.1 [4] would have produced the same results.

[4] A mean width budget of 0.1 is equivalent to the budget of 0.05 and doubling the resulting c-factor once.

6.5.2 Detailed Configuration

The configuration of the different network models is kept the same in terms of architecture, training epochs and regularization as in the previous experiments. All network models have to be adjusted to have two inputs in the input layer. The only other changes are made to the MC Dropout method. To use closer configuration to the original paper the dropout probability is lowered from 0.5 to 0.2. Also since for the MC Dropout method the evaluation of any points takes particularly long since it has to be repeated many times the number of network samples is reduced from 500 to 10. This aspect is relevant in the context of acquisition function optimization since there are a lot model evaluations made. For this two dimensional experiment still the Grid Search algorithm is used since it is still feasible, however the number of grid intervals per dimensions is reduced to 200, so that there are 40 000 grid points in total. Consequently the ε_n is increased from 0.001 to 0.01. To also increase the robustness of the means over the different instances the total number of instances is increased slightly up to 40.

6.5.3 Results

By looking at the regret plots (Figure 6.14) the first thing that stands out is that for four out of the five functions the Gaussian process which is not affected by the dynamic C performs worse than when the dynamic C and the mean width based scaling is applied. This is surprising since in the one dimensional experiments it was clearly visible that the dynamic C scaling slowed down the progress. This observation therefore has two logical explanations. First it is possible that the mean width based scaling itself does most of the improvements and then adding the dynamic C counteracts again. The other explanation is that increasing the dimensionality the setting got changed so much that the dynamic C scaling becomes an improvement. To further inspect graph which depicts how often the dynamic C was applied. However for all the functions for all the functions being tested, the scaling factor c was often doubled for the Gaussian process making it not possible to separate the effect of mean width scaling and dynamic C doubling (Figure 6.13).
Now focusing more on the NN-based methods it is apparent that for every function the situation differs, there is no method that clearly separates itself from the others throughout the whole experiment. However, when ranking the methods according to their final mean regret and the ranking rule, then the NOMU is always ranked first or second as seen in Table 6.6. The same Table also shows that the MC dropout method is always the worst neural network based method, only exception is the Camelback function where the Deep Ensemble performs the worst. For the Goldstein Price function the NOMU method manages to separate itself greatly from the other methods in terms of the median. When inspecting individual instances it can be observed that the NOMU method often explores the region around the optima to a larger extend whereas the other methods often exploit one particular point. In the case of the Goldstein Price, its flat structure often concentrates on a single point which may not exactly represent the global optima.

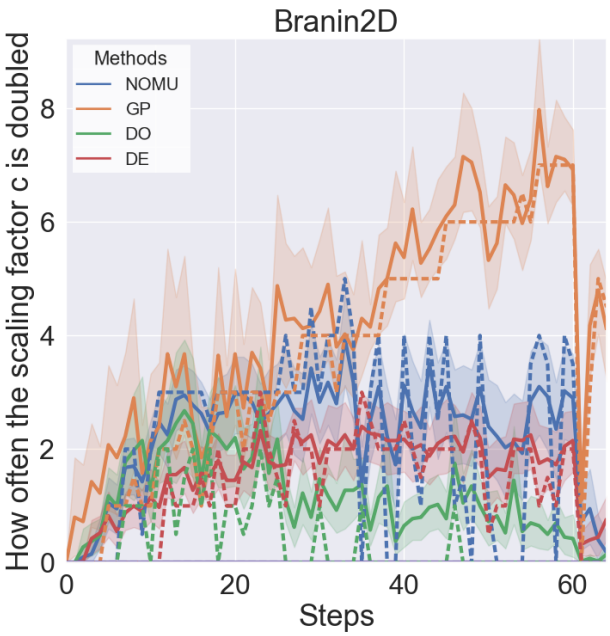
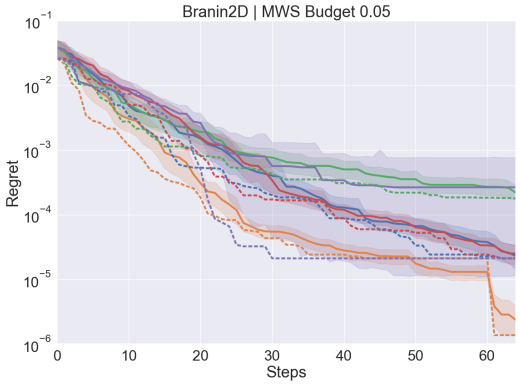
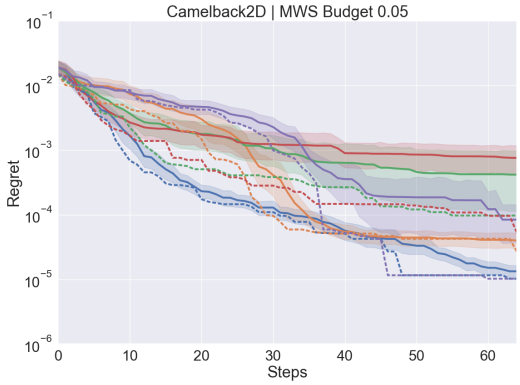


Figure 6.13: **Number of time Dynamic C is active.** This graph displays how often the scaling factor c was doubled due to the dynamic C adjustment. The solid line represents the mean over all the 40 instances and the colored area displays the 95% confidence interval.

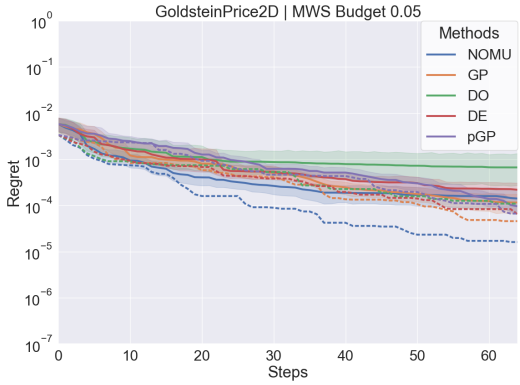
Further inspecting the instances where the BO samples the target function throughout the process it is noticeable that for the pure Gaussian process it can happen that it gets stuck in some local minimal. This was already indicated by the large variance of the runs. In the specific case of the Branin function it is very apparent that throughout the process almost exclusively boundary points were sampled by the pure Gaussian process (Figure 6.15e). Also a lot of points were samples very close together. The Gaussian process with dynamic C however behaves differently. It explores much more inside the input space and consequently finds a global optima for the same configuration of starting samples. When comparing the NN-based methods an interesting observation is that the MC Dropout method, as the only method, only finds one of the global optima and then only exploits this region. NOMU samples at two of the three optima and Deep Ensemble even at all three. The behaviour of the MC Dropout method can be explained by the tube-like shape of the uncertainty and the characteristics that it produces quite large uncertainty at the samples itself. Therefore the acquisition function value at a point is not greatly decreased by taking a sample there. For the other methods however taking a sample always reduces the uncertainty to a big extent which leads to a large change of the acquisition function.



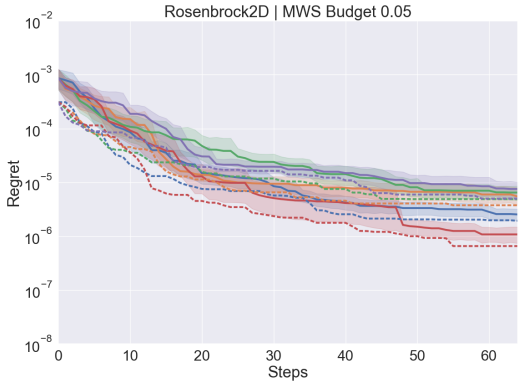
(a) Branin



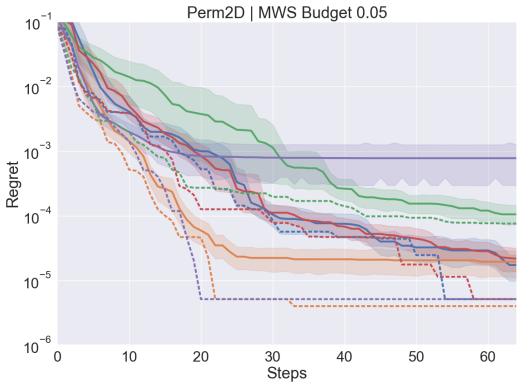
(b) Camelback



(c) Goldstein Price

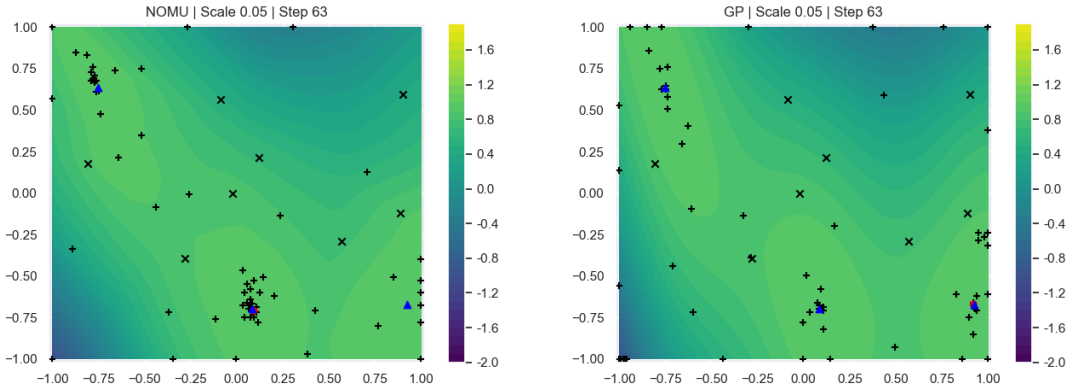


(d) Rosenbrock



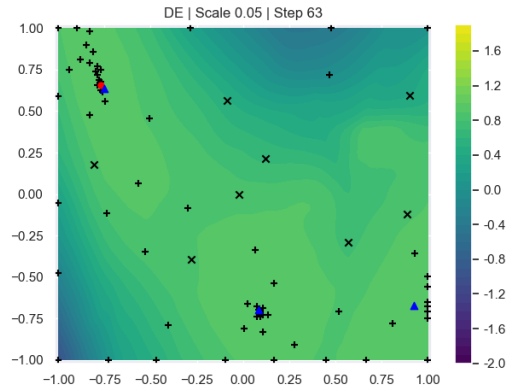
(e) Perm

Figure 6.14: **All method compared on their best budget for all functions.** This graphs show the mean and median regret curves for all function. For each method the curve for the budget with the smallest final regret is shown where mean and median do not have to have the same best budget. Legend from (c) applies to all graphs.

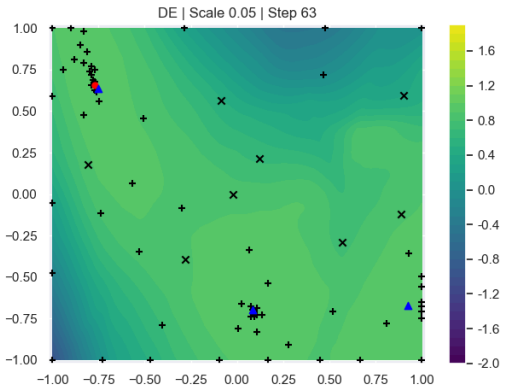


(a) NOMU

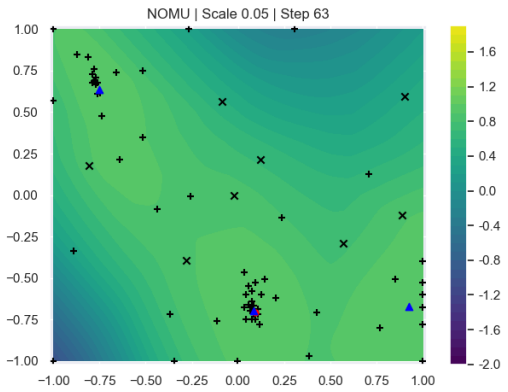
(b) Gaussian process



(c) MC Dropout



(d) Deep Ensemble



(e) pGP

Figure 6.15: **Sampled points for the different methods on Branin.** Each contour graph displays all samples taken during the BO process for the given method. The large crosses show the locations of the initial samples. The small plus sign show where the BO decided to sample. The blue triangle show where the true optima are located. The red circle indicates where the BO proposes to sample next. The contour lines indicate the estimated acquisition function.

	Estimator Methods									
	NOMU		GP		DE		DO		pGP	
function	μ	M	μ	M	μ	M	μ	M	μ	M
Branin	2	2	1	1	2	2	4	5	1	2
Rosenbrock	2	2	3	4	1	1	2	3	3	5
Camelback	1	1	2	3	5	4	3	4	2	1
Perm	1	2	1	1	1	2	4	5	5	2
Goldstein Price	1	1	1	2	2	3	3	5	1	4

Table 6.6: **Mean and median ranks of the methods per function.** For each function the final regrets of the different estimators are ranked. The ranks based on the mean are listed in the column μ. The ranks for the median are listed in the column with header M.

6.5.4 Conclusions

The NOMU method produces top two ranked final regrets for all the two dimensional synthetic functions tested indicating that it performs robustly well irrespective of the underlying functions. This stands in contrast to the Deep Ensemble and the MC Dropout method both of which depend a lot on the underlying function and thus end up performing worst for at least one of the functions at test. Especially the MC Dropout performs rather bad on the two dimensional setting.
For the Gaussian process when used with the default configuration it can very well happen that the BO gets stuck in a local optima when no dynamic C is applied to it. Therefore when interested in the overall mean then using the Dynamic C adjustment is beneficial even for the Gaussian process.
For functions that have multiple optima approaches like the Deep Ensemble and the NOMU are able to identify multiple of the optima whereas the MC Dropout methods only finds one.

6.6 Multi-Dimensional Synthetic Functions

After exploring the two dimensional setting the logical next step is to explore even higher dimensions. Thus, in this section the experiment is based on the five dimensional input space. Like in the two dimensional setting the main goal it so observe the behaviour of the methods in higher dimensions to identify key difference between the method when the input dimensions are further increased.

6.6.1 Experiment Setup

Different Mean Width Budgets: In the previous experiment sequence the mean width budget that was used, was 0.05 which is quite small. The reason for this was that together with the dynamic C scaling the uncertainty scaling can still be increased if needed to get out of a local optima. Using such a small mean width budget to begin with however means that most of the Upper Bound acquisition function will depend on the mean prediction and the uncertainty will have almost no importance in the decision where to evaluate the function next. In this thesis however the goal is not only to compare the quality of the mean prediction the subsequent use of it in the Bayesian optimization another goal is to also assess the quality of the uncertainty prediction and its influence in the Bayesian optimization. Therefore in addition to running the experiment as before with a mean width budget 0.05 it is also run with a budget of 0.5 with the intention to identify how the methods performance is influenced by the scaling of the uncertainty.

Mean Width Calculation: With increasing dimensions a grid based sampling for the mean width calculation is no longer feasible since with a grid of reasonable resolution the number of samples would just grow to fast. As a result another sampling method is implemented. Instead of using grid based samples the Monte Carlo approach is used which generates randome samples over the whole input space. Given these samples which return the magnitude of the uncertainty which also represent half of the width between the uncertainty bounds, the mean width can easily be calculated for the given set of samples. Using this Monte Carlo approach it is possible to specify exactly how many samples to take which allows to better control the run-time of the whole algorithm. However it must be said that using the Monte Carlo approach it highly depends on the number of samples how accurate the mean width calculation is. That being said, in this case the mean width calculation is not highly critical as it is just used to get a rough estimator for the scaling factor c of the uncertainty and since there is the dynamic C approach also active it can correct any underestimation of the scaling factor c.

DIRECT optimizer: Similar to the mean width calculation the acquisition function optimization has to be changed since using the Grid Search would need an enormous amount of evaluations[5] or a very large grid resolution where the first requires a lot of

[5] With the same grid resolution with 200 intervals per dimensions as in the two dimensional setting the Grid Search would require 320 000 000 000 evaluations per acquisition function evaluation.

computational resources and the later can no longer ensure that a reasonable global optima is found. As a result the DIRECT optimization scheme is used which actively samples new points to find out where to sample next. Therefore, this method requires much less acquisition function evaluations and thus Gaussian process or network predictions (Section 5.2.2).

Random Search Benchmark: Since in five dimensions the input space is so large and very few points are sampled chances are high that the estimator models are not able to make reasonable predictions and thus proposed samples resulting from the BO reveal much information. So to assess whether the BO makes reasonable process the Random Search algorithm is used as a benchmark. Random Search samples point at random, eventually it will sample a point that evaluates to a higher function value and thus makes progress and the regret decreases. Whenever a method performs equal or worse than Random Search it would be beneficial to just sample the points randomly. This can happen for example for optimization schemes which eventually get stuck in a local optima.

6.6.2 Detailed Configuration

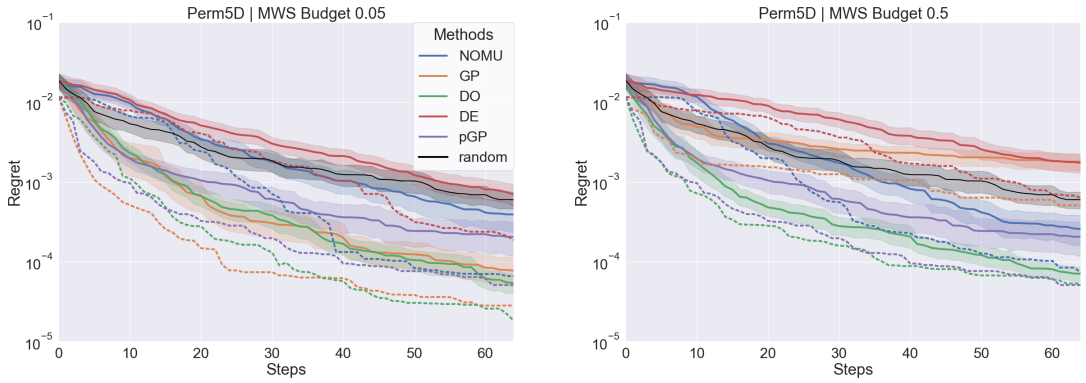
DIRECT configuration The DIRECT algorithm can be configured with some limits for the number of evaluations or iterations. For this experiment the fallback limits of 20 000 evaluations or 6 000 iterations are used.
Monte Carlo Mean Width Sampling For the calculation of the mean width of the uncertainty bounds the Monte Carlo Sampling is used with20 000 samples over which the mean is calculated.

6.6.3 Results

Comparing the results from the experiment with 0.05 mean width budget with the results using 0.5 as the mean width budget, then it is noticeable that in the later case most methods perform worse. Sometimes, depending on the underlying function, one of the methods produces drastically worse data than with a mean width of 0.05. This applies to the Deep Ensemble method, the MC Dropout as well as the Gaussian process. The only exception is the NOMU method for which the regret slightly increases for all underling function but never really drastically.
Looking at Figure 6.16 it is noticeable that for the Perm function all the methods produce results within the same regret magnitude. Also it stands out that even the Random Search produced results which are similar. Consequently it seems that the Perm function is not really solved by the methods or the function is so flat that there are so many points close to the optima so that randomly selecting the samples it just as good as following a certain strategies. As a result the Perm function is excluded from the further analysis of the results.

	NOMU		GP		DE		DO		pGP	random
mws budget	0.05	0.5	0.05	0.5	0.05	0.5	0.05	0.5		
GFunction5D	2	2	5	9	5	5	**1**	2	5	10
Levy5D	**1**	**1**	8	4	3	4	6	8	6	10
(Perm5D)	(4)	(4)	(**1**)	(9)	(6)	(9)	(**1**)	(**1**)	(3)	(7)
Rosenbrock5D	**1**	2	2	5	**1**	9	2	5	5	9

Table 6.7: **Ranks of the estimators per 5D function.** This table lists all the ranks of the different method for one function in a row. The regrets for the mean width budgets 0.05 and 0.5 are combined into the same ranking. The pure Gaussian process and the RANDOM SEARCH do not depend on the mean with therefore ranks from 1 to 10 are possible. As the PERM function is excluded from further analysis its values are put into brackets.



(a) mean width budget 0.05

(b) mean width budget 0.5

Figure 6.16: **Mean and median regret curve - Perm.** This graphs display the mean (solid lines) and median (dashed lines) final regret curve including confidence interval (colored area) for the different estimators. In (a) the results for a mean width budget of 0.05 are displayed and in (b) the ones for a budget of 0.5. Both graphs depict the situation for the Perm function

When comparing the remaining three methods, G-FUNCTION, LEVY and ROSENBROCK then one can notice that similarly to the two dimensional case the NOMU method consistently produces good results being ranked first twice. Only for the G-FUNCTION for which all of the methods produce a rather large regret, the MC Dropout method achieves better results. Looking at Table 6.7 shows the effect that for all methods the variant with 0.05 mean width got a better ranking. Only exception is the Gaussian process with dynamic C for the LEVY function.

6.6.4 Conclusions

This experiment showed that similar to the two dimensional case with five input dimensions the NOMU method is able to produce among others the lowest regret as long the the BO approach in general is superior to the random search. The results also consolidate the findings of previous experiment showing that the NOMU method is more robust on the changing of the mean width budget and also generalizes best for different underlying true functions.

7

Conclusions

This thesis examines the state-of-the-art model uncertainty predicting neural network estimators in the context of Bayesian optimization tasks. For the setting where the target function has one input dimension the results of this work show that the Gaussian process achieves outperforms all neural network based estimators when each method uses its best mean width budget. However for all of the estimators it was noticed that depending on the set of starting samples the Bayesian optimization algorithm can get stuck in local optima. With the goal of making the process more robust to the choice of the starting sample sets, the calibration technique of Dynamic C was introduced. This mechanism checks if the Bayesian optimization tries to exploit a region by determining if the proposed sample lies within an ε-region around already observed samples. In this case the algorithm is forced to explore another region. A dedicated experiment compared a linear and a exponential decay strategy for the ε-region and showed that the exponential decay performs better.

Further experiments explored the behaviour of the different estimators when applied to Bayesian optimization tasks in higher input dimensions. It identified that the neural network based methods were no longer inferior to the Gaussian process in two and five dimensions and that the NOMU method ranked consistently, irrespective of the underlying synthetic function, among the top two methods.

Over the course of the various experiments several observations about characteristics of the different estimators were made. It showed that the Deep ensemble method by default estimates uncertainty with a very small magnitude in comparison to the other methods. For the MC Dropout method is was observed that the uncertainty bounds lie like a tube around the mean prediction with rather large uncertainties even at already observed points. This contrasts the shape of the uncertainty bounds of the other methods which look more like arches from one sample to the next sample [1].

A user friendly and highly configurable framework for running Bayesian optimization was developed and used to perform all experiments. The framework was developed in the

[1] In the noiseless case which is considered here, taking a sample should give an accurate result without any uncertainty, thus small or zero uncertainty at observed points are favorable.

programming language Python and focuses on modularity and extensibility to facilitate future extensions and modifications to certain sub processes. For an easy distribution the framework can be packaged as a Python library.

The Mixed Integer Programming Problem for solving a neural network to find its optima and the respective input was adjusted to fit the network structure of the NOMU method.

8

Future Work

In this thesis various different synthetic functions were tested allowing a general assessment of the different estimators. One possible way of extending this work is to test more function and function classes with the goal to characterize in more details how they affects the different methods.

In this thesis the application of the Mixed Integer Programming Solver as a method of acquisition function optimization for the NOMU methods was introduced but never used in a proper experimental setup. Thus one natural extension to this work is to explore the implications of using an MIP solver to optimize the acquisition function for the NOMU method and possibly other methods. As mentioned in this work even the DIRECT optimizer suffers the curse of dimensionality and thus the MIP optimizer could be used as the base for experiments in even higher dimension which previously were considered as infeasible for Bayesian optimization using the Gaussian process scheme.

This thesis showed that the different neural network estimators produce uncertainty estimation of very different scales. Therefore another interesting topic for further investigation is how to derive a reasonable scale for the uncertainty, how exactly does the scale itself influences the Bayesian optimization performance and which scales should be used in practice for real world problems. One can transition into the topic of hyperparameter optimization as the scale of the uncertainty can also be seen as a hyperparameter. There are multiple hyperparameters that can be configured for all of the neural network based estimators as well as the adaptions presented in this thesis. For practice it would be interesting to optimize these parameters with the goal of further improve the Bayesian optimization results.

This thesis assessed the application of neural networks for Bayesian optimization exclusively on synthetic functions. Presenting practical solutions to the integration of data noise and providing an extension to real world data are possible extensions to this work.

References

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. In *32nd International Conference on Machine Learning (ICML)*.

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

Chen, T., Fox, E. B., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo.

Cheng, C.-H., Nührenberg, G., and Ruess, H. (2017). Maximum resilience of artificial neural networks.

Chryssolouris, G., Lee, M., and Ramsey, A. (1996). Confidence interval prediction for neural network models. *IEEE Transactions on Neural Networks*, 7(1):229–232.

Curi, S., Berkenkamp, F., and Krause, A. (2020). Efficient model-based reinforcement learning through optimistic policy search and planning.

Damianou, A. and Lawrence, N. D. (2013). Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. (2013). Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, page 3.

Fischetti, M. and Jo, J. (2018). Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3):296–309.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Frénay, B. and Verleysen, M. (2013). Classification in the presence of label noise: a survey. *IEEE transactions on neural networks and learning systems*, 25(5):845–869.

Gal, Y. and Ghahramani, Z. (2016). Dropout as a bayesian approximation: Representing model uncertainty in deep learning.

Gal, Y., Islam, R., and Ghahramani, Z. (2017). Deep bayesian active learning with image data.

Graves, A. (2011). Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356.

Grossmann, I. E. (2002). Review of nonlinear mixed-integer and disjunctive programming techniques. *Optimization and engineering*, 3(3):227–252.

Heckerman, D. (2008). A tutorial on learning with bayesian networks. In *Innovations in Bayesian networks*, pages 33–82. Springer.

Heiss, J., Wutte, H., Weissteiner, J., Seuken, S., and Teichmann, J. (2021). Nomu: Neural optimization-based model uncertainty.

Hernández-Lobato, J. M. and Adams, R. (2015). Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning*, pages 1861–1869.

Huang, D., Allen, T. T., Notz, W. I., and Miller, R. A. (2006). Sequential kriging optimization using multiple-fidelity evaluations. *Structural and Multidisciplinary Optimization*, 32(5):369–382.

IBM (2021). IBM CPLEX Optimizer. *https://www.ibm.com/analytics/cplex-optimizer*. [Online; accessed 2-Feb-2021].

Jones, D. R. (2001a). *Direct Global Optimization Algorithm*, pages 431–440. Springer US, Boston, MA.

Jones, D. R. (2001b). A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383.

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

Khardon, R. and Wachman, G. (2007). Noise tolerant variants of the perceptron algorithm. *Journal of Machine Learning Research*, 8(Feb):227–248.

Khosravi, A., Nahavandi, S., Creighton, D., and Atiya, A. F. (2010). Lower upper bound estimation method for construction of neural network-based prediction intervals. *IEEE transactions on neural networks*, 22(3):337–346.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2016). Simple and scalable predictive uncertainty estimation using deep ensembles.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Li, C.-L., Kandasamy, K., Póczos, B., and Schneider, J. (2016). High dimensional bayesian optimization via restricted projection pursuit models. In *Artificial Intelligence and Statistics*, pages 884–892. PMLR.

Malkomes, G. and Garnett, R. (2018). Automating bayesian optimization with bayesian optimization. *Advances in Neural Information Processing Systems*, 31:5984–5994.

Matérn, B. (2013). *Spatial variation*, volume 36. Springer Science & Business Media.

Močkus, J. (1975a). On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer.

Močkus, J. (1975b). On bayesian methods of optimization. *Towards Global Optimization*, pages 166–181.

Močkus, J. and Močkus, L. (1991). Bayesian approach to global optimization and application to multiobjective and constrained problems. *Journal of Optimization Theory and Applications*, 70(1):157–172.

Moriconi, R., Deisenroth, M. P., and Kumar, K. S. S. (2020). High-dimensional bayesian optimization using low-dimensional feature spaces.

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*.

Pan, S. J. and Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359.

Pearce, T., Zaki, M., Brintrup, A., and Neely, A. (2018). High-quality prediction intervals for deep learning: A distribution-free, ensembled approach.

Sasena, M. J. (2002). *Flexibility and efficiency enhancements for constrained global design optimization with kriging approximations*. PhD thesis, University of Michigan Ann Arbor, MI.

Sigurdsson, S., Larsen, J., Hansen, L. K., Philipsen, P. A., and Wulf, H.-C. (2002). Outlier estimation and detection application to skin lesion classification. In *IEEE International Conference On Acoustic Speech and Signal Processing*, volume 1, pages I–1049. IEEE; 1999.

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

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. W. (2012). Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Swersky, K., Snoek, J., and Adams, R. P. (2013). Multi-task bayesian optimization. In *Advances in neural information processing systems*, pages 2004–2012.

Wang, J., Clark, S. C., Liu, E., and Frazier, P. I. (2020). Parallel bayesian global optimization of expensive functions. *Operations Research*.

Wang, Z., Hutter, F., Zoghi, M., Matheson, D., and de Freitas, N. (2016). Bayesian optimization in a billion dimensions via random embeddings.

Weissteiner, J. and Seuken, S. (2020). Deep learning-powered iterative combinatorial auctions. In *AAAI*, pages 2284–2293.

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

Wilson, J. T., Hutter, F., and Deisenroth, M. P. (2018). Maximizing acquisition functions for bayesian optimization.

Zhilinskas, A. (1975). Single-step bayesian search method for an extremum of functions of a single variable. *Cybernetics*, 11(1):160–166.

A

Appendix

The following additional materials presents complementing graphs to the figures previously shown. This additional material section is structured in the same way as the experimental section (Section 6). Only exception is that the additional material for Section 6.3 and Section 6.4 are combined into one section presenting all results to the one dimensional experiments. Subsequent to the sections extending on the experimental work, in section A.4 important information about the implementation part of this work is given together with the overall structure of the code base and its main characteristics.

A.1 One Dimensional Synthetic Functions Experiment

A.1.1 Preparatory Work

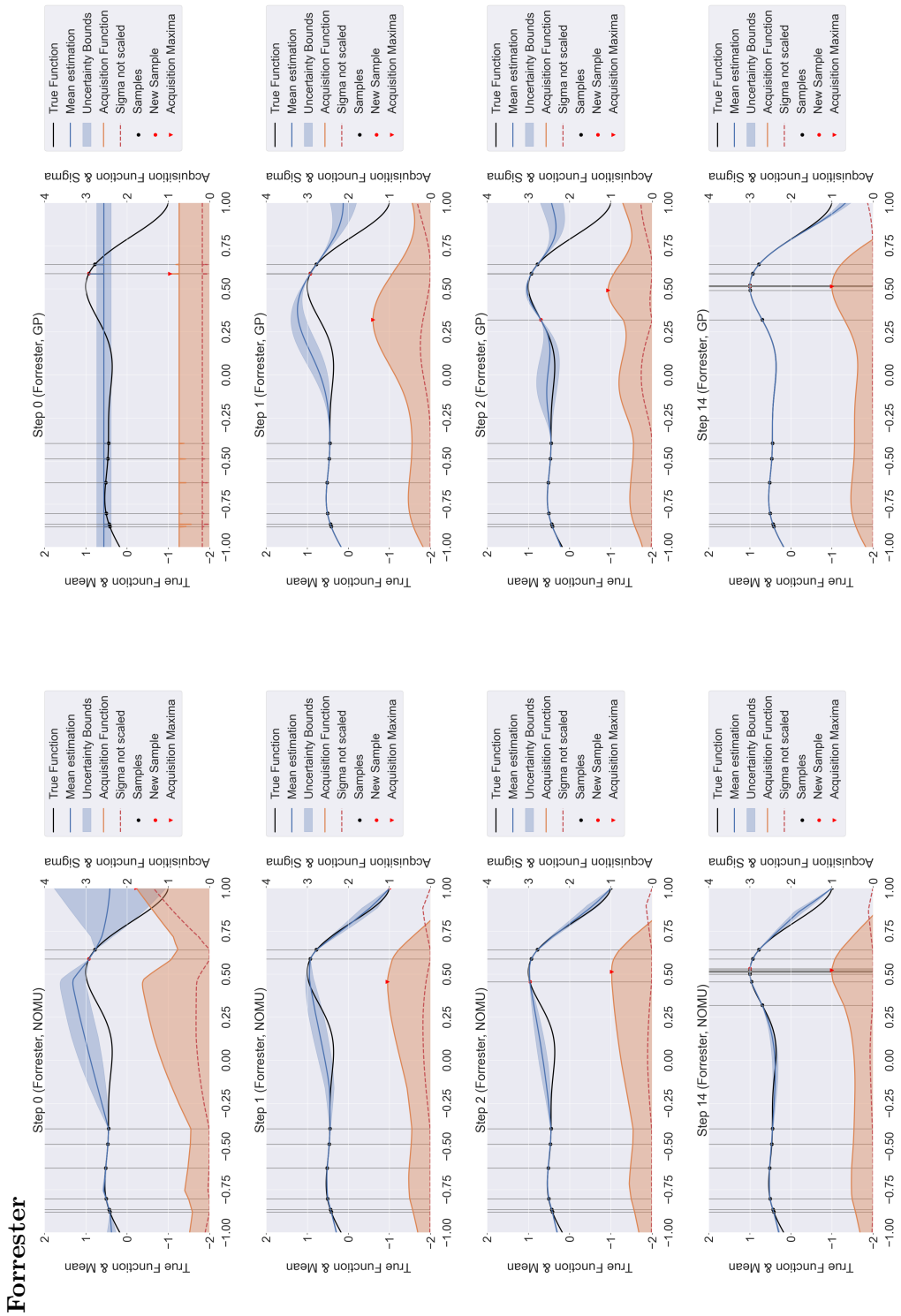


Figure A.1: **BO run for the NOMU and GP method**, instance Nr. 8, NOMU (left) and Gaussian process (right), first three steps and last step

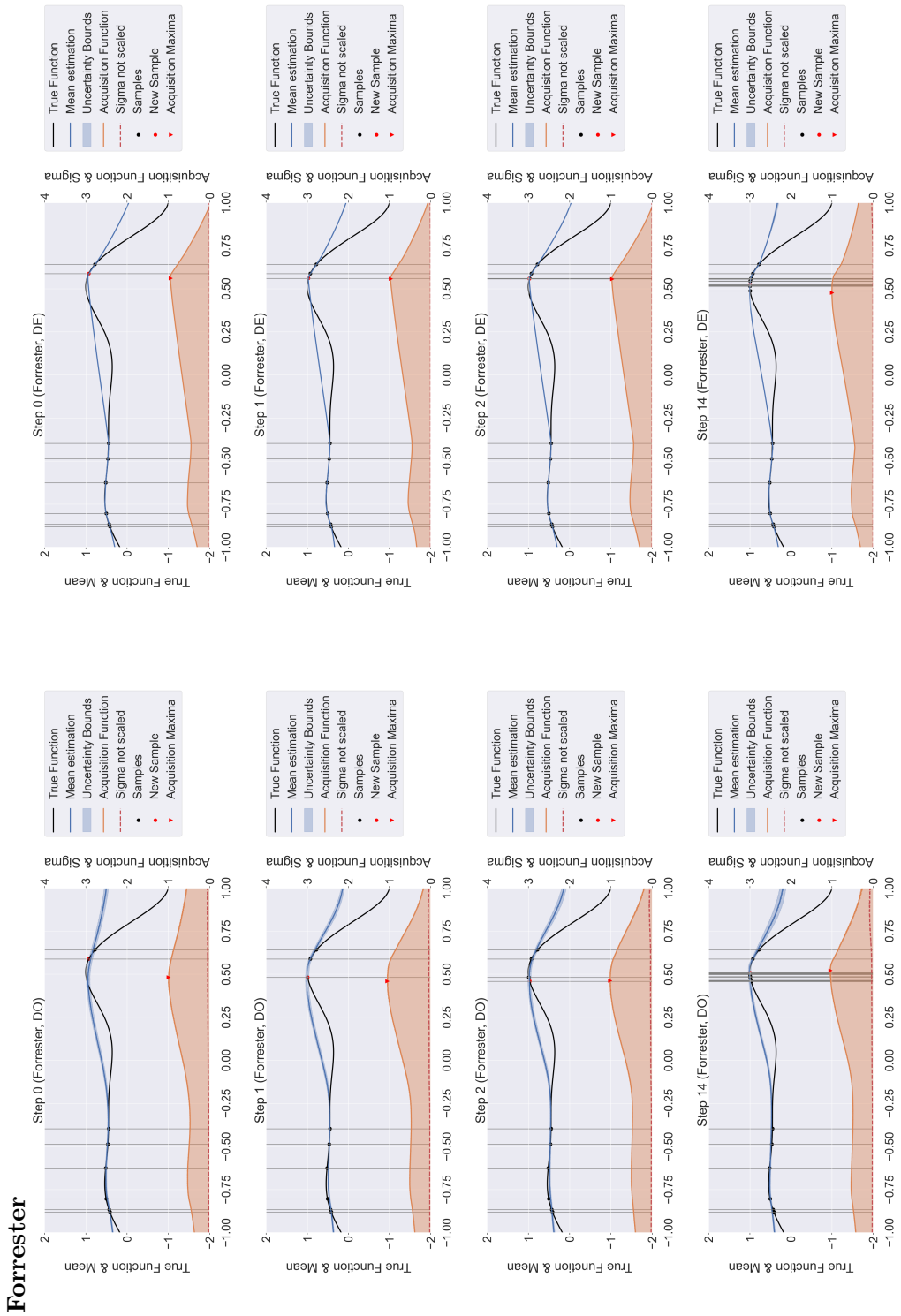
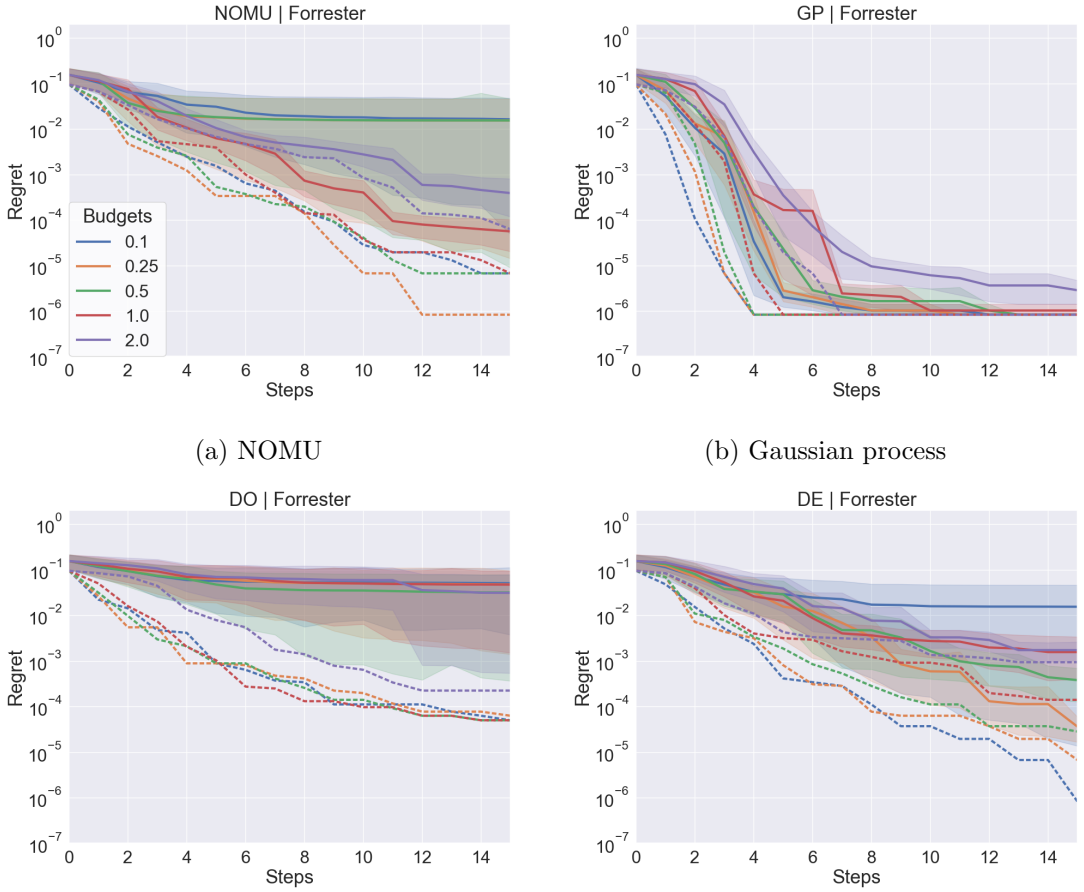


Figure A.2: **BO run for the DO and DE method**, instance Nr. 8, MC Dropout (left) and Deep Ensembles (right), first three steps and last step

A.1.2 Mean Width Budget Effects



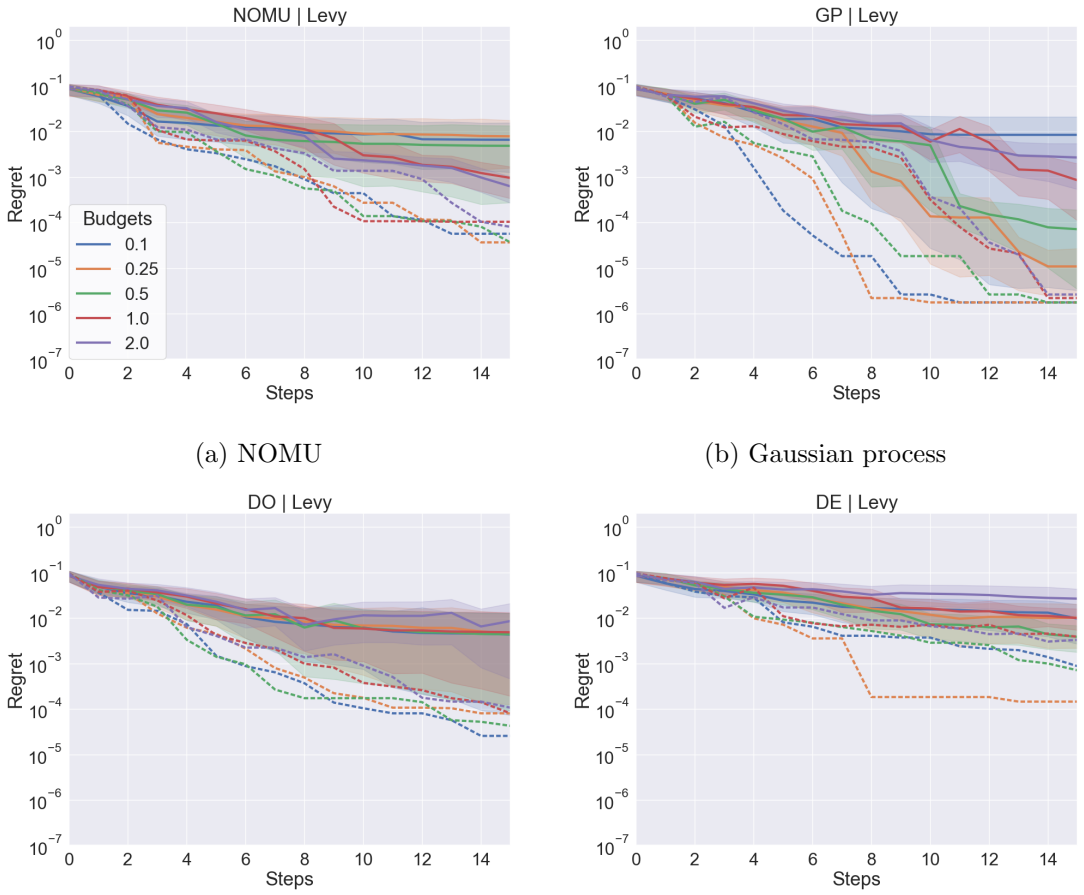
(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.3: **1D Forrester, regret curve for different mean width budgets per method**. Each plot depicts one curve of the mean regret over 100 instances for each of the mean width budgets (0.1, 0.25, 0.5, 1.0, 2.0). The colored area shows the bootstrapped 95% confidence interval for the mean and the dashed line depicts the median over the set of instances. The graph starts at step 0 where the regret of the starting sample set of 8 samples is shown.



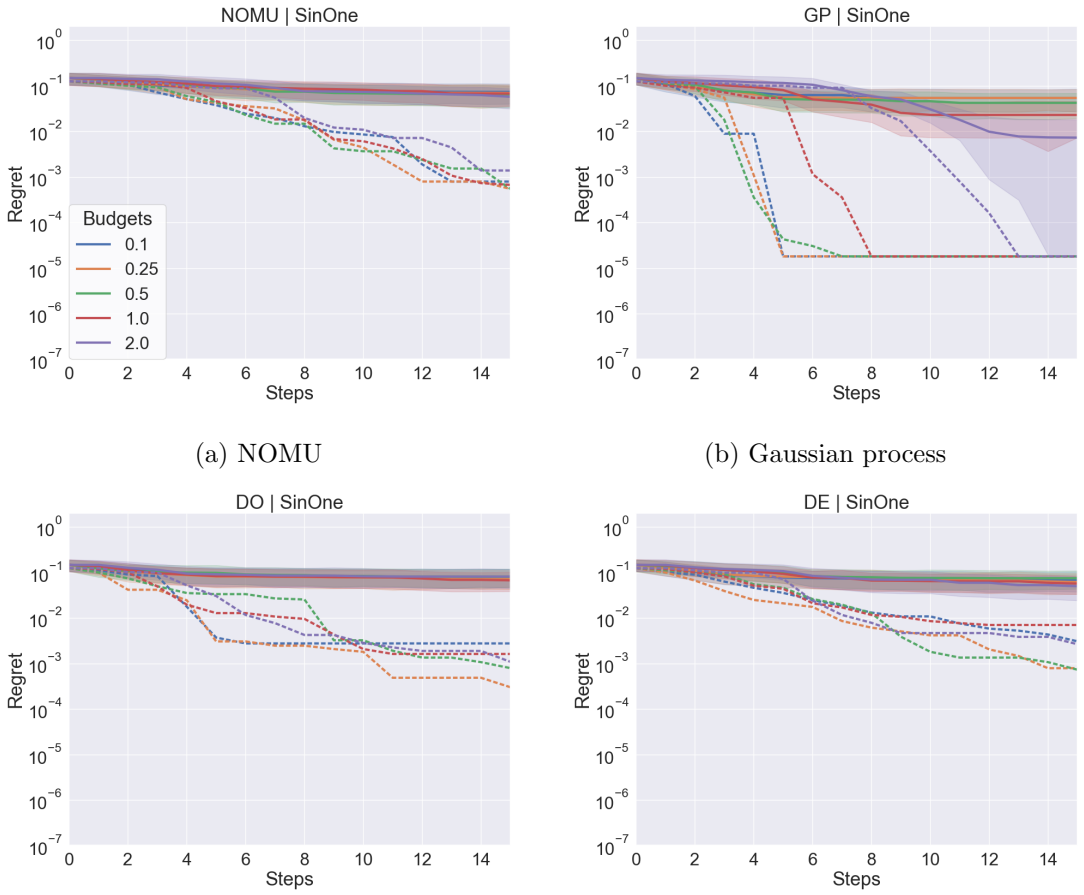
(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.4: **1D Levy, regret curve for different mean width budgets per method**. Each plot depicts one curve of the mean regret over 100 instances for each of the mean width budgets (0.1,0.25,0.5,1.0,2.0). The colored area shows the bootstrapped 95% confidence interval for the mean and the dashed line depicts the median over the set of instances. The graph starts at step 0 where the regret of the starting sample set of 8 samples is shown.



(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.5: **1D SinOne, regret curve for different mean width budgets per method**. Each plot depicts one curve of the mean regret over 100 instances for each of the mean width budgets (0.1, 0.25, 0.5, 1.0, 2.0). The colored area shows the bootstrapped 95% confidence interval for the mean and the dashed line depicts the median over the set of instances. The graph starts at step 0 where the regret of the starting sample set of 8 samples is shown.

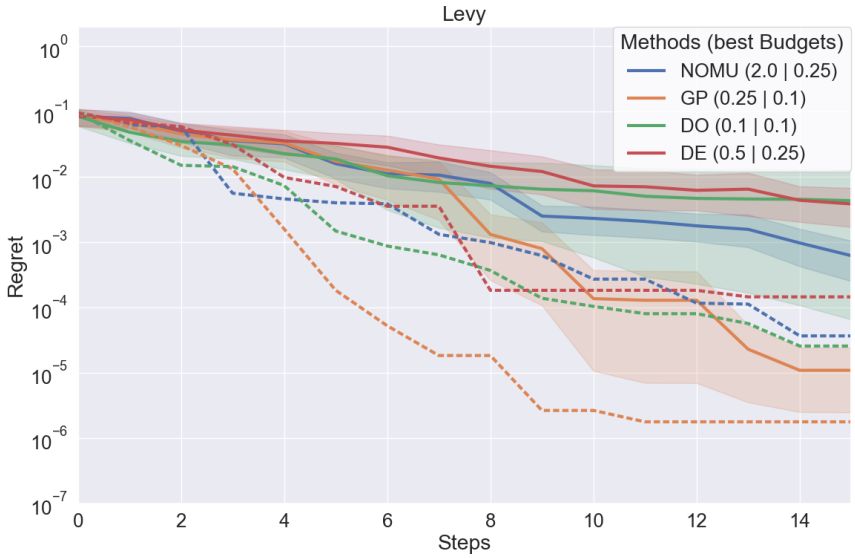


Figure A.6: **Mean and median regret curve for all estimator with their best budget on Levy.** For each estimator the best mean width budget is selected for the mean and the median the respective regret curves are displayed. For the mean the budgets are selected by lowest upper confidence bound of the final regrets. The selected budgets are indicated in the legend after the method identifiers by the numbers in the bracket where the first one displays the best budget for the mean and the second one the best for the median.

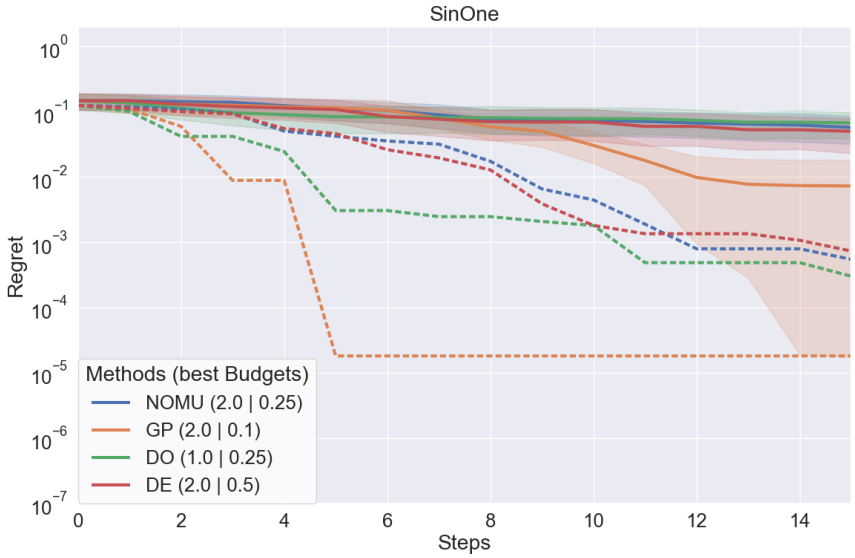
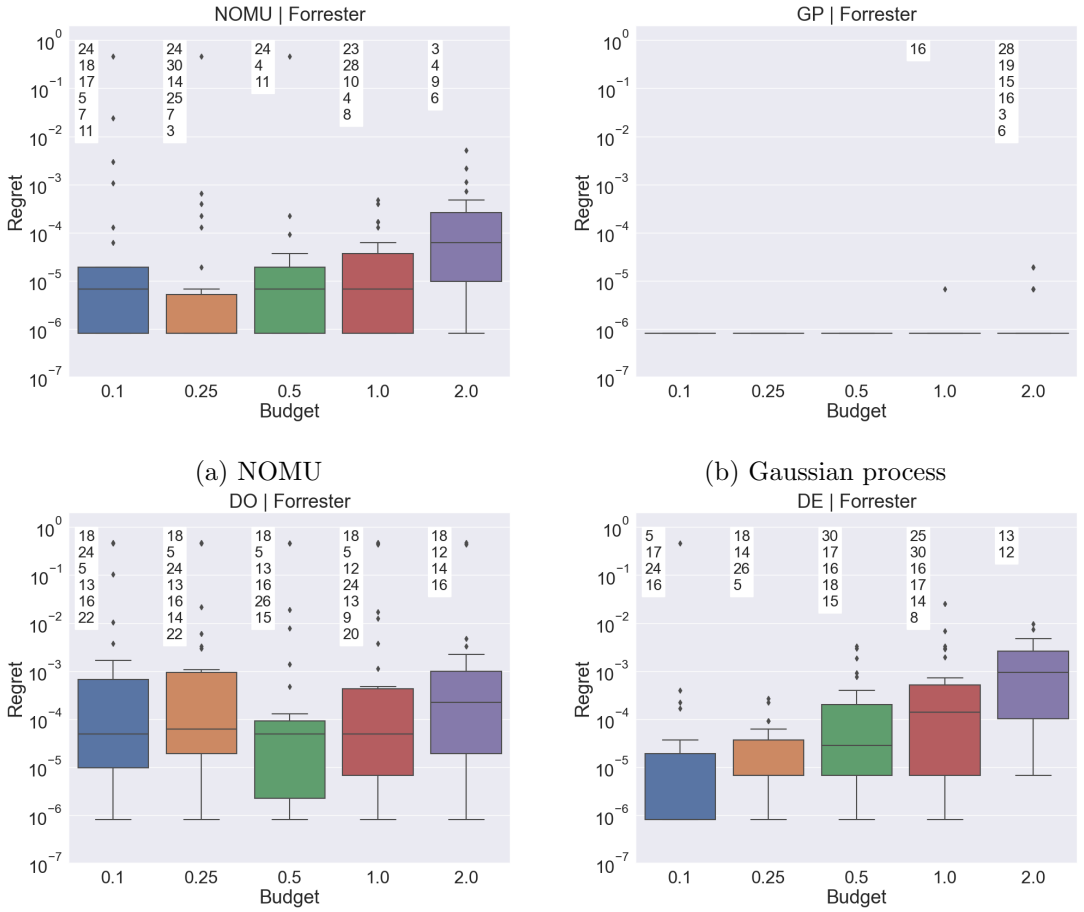


Figure A.7: **Mean and median regret curve for all estimator with their best budget on SinOne.** For each estimator the best mean width budget is selected for the mean and the median the respective regret curves are displayed. For the mean the budgets are selected by lowest upper confidence bound of the final regrets. The selected budgets are indicated in the legend after the method identifiers by the numbers in the bracket where the first one displays the best budget for the mean and the second one the best for the median.

		Mean Width Budget									
f	estimator	0.1		0.25		0.5		1.0		2.0	
Forrester	NOMU	6.70×10^{-6}	2	$\mathbf{8.25 \times 10^{-7}}$	**1**	6.70×10^{-6}	2	6.70×10^{-6}	2	6.27×10^{-5}	3
	GP	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**	$\mathbf{8.25 \times 10^{-7}}$	**1**
	DE	$\mathbf{8.25 \times 10^{-7}}$	**1**	6.70×10^{-6}	2	2.82×10^{-5}	3	1.40×10^{-4}	4	9.34×10^{-4}	5
	DO	$\mathbf{4.98 \times 10^{-5}}$	**1**	6.27×10^{-5}	2	$\mathbf{4.98 \times 10^{-5}}$	**1**	$\mathbf{4.98 \times 10^{-5}}$	**1**	2.23×10^{-4}	3
Levy	NOMU	3.75×10^{-5}	2	$\mathbf{2.09 \times 10^{-5}}$	**1**	$\mathbf{2.09 \times 10^{-5}}$	**1**	1.04×10^{-4}	4	5.44×10^{-5}	3
	GP	$\mathbf{1.76 \times 10^{-6}}$	**1**	$\mathbf{1.76 \times 10^{-6}}$	**1**	$\mathbf{1.76 \times 10^{-6}}$	**1**	$\mathbf{1.76 \times 10^{-6}}$	**1**	2.20×10^{-6}	2
	DE	8.73×10^{-4}	3	$\mathbf{1.41 \times 10^{-4}}$	**1**	4.98×10^{-4}	2	4.55×10^{-3}	5	1.60×10^{-3}	4
	DO	$\mathbf{2.56 \times 10^{-5}}$	**1**	4.25×10^{-5}	2	4.25×10^{-5}	2	5.66×10^{-5}	3	1.04×10^{-4}	4
SinOne	NOMU	7.90×10^{-4}	3	$\mathbf{5.45 \times 10^{-4}}$	**1**	$\mathbf{5.45 \times 10^{-4}}$	**1**	6.69×10^{-4}	2	1.38×10^{-3}	4
	GP	$\mathbf{1.80 \times 10^{-5}}$	**1**	$\mathbf{1.80 \times 10^{-5}}$	**1**	$\mathbf{1.80 \times 10^{-5}}$	**1**	$\mathbf{1.80 \times 10^{-5}}$	**1**	$\mathbf{1.80 \times 10^{-5}}$	**1**
	DE	3.07×10^{-3}	4	7.86×10^{-4}	2	$\mathbf{7.29 \times 10^{-4}}$	**1**	7.01×10^{-3}	5	2.65×10^{-3}	3
	DO	2.76×10^{-3}	5	$\mathbf{3.00 \times 10^{-4}}$	**1**	7.90×10^{-4}	2	1.62×10^{-3}	4	1.09×10^{-3}	3

Table A.1: **C values per method and mean width budget.** This table shows the mean scaling factor c derived from the estimated uncertainty and the give mean width budget per estimator method. These c-values have then been used throughout the whole process to scale the uncertainty.



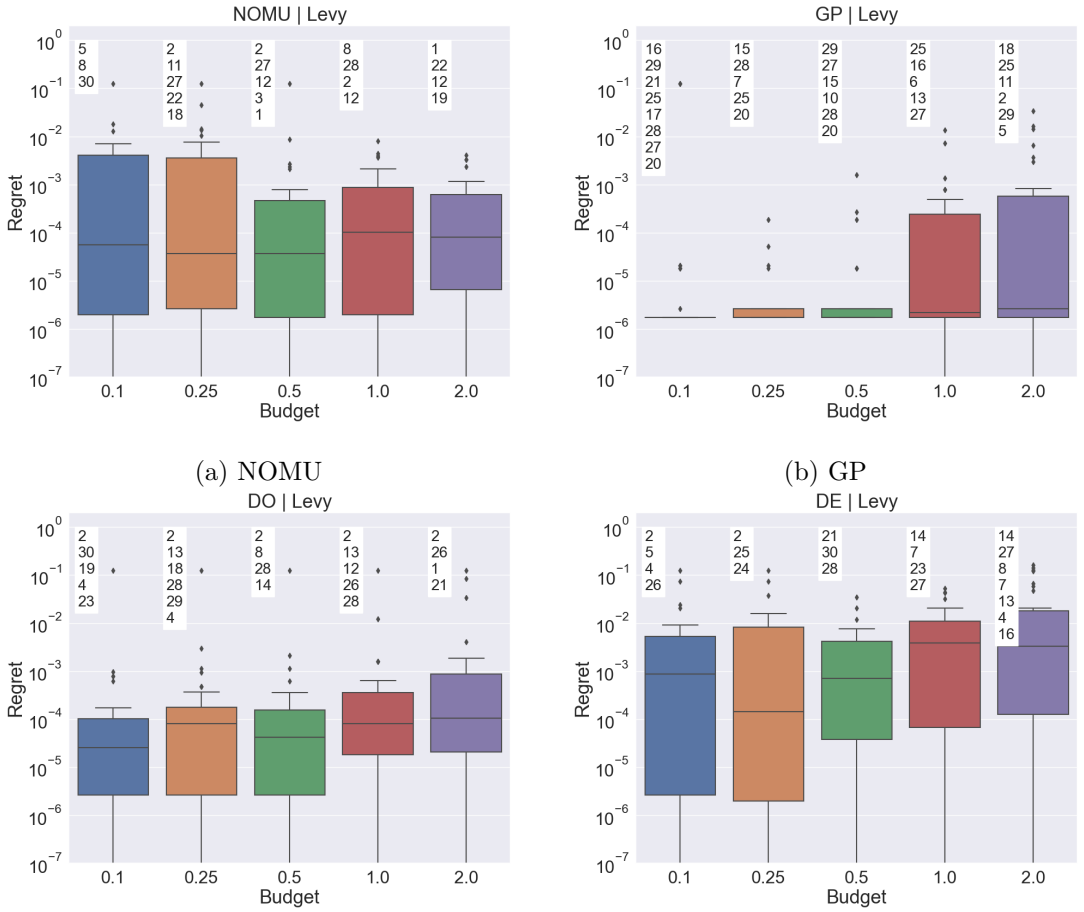
(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensemble

Figure A.8: **Forrester - Final regret distribution for all mean width budgets method.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret. All these results represent the situation on the FORRESTER function



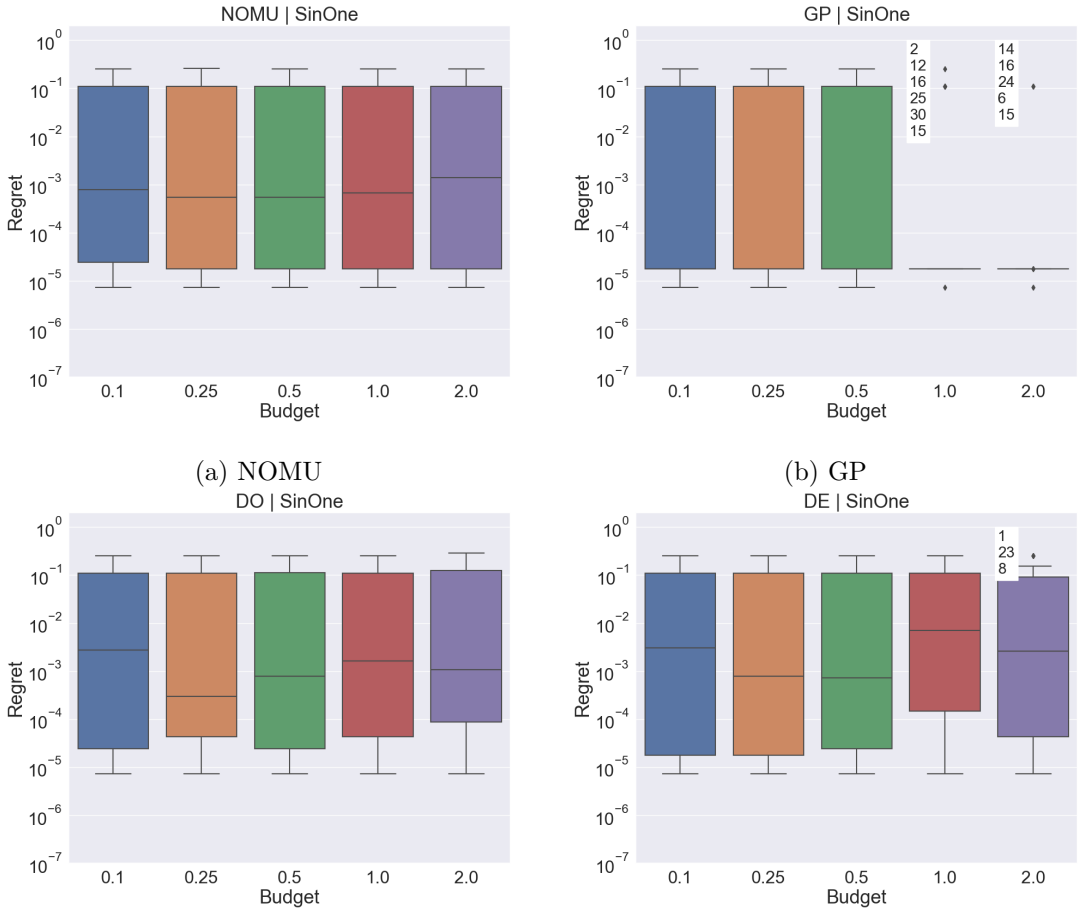
(a) NOMU

(b) GP

(c) DO

(d) DE

Figure A.9: **Levy - Final regret distribution for all mean width budgets method.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret. All these results represent the situation on the Levy function



(a) NOMU

(b) GP

(c) DO

(d) DE

Figure A.10: **SinOne - Final regret distribution for all mean width budgets method.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret. All these results represent the situation on the SinOne function

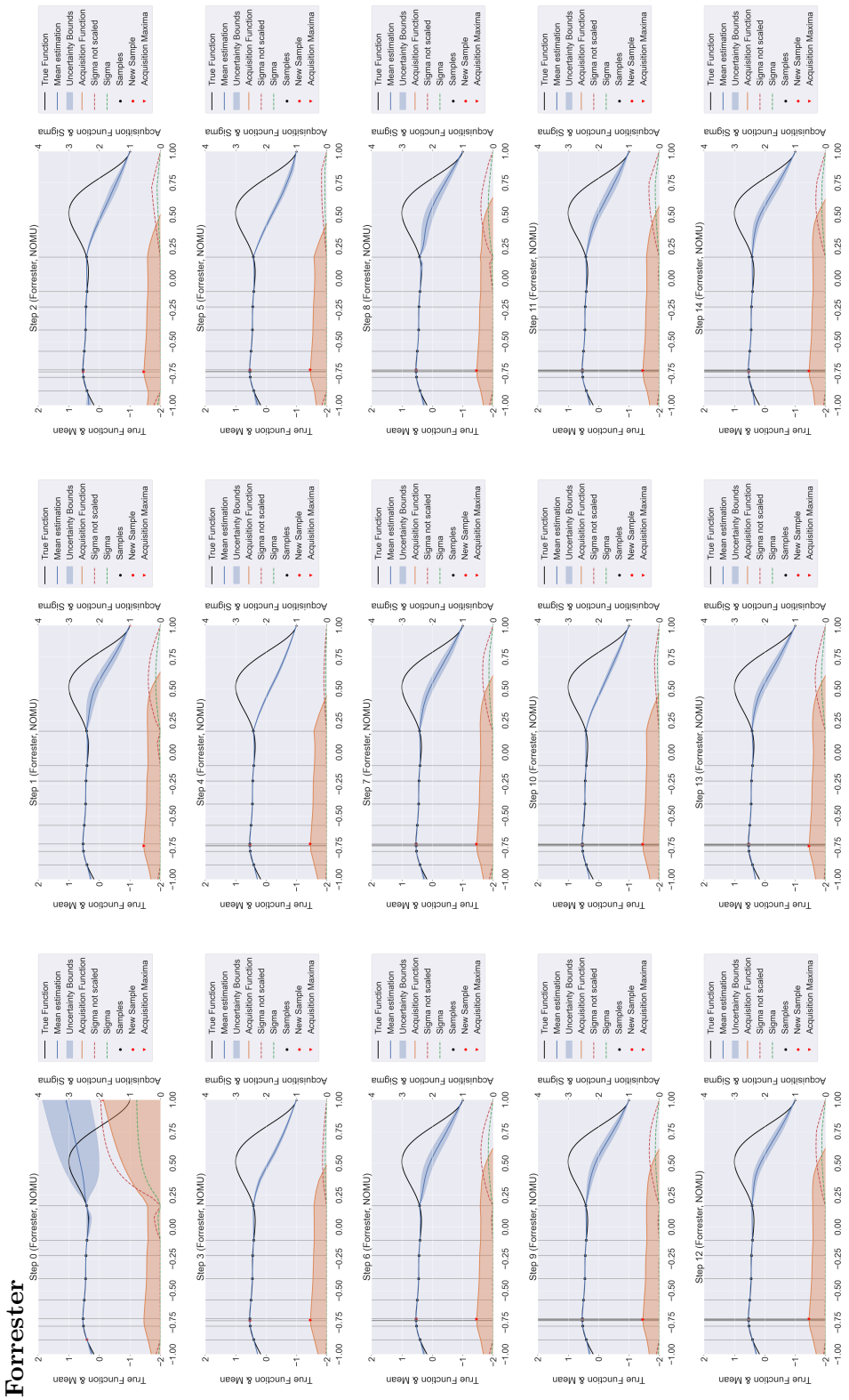
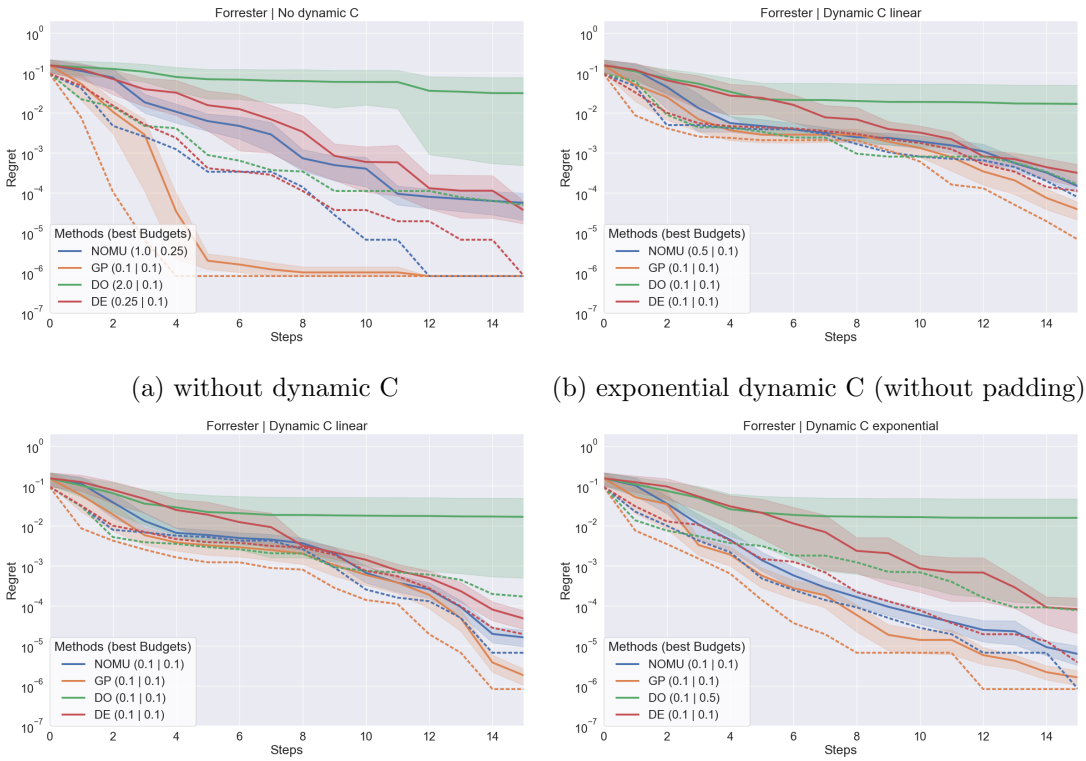


Figure A.11: **BO run for the NOMU method, run 24**, mean width budget: 0.5

A.1.3 Dynamic C strategies Comparisons



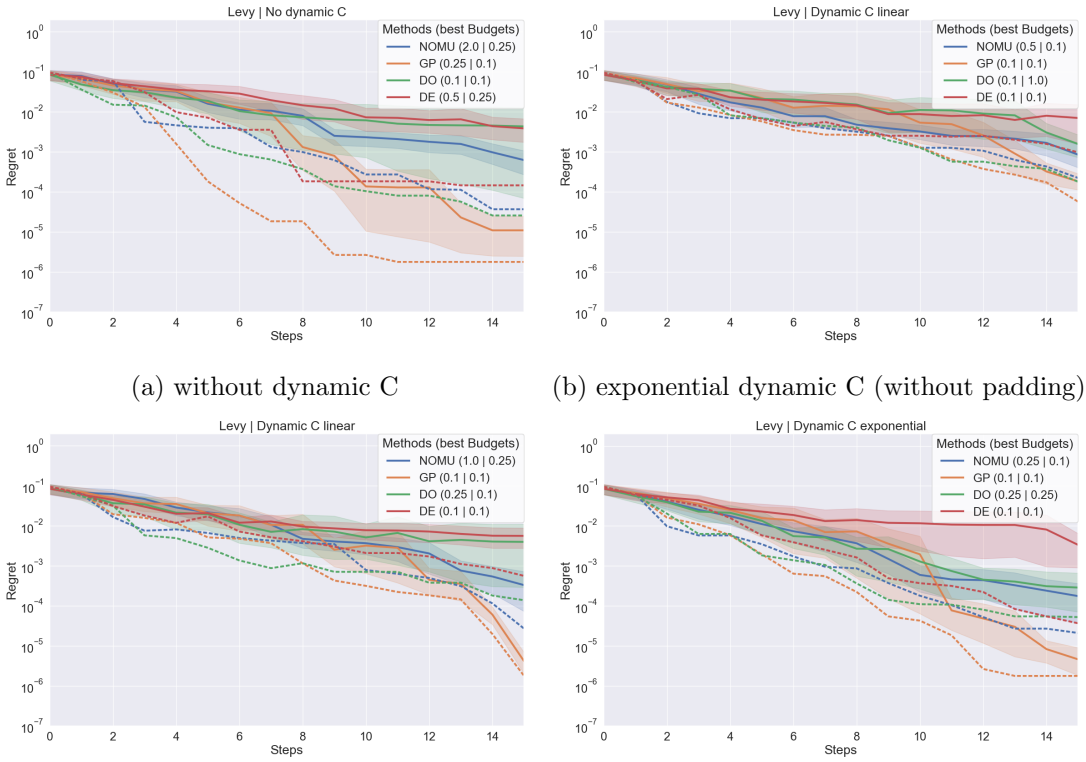
(a) without dynamic C

(b) exponential dynamic C (without padding)

(c) exponential dynamic C (with padding)

(d) exponential dynamic C (with padding)

Figure A.12: **Forrester - All methods with best budget, different Dynamic C approaches.** Each graph displays one regret curve per method. For each methods the budget which produces the smallest regrets are selected, individually for mean and median.



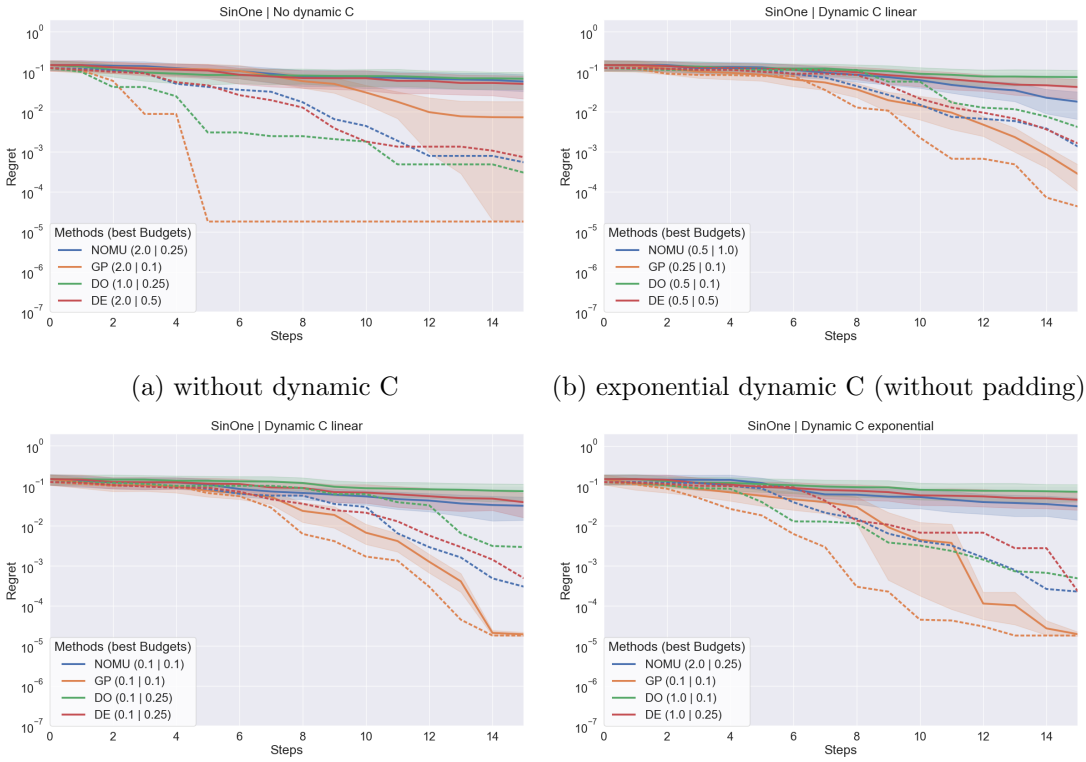
(a) without dynamic C

(b) exponential dynamic C (without padding)

(c) exponential dynamic C (with padding)

(d) exponential dynamic C (with padding)

Figure A.13: **Levy - All methods with best budget, different Dynamic C approaches.** Each graph displays one regret curve per method. For each methods the budget which produces the smallest regrets are selected, individually for mean and median.



(a) without dynamic C

(b) exponential dynamic C (without padding)

(c) exponential dynamic C (with padding)

(d) exponential dynamic C (with padding)

Figure A.14: **SinOne - All methods with best budget, different Dynamic C approaches.** Each graph displays one regret curve per method. For each methods the budget which produces the smallest regrets are selected, individually for mean and median.

			Mean Width Budget									
			0.1		0.25		0.5		1.0		2.0	
f		variant	μ	M	μ	M	μ	M	μ	M	μ	M
Levy	NOMU	no dc	**1**	10	**1**	6	**1**	6	3	16	3	13
		lin.	**1**	15	**1**	18	**1**	17	**1**	19	**1**	20
		lin. pad.	**1**	5	**1**	3	**1**	11	**1**	6	**1**	12
		exp. pad.	**1**	**1**	**1**	**1**	**1**	4	**1**	6	**1**	14
	GP	no dc	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	11
		lin.	5	14	**1**	18	**1**	15	**1**	19	10	20
		lin. pad.	**1**	**1**	**1**	**1**	**1**	**1**	**1**	11	10	16
		exp. pad.	**1**	**1**	**1**	**1**	**1**	**1**	3	13	12	16
	DE	no dc	**1**	7	**1**	2	**1**	6	**1**	18	4	12
		lin.	**1**	4	**1**	13	**1**	9	**1**	15	13	20
		lin. pad.	**1**	5	**1**	10	**1**	14	**1**	15	5	19
		exp. pad.	**1**	**1**	**1**	3	**1**	8	**1**	11	3	15
	DO	no dc	**1**	**1**	**1**	2	**1**	2	**1**	7	**1**	10
		lin.	**1**	17	**1**	14	**1**	15	**1**	13	**1**	20
		lin. pad.	**1**	5	**1**	10	**1**	19	3	15	**1**	18
		exp. pad.	**1**	5	**1**	4	**1**	10	**1**	9	**1**	7

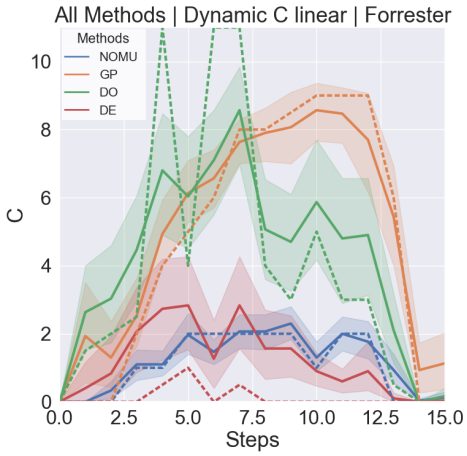
Table A.2: **Levy - Rank table for the different budgets per estimator and Dynamic C variant.** For each estimator and each Dynamic C variant the final regrets for the different mean width budgets are ranked according to the ranking scheme which results in the ranks indicated in the column μ. Using a normal ranking where lowest equals best also the medians are ranked in the column with header M. The whole table presents results for the Levy function. "no dc" means that there in no Dynamic C however mean width scaling is still present. "lin." means that the ε decay linearly. "lin. pad." additionally has padding added to the strategy. The same goes for "exp. pad." where however the ε decays exponentially.

			Mean Width Budget									
			0.1		0.25		0.5		1.0		2.0	
f		variant	μ	M	μ	M	μ	M	μ	M	μ	M
SinOne	NOMU	no dc	2	13	2	6	**1**	6	2	8	**1**	16
		lin.	**1**	20	**1**	18	**1**	17	**1**	15	**1**	19
		lin. pad.	**1**	3	**1**	8	**1**	8	**1**	8	**1**	12
		exp. pad.	**1**	14	**1**	**1**	**1**	2	**1**	5	**1**	3
	GP	no dc	17	**1**	17	**1**	16	**1**	16	**1**	**1**	**1**
		lin.	9	16	10	16	11	16	11	16	11	20
		lin. pad.	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
		exp. pad.	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	DE	no dc	**1**	15	**1**	6	**1**	4	**1**	17	**1**	13
		lin.	**1**	12	**1**	14	**1**	9	**1**	20	**1**	18
		lin. pad.	**1**	4	**1**	3	**1**	11	**1**	10	**1**	19
		exp. pad.	**1**	7	**1**	**1**	**1**	**1**	**1**	8	**1**	16
	DO	no dc	**1**	9	**1**	**1**	**1**	5	**1**	8	**1**	7
		lin.	**1**	14	**1**	15	**1**	17	**1**	18	**1**	20
		lin. pad.	**1**	12	**1**	10	**1**	13	**1**	16	**1**	19
		exp. pad.	**1**	2	**1**	3	**1**	4	**1**	6	**1**	10

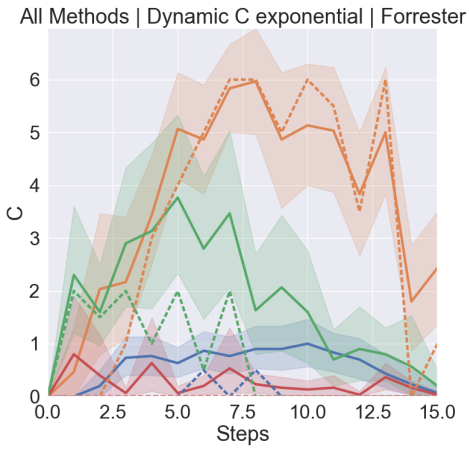
Table A.3: **SinOne - Rank table for the different budgets per estimator and Dynamic C variant.** For each estimator and each Dynamic C variant the final regrets for the different mean width budgets are ranked according to the ranking scheme which results in the ranks indicated in the column μ. Using a normal ranking where lowest equals best also the medians are ranked in the column with header M. The whole table presents results for the SinOne function. "no dc" means that there in no Dynamic C however mean width scaling is still present. "lin." means that the ε decay linearly. "lin. pad." additionally has padding added to the strategy. The same goes for "exp. pad." where however the ε decays exponentially.

			Mean Width Budget									
			0.1		0.25		0.5		1.0		2.0	
f		variant	μ	M	μ	M	μ	M	μ	M	μ	M
Forrester	NOMU	UB: no dc	**1**	4	**1**	**1**	**1**	4	2	4	7	28
		UB: lin.	16	30	13	30	14	33	18	36	19	36
		UB: lin. pad.	**1**	4	2	4	5	4	7	21	23	40
		UB: exp. pad.	**1**	**1**	**1**	4	**1**	4	2	18	12	33
		EI: no dc	**1**	4	**1**	4	**1**	4	9	23	11	23
		Ei: lin.	9	28	7	26	11	30	18	35	26	39
		EI: lin. pad.	2	4	7	19	5	19	9	23	18	36
		EI: exp. pad.	**1**	**1**	**1**	4	**1**	4	9	21	17	26
	GP	UB: no dc	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	8	**1**
		UB: lin.	27	31	27	36	28	36	32	39	32	40
		UB: lin. pad.	8	**1**	10	**1**	8	**1**	20	31	26	31
		UB: exp. pad.	8	**1**	**1**	**1**	10	**1**	10	**1**	20	31
		EI: no dc	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
		Ei: lin.	26	29	22	**1**	29	35	26	29	30	36
		EI: lin. pad.	8	**1**	**1**	**1**	10	**1**	**1**	**1**	10	**1**
		EI: exp. pad.	8	**1**	**1**	**1**	10	**1**	10	**1**	13	**1**
	DE	UB: no dc	**1**	**1**	**1**	3	3	11	**1**	24	19	40
		UB: lin.	3	18	5	24	5	28	**1**	27	18	36
		UB: lin. pad.	**1**	9	**1**	16	5	23	7	24	18	38
		UB: exp. pad.	**1**	2	**1**	3	**1**	3	5	18	12	37
		EI: no dc	**1**	8	**1**	3	**1**	12	3	18	10	32
		Ei: lin.	3	17	8	29	**1**	18	12	34	19	39
		EI: lin. pad.	**1**	12	**1**	12	7	18	9	31	17	35
		EI: exp. pad.	**1**	3	**1**	9	5	12	7	30	11	33
	DO	UB: no dc	**1**	2	**1**	5	**1**	2	**1**	2	**1**	21
		UB: lin.	**1**	18	**1**	21	**1**	30	**1**	33	**1**	40
		UB: lin. pad.	**1**	19	**1**	32	**1**	21	**1**	39	**1**	34
		UB: exp. pad.	**1**	10	**1**	16	**1**	7	**1**	20	**1**	21
		EI: no dc	**1**	**1**	**1**	10	**1**	5	**1**	14	**1**	25
		Ei: lin.	**1**	25	**1**	35	**1**	36	**1**	38	**1**	36
		EI: lin. pad.	**1**	10	**1**	27	**1**	28	**1**	29	**1**	31
		EI: exp. pad.	**1**	7	**1**	10	**1**	17	**1**	7	**1**	14

Table A.4: **Rank table - Upper Bound and Expected Improvement.** Ranks per method over all Dynamic C variants for both acquisition functions. All results refer to the FORRESTER function.



(a) linear dynamic C with padding



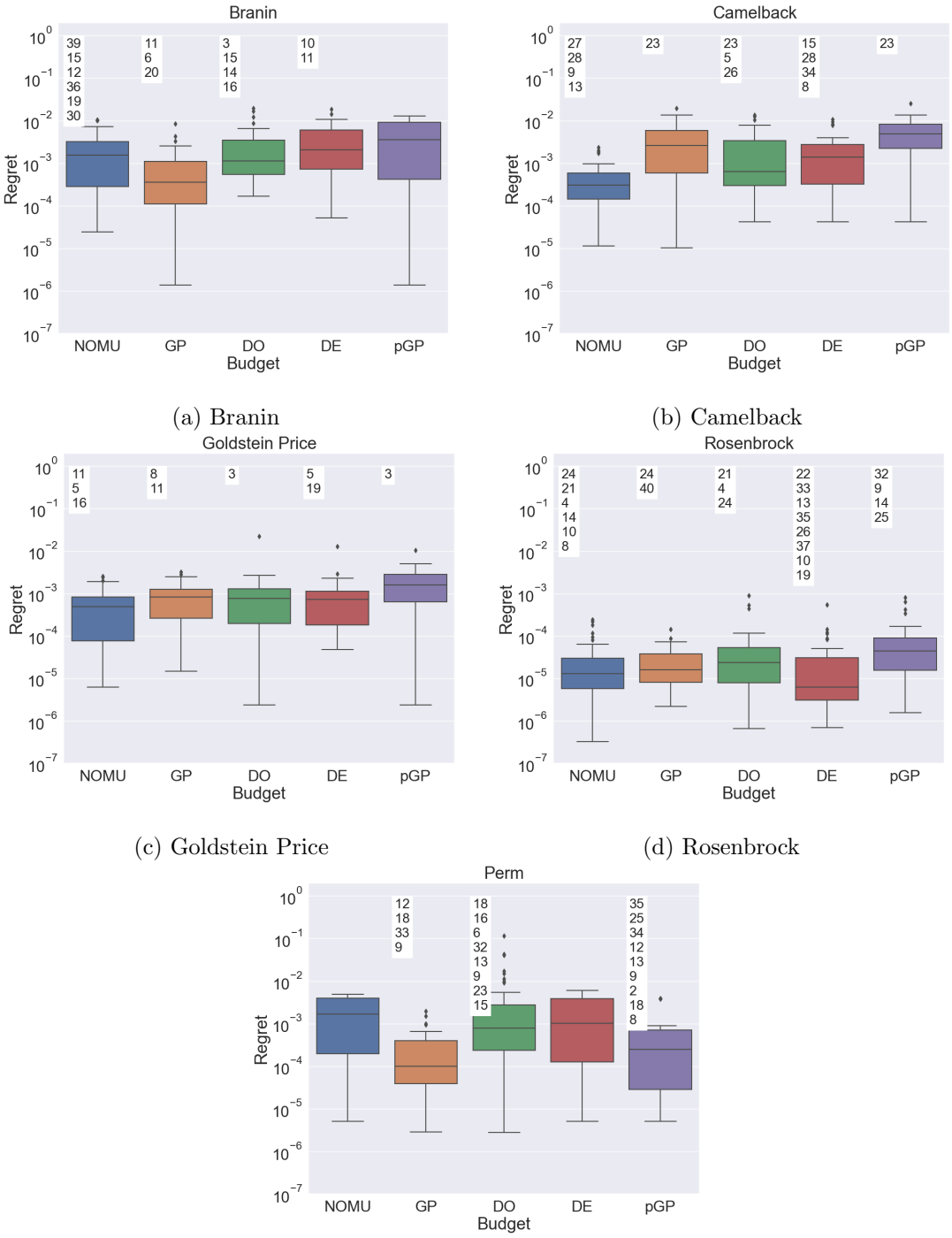
(b) exp dynamic C with padding

Figure A.15: **Comparison of C doubling between linear and exponential Dynamic C.** The solid lines depict how often the scaling factor c was doubled during the process as a mean over the different instances. The dashed lines show the median of the same data. Both graphs show the situation for the FORRESTER function.

A.2 Two Dimensional Synthetic Functions Experiment

	Estimator Methods				
function	NOMU	GP	DE	DO	pGP
Branin	$2.30 \cdot 10^{-5}$ $\pm 9.40 \cdot 10^{-6}$	$\mathbf{2.36 \cdot 10^{-6}}$ $\mathbf{\pm 1.35 \cdot 10^{-6}}$	$2.52 \cdot 10^{-5}$ $\pm 8.63 \cdot 10^{-6}$	$2.83 \cdot 10^{-4}$ $\pm 1.21 \cdot 10^{-4}$	$\mathbf{2.66 \cdot 10^{-4}}$ $\mathbf{\pm 4.91 \cdot 10^{-4}}$
Rosenbrock	$2.54 \cdot 10^{-6}$ $\pm 6.53 \cdot 10^{-7}$	$5.47 \cdot 10^{-6}$ $\pm 2.12 \cdot 10^{-6}$	$\mathbf{1.08 \cdot 10^{-6}}$ $\mathbf{\pm 3.62 \cdot 10^{-7}}$	$6.18 \cdot 10^{-6}$ $\pm 3.27 \cdot 10^{-6}$	$7.55 \cdot 10^{-6}$ $\pm 2.58 \cdot 10^{-6}$
Camelback	$\mathbf{1.33 \cdot 10^{-5}}$ $\mathbf{\pm 2.89 \cdot 10^{-6}}$	$4.00 \cdot 10^{-5}$ $\pm 1.14 \cdot 10^{-5}$	$7.57 \cdot 10^{-4}$ $\pm 3.73 \cdot 10^{-4}$	$1.02 \cdot 10^{-4}$ $\pm 3.86 \cdot 10^{-5}$	$8.40 \cdot 10^{-5}$ $\pm 5.44 \cdot 10^{-5}$
Perm	$\mathbf{1.73 \cdot 10^{-5}}$ $\mathbf{\pm 8.10 \cdot 10^{-6}}$	$\mathbf{1.95 \cdot 10^{-5}}$ $\mathbf{\pm 9.01 \cdot 10^{-6}}$	$\mathbf{2.16 \cdot 10^{-5}}$ $\mathbf{\pm 8.70 \cdot 10^{-6}}$	$1.60 \cdot 10^{-4}$ $\pm 7.19 \cdot 10^{-5}$	$7.77 \cdot 10^{-4}$ $\pm 4.85 \cdot 10^{-4}$
Goldstein Price	$\mathbf{1.42 \cdot 10^{-4}}$ $\mathbf{\pm 7.47 \cdot 10^{-5}}$	$\mathbf{1.16 \cdot 10^{-4}}$ $\mathbf{\pm 5.48 \cdot 10^{-5}}$	$2.19 \cdot 10^{-4}$ $\pm 8.57 \cdot 10^{-5}$	$3.44 \cdot 10^{-4}$ $\pm 1.39 \cdot 10^{-4}$	$\mathbf{9.58 \cdot 10^{-5}}$ $\mathbf{\pm 3.38 \cdot 10^{-5}}$

Table A.5: **Mean final regret per methods and function.** This table lists for all methods their mean final regret (top row) and the 95% confidence interval (bottom row) per function.



(a) Branin

(b) Camelback

(c) Goldstein Price

(d) Rosenbrock

(e) Perm

Figure A.16: **Final regret distribution for all methods per function.** For each mean width budget the distribution of the final regrets is displayed as a box plot where the box represents the interquartile range. The solid black line represents the median and the black dots show the outliers. For the outliers the instance number is listed top left ordered by size beginning with the largest regret.

Branin

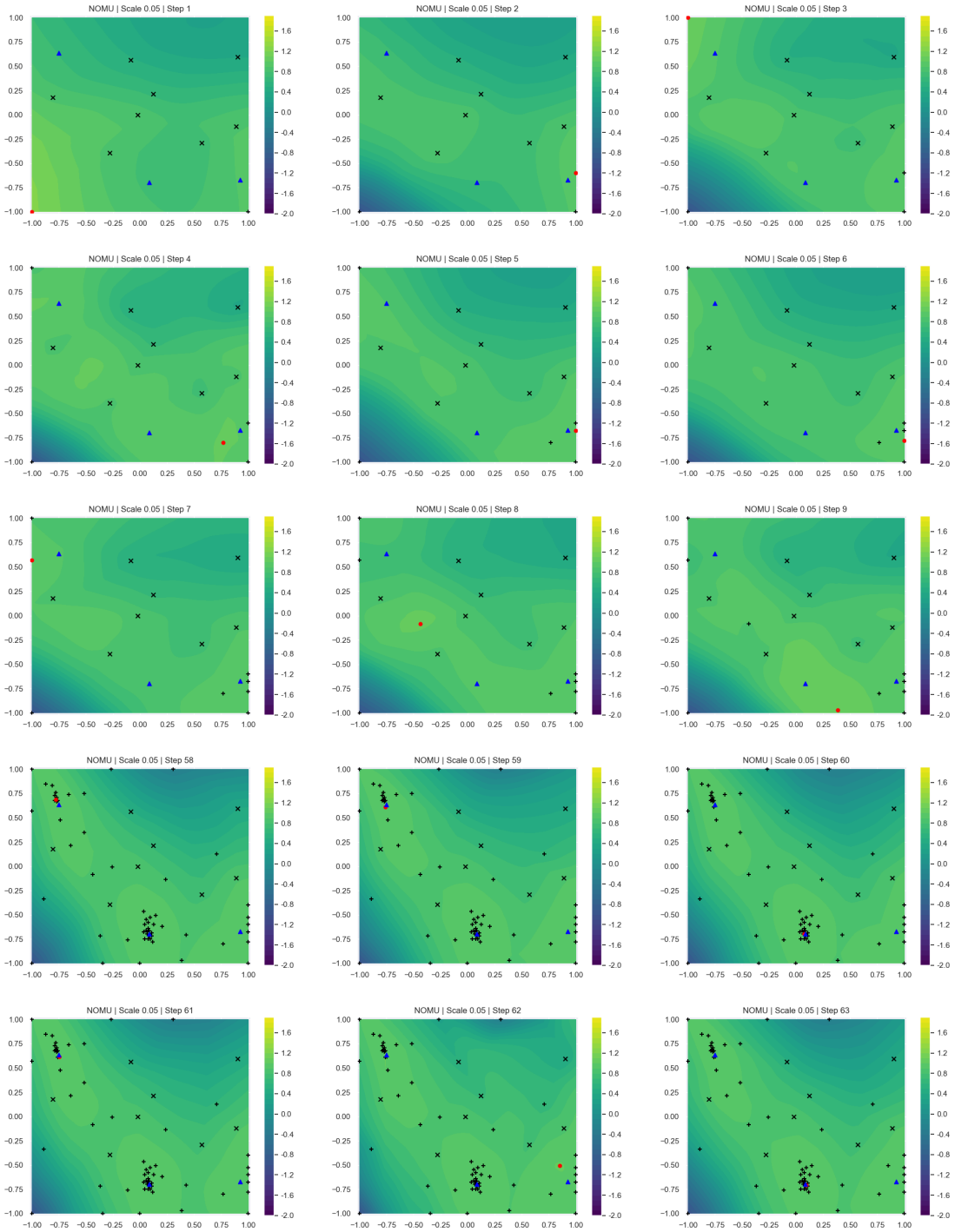


Figure A.17: **BO run for the NOMU method, run 30** First 9 and last 6 steps during the Bayesian optimization of the Branin function using NOMU with a mean width budget of 0.05.

Forrester

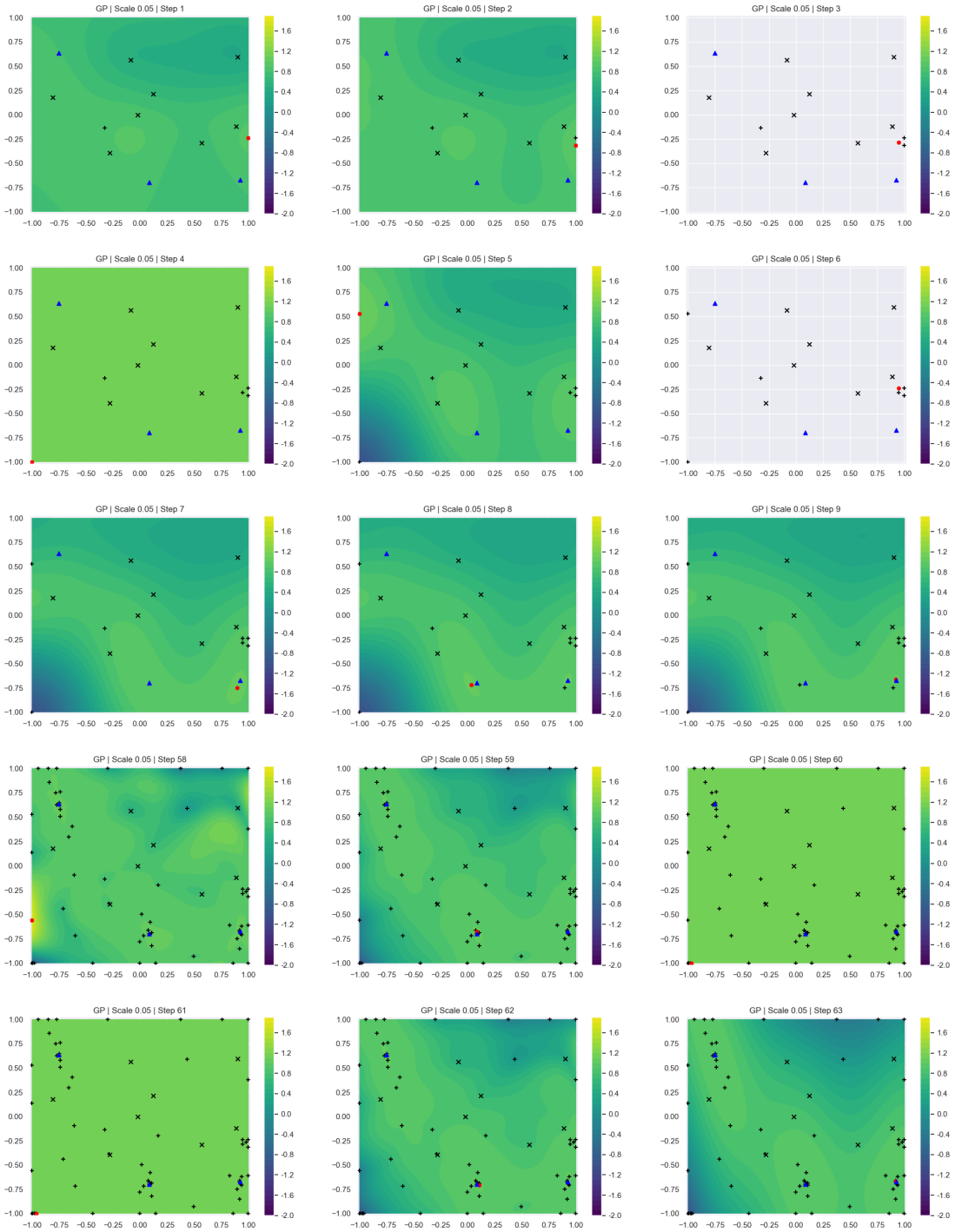


Figure A.18: **BO run for the Gaussian process, run 30.** First 9 and last 6 steps during the Bayesian optimization of the Branin function using Gaussian process with a mean width budget of 0.05.

Forrester

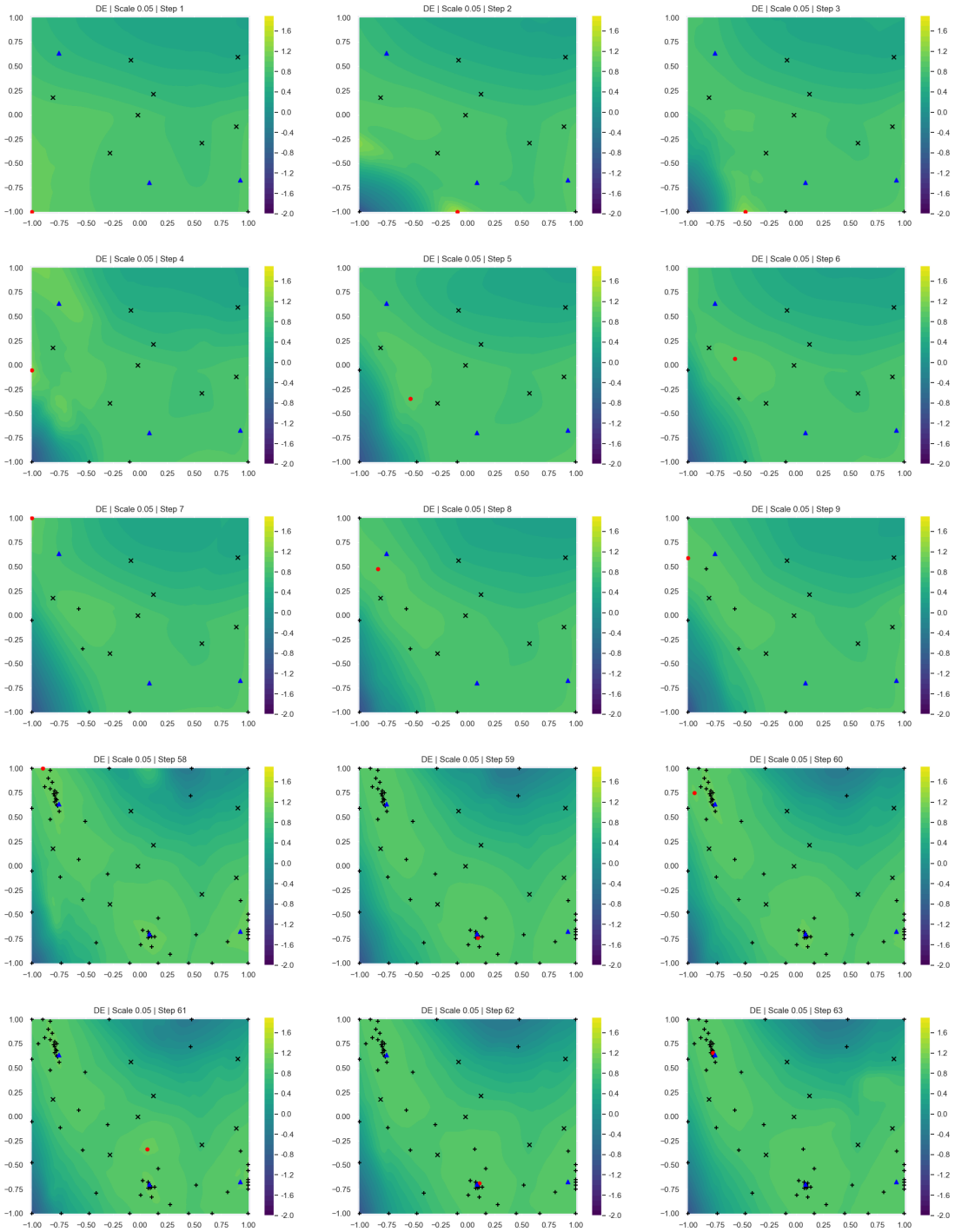


Figure A.19: **BO run for the MC Dropout method, run 30.** First 9 and last 6 steps during the Bayesian optimization of the Branin function using MC Dropout method with a mean width budget of 0.05.

Forrester

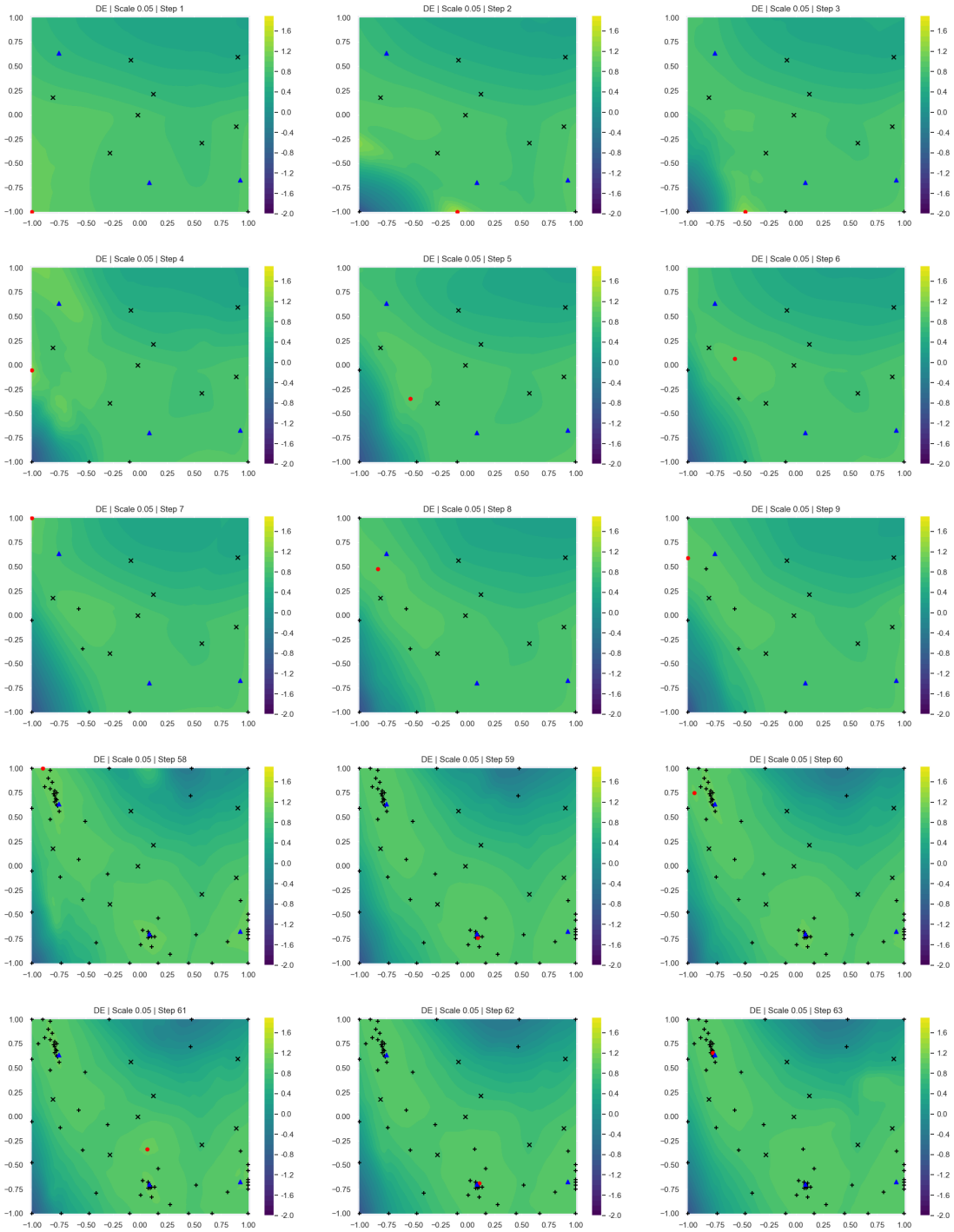


Figure A.20: **BO run for the Deep Ensemble method, run 30.** First 9 and last 6 steps during the Bayesian optimization of the Branin function using Deep Ensemble method with a mean width budget of 0.05.

Forrester

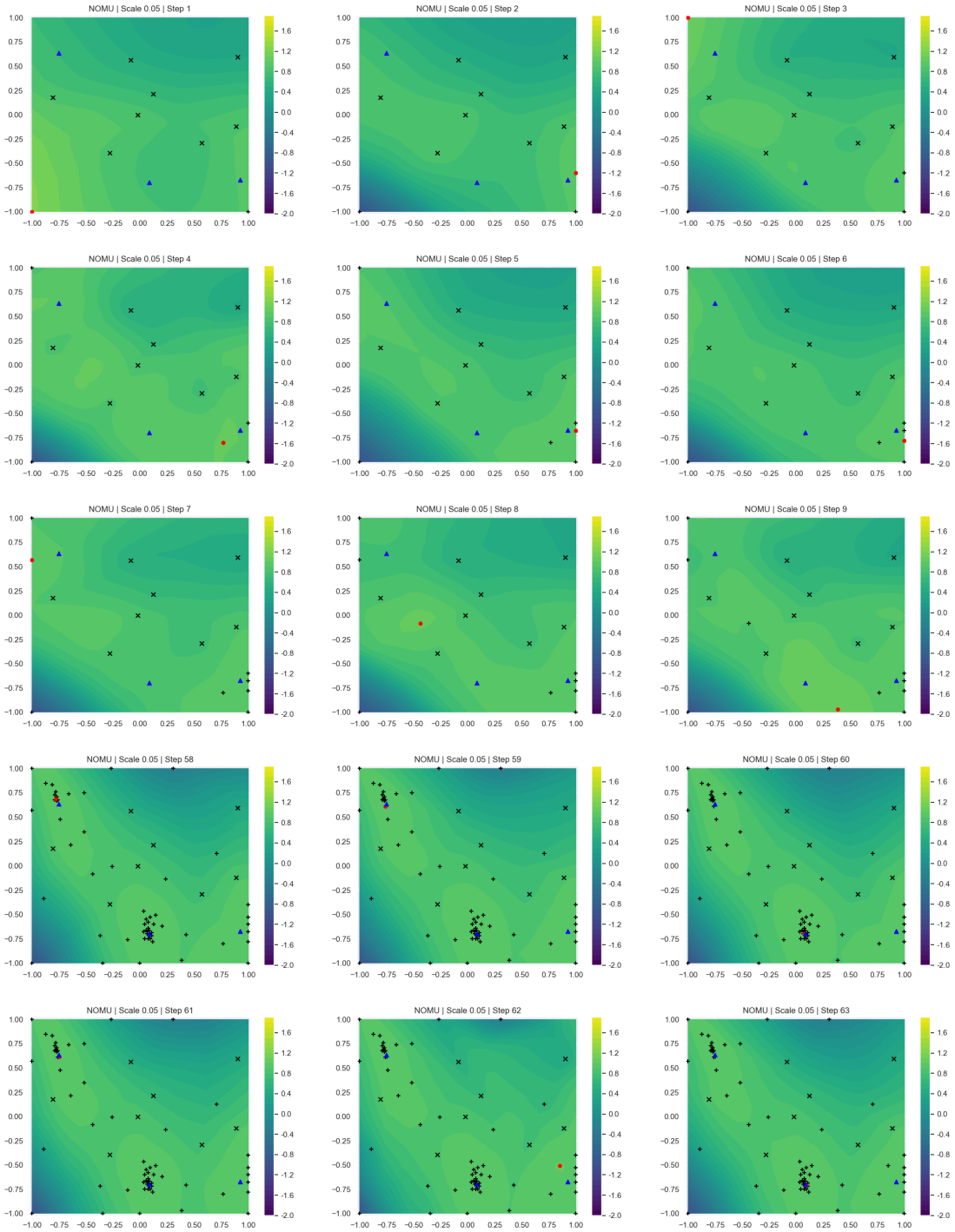
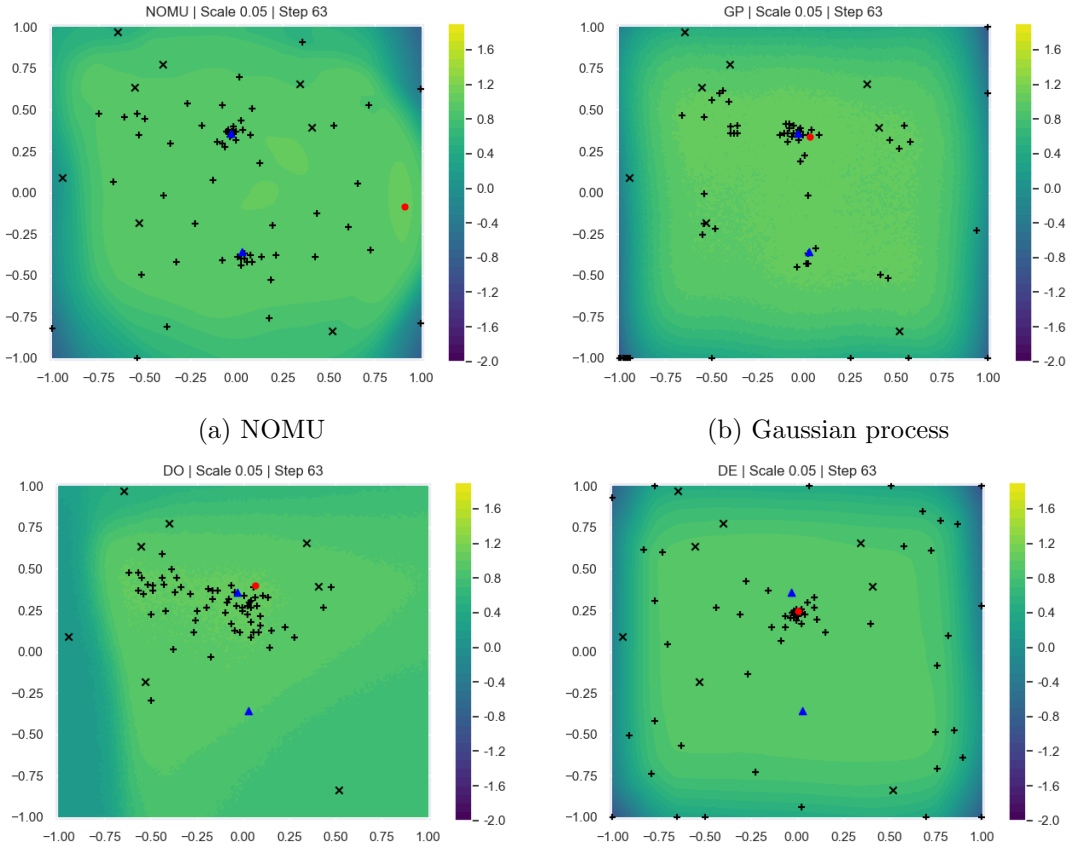


Figure A.21: **BO run for the pure Gaussian process, run 30.** First 9 and last 6 steps during the Bayesian optimization of the Branin function using the pure Gaussian process with a mean width budget of 0.05.



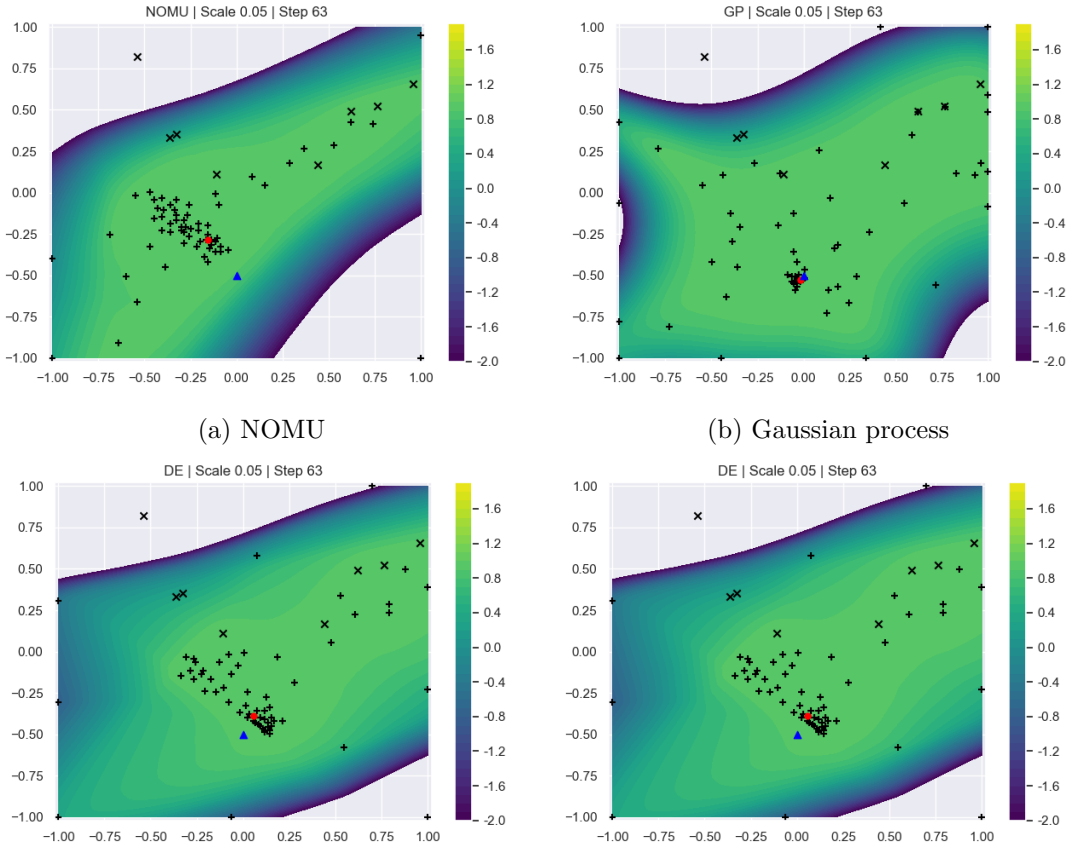
(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.22: **Final step of BO on Camelback function.** Large crosses indicate the starting samples and the plus signs show where it was sampled during the process. The blue triangle indicate the location of the optima and the red circle shows where the BO proposes to sample next. The contour displays the shape of the acquisition function.



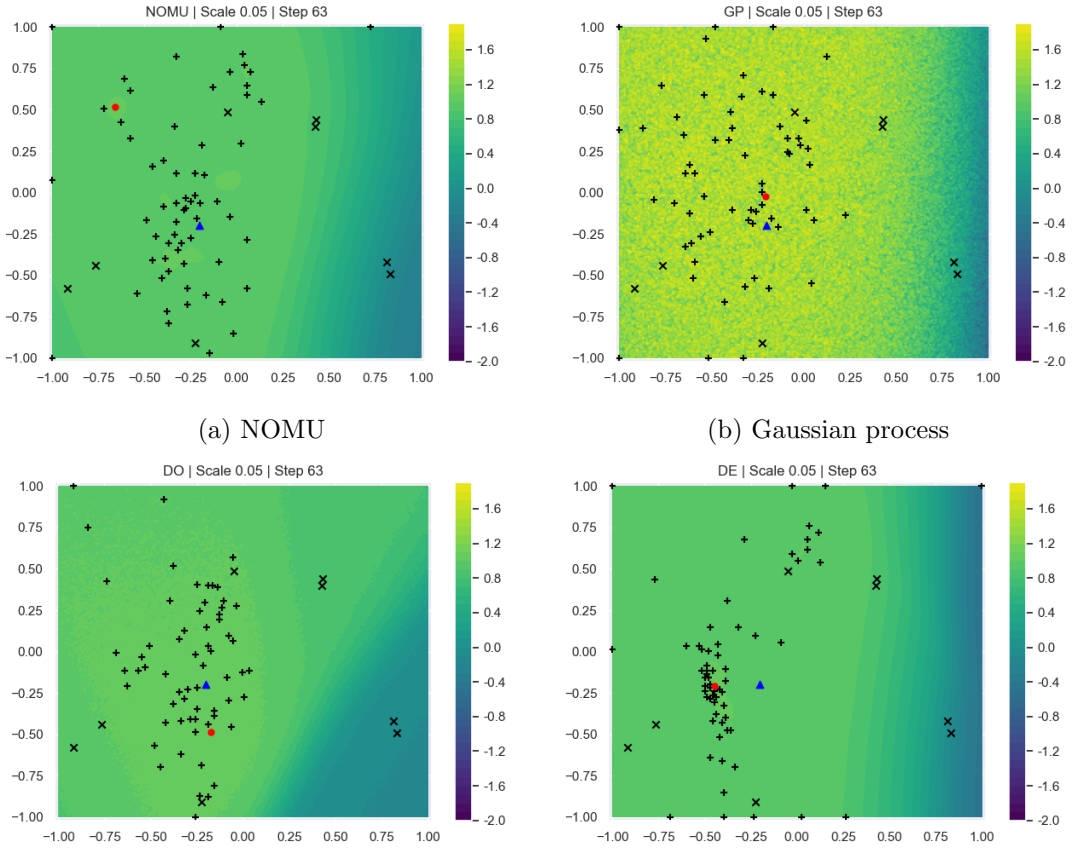
(a) NOMU

(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.23: **Final step of BO on Goldstein Price function.** Large crosses indicate the starting samples and the plus signs show where it was sampled during the process. The blue triangle indicate the location of the optima and the red circle shows where the BO proposes to sample next. The contour displays the shape of the acquisition function.



(a) NOMU

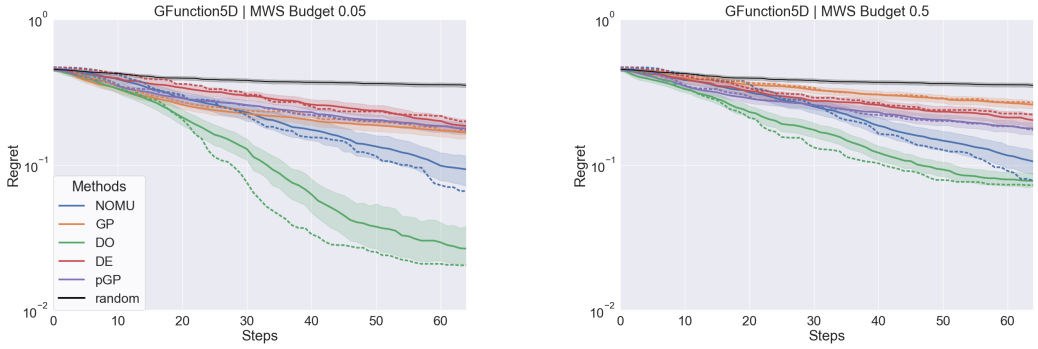
(b) Gaussian process

(c) MC Dropout

(d) Deep Ensembles

Figure A.24: **Final step of BO on Rosenbrock function.** Large crosses indicate the starting samples and the plus signs show where it was sampled during the process. The blue triangle indicate the location of the optima and the red circle shows where the BO proposes to sample next. The contour displays the shape of the acquisition function.

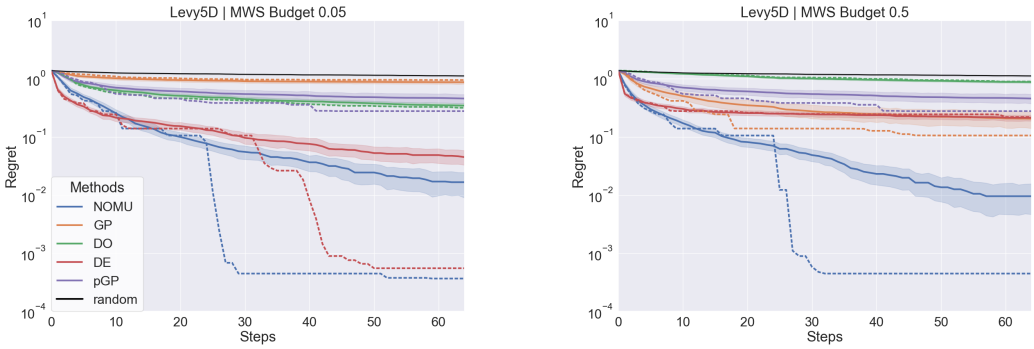
A.3 Multi-Dimensional Synthetic Functions Experiment



(a) mean width budget 0.05

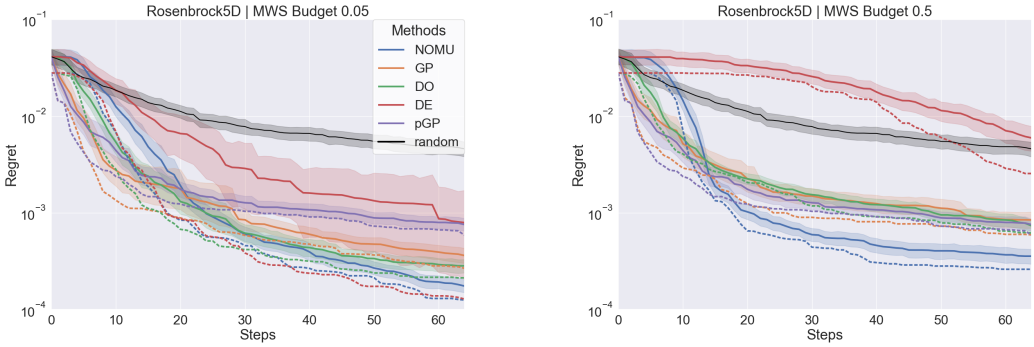
(b) mean width budget 0.5

Figure A.25: **Mean and median regret curve - G-function.** This graphs display the mean (solid lines) and median (dashed lines) final regret curve including confidence interval (colored area) for the different estimators. In (a) the results for a mean width budget of 0.05 are displayed and in (b) the ones for a budget of 0.5. Both graphs depict the situation for the G-FUNCTION



(a) mean width budget 0.05 (b) mean width budget 0.5

Figure A.26: **Mean and median regret curve - Levy.** This graphs display the mean (solid lines) and median (dashed lines) final regret curve including confidence interval (colored area) for the different estimators. In (a) the results for a mean width budget of 0.05 are displayed and in (b) the ones for a budget of 0.5. Both graphs depict the situation for the LEVY function



(a) mean width budget 0.05 (b) mean width budget 0.5

Figure A.27: **Mean and median regret curve - Rosenbrock.** This graphs display the mean (solid lines) and median (dashed lines) final regret curve including confidence interval (colored area) for the different estimators. In (a) the results for a mean width budget of 0.05 are displayed and in (b) the ones for a budget of 0.5. Both graphs depict the situation for the ROSENBROCK function

A.4 Implementation

A.4.1 General Structure

The implementation for this thesis is two fold. First, there is the core logic which is responsible for example for the different estimator methods, the acquisition functions or the Bayesian optimization algorithm. This core logic is implemented in a package structure which can be built into a "pip"-library. Second, there are all the different scripts which are needed to actually run the experiments or to analyze the results. These scripts are implemented as stand-alone Python-scripts. As a result the process of running and analyzing an experiments starts with defining the parameters to run the experiment with followed by running the executing script. The result of the run are saved to files. These results can then be enhanced by calculating further metrics using another script. Finally the results can be analyzed by plotting or listing the enhanced data using further scripts.

A.4.2 Design Decisions

The implementation of the core logic is highly modularized making use in inheritance to abstract as much logic as possible so that it can be recycled for subsequent approached with slightly different implementations. For structures which can be modified repeatedly like the acquisition functions which can be extended with mean width scaling or the kernels for the Gaussian process which can be combined the decorator pattern was used to allow a layered modification approach. For an easy experimental setup for all core modules interpreters were written so that the parameter for the sub processes can be specified using a single configuration file.

A.4.3 Testing

For the core logic of the package over 200 automated test were written to ensure that the code is actually executable and runs correctly.

List of Figures

List of Tables